



DOCA Documentation v2.7.0

Table of contents

DOCA SDK v2.7.0	42
NVIDIA DOCA Overview	42
NVIDIA DOCA Release Notes	48
BlueField and DOCA User Types	76
NVIDIA DOCA EULA	80
Quick Start for BlueField Developers	88
NVIDIA DOCA Developer Quick Start Guide	88
Installation and Setup	91
NVIDIA DOCA Profiles	91
NVIDIA DOCA Installation Guide for Linux	101
NVIDIA DOCA Developer Guide	155
DOCA Programming Guide	170
DOCA Programming Overview	170
DOCA Backward Compatibility Policy	171
DOCA Development Best Practices	173
DOCA Libraries	174
DOCA Utils	175
DOCA Drivers	175
DOCA Applications	176
NVIDIA DOCA Allreduce Application Guide	184
NVIDIA DOCA App Shield Agent Application Guide	205

NVIDIA DOCA DMA Copy Application Guide	221
NVIDIA DOCA DPA All-to-all Application Guide	234
NVIDIA DOCA DPA L2 Reflector Application Guide	250
NVIDIA DOCA East-West Overlay Encryption Application	261
NVIDIA DOCA Eth L2 Forwarding Application Guide	284
NVIDIA DOCA File Compression Application Guide	295
NVIDIA DOCA File Integrity Application Guide	307
NVIDIA DOCA GPU Packet Processing Application Guide	318
NVIDIA DOCA IPsec Security Gateway Application Guide	340
NVIDIA DOCA NAT Application Guide	378
NVIDIA DOCA PCC Application Guide	393
NVIDIA DOCA PSP Gateway Application Guide	414
NVIDIA DOCA Secure Channel Application Guide	442
NVIDIA DOCA Simple Forward VNF Application Guide	455
NVIDIA DOCA Switch Application Guide	470
NVIDIA DOCA UROM RDMO Application Guide	487
NVIDIA DOCA YARA Inspection Application Guide	510
DOCA Tools	527
NVIDIA DOCA Bench	528
NVIDIA DOCA Capabilities Print Tool	569
NVIDIA DOCA Comm Channel Admin Tool	589
NVIDIA DPA Tools	593
NVIDIA DOCA PCC Counter Tool	594

NVIDIA DOCA Socket Relay	598
DOCA Services	604
NVIDIA BlueField Container Deployment Guide	609
NVIDIA DOCA BlueMan Service Guide	624
NVIDIA DOCA Firefly Service Guide	631
NVIDIA DOCA Flow Inspector Service Guide	684
NVIDIA DOCA HBN Service Guide	699
NVIDIA DOCA Management Service Guide	844
NVIDIA DOCA Telemetry Service Guide	856
NVIDIA DOCA UROM Service Guide	894
DOCA Switching	901
OpenvSwitch Offload (OVS in DOCA)	903
VirtIO Acceleration through Hardware vDPA	905
Bridge Offload	911
Link Aggregation	912
Controlling Host PF and VF Parameters	923
API References	926
NVIDIA DOCA Driver APIs	926
NVIDIA DOCA Library APIs	926
Miscellaneous (Runtime)	927
NVIDIA DOCA Glossary	927
NVIDIA DOCA Crypto Acceleration	933
NVIDIA DOCA Services Fluent Logger	935

NVIDIA DOCA DPU CLI	938
NVIDIA DOCA Emulated Devices	943
NVIDIA BlueField Modes of Operation	972
NVIDIA DOCA with OpenSSL	985
NVIDIA BlueField DPU Scalable Function User Guide	989
NVIDIA TLS Offload Guide	1002
NVIDIA DOCA Troubleshooting Guide	1025
NVIDIA DOCA Virtual Functions User Guide	1058
Archives	1067
NVIDIA DOCA LTS Versions	1067
NVIDIA DOCA Documentation Archives	1068

List of Figures

Figure 0. Image 2024 3 18 11 7 56 1 Version 1 Modificationdate 1710752876623 Api V2

Figure 1. Image 2024 3 18 11 20 56 1 Version 2 Modificationdate 1711015642223 Api V2

Figure 2. Image 2024 5 5 13 30 16 Version 1 Modificationdate 1714905015383 Api V2

Figure 3. Doca As Software Framework Version 1 Modificationdate 1707498875940 Api V2

Figure 4. Doca Arch 2 Version 1 Modificationdate 1709736679783 Api V2

Figure 5. Doca Arch Version 1 Modificationdate 1709736679077 Api V2

Figure 6. Functional Isolation Version 1 Modificationdate 1709736679560 Api V2

Figure 7. Profiles

Figure 8. Developing Using Bluefield Setup Version 1 Modificationdate 1707815263100 Api V2

Figure 9. Developing Without Bluefield Setup Version 1 Modificationdate 1707815262023 Api V2

Figure 10. Cross Compilation From Host Diagram Version 1 Modificationdate 1707815262627 Api V2

Figure 11. Device Subsystem Version 1 Modificationdate 1702684190280 Api V2

Figure 12. Memory Subsystem Version 1 Modificationdate
1702684191093 Api V2

Figure 13. Execution Model Version 1 Modificationdate 1702684190873
Api V2

Figure 14. Doca Device Diagram Version 1 Modificationdate
1712136254887 Api V2

Figure 15. Device Discovery Version 1 Modificationdate 1712136252373
Api V2

Figure 16. Rep Device Discovery Version 1 Modificationdate
1713947539030 Api V2

Figure 17. Mmap Memrange Doca Version 1 Modificationdate
1712136260263 Api V2

Figure 18. Mmap Export Diagram Version 1 Modificationdate
1712136259710 Api V2

Figure 19. Doca Buf Version 1 Modificationdate 1712136253503 Api V2

Figure 20. Mmap Init Version 1 Modificationdate 1712136264907 Api V2

Figure 21. From Export Buf Flow Version 1 Modificationdate
1712136264603 Api V2

Figure 22. Execution Env Classes Version 1 Modificationdate
1712136264280 Api V2

Figure 23. Ctx Init Version 1 Modificationdate 1712136265113 Api V2

Figure 24. Doca Ctx 4 States Version 1 Modificationdate 1712136263617
Api V2

Figure 25. Doca Task Lifecycle Legend Version 1 Modificationdate
1712136267787 Api V2

Figure 26. Doca Task Lifecycle Alloc Init Submit Version 1
Modificationdate 1712136266273 Api V2

Figure 27. Doca Task Lifecycle Complet Free Version 1 Modificationdate
1712136267263 Api V2

Figure 28. Doca Task Lifecycle Complet Reuse Version 1
Modificationdate 1712136267473 Api V2

Figure 29. Doca Task Lifecycle Args Version 1 Modificationdate
1712136266560 Api V2

Figure 30. Workq Init Version 1 Modificationdate 1712136265387 Api V2

Figure 31. Doca Core Pe Poll Version 1 Modificationdate
1712136263323 Api V2

Figure 32. Doca Core Pe Wait Notify Version 1 Modificationdate
1712136263113 Api V2

Figure 33. Doca Core Event Simple Version 1 Modificationdate
1712136262360 Api V2

Figure 34. Workq Error Version 1 Modificationdate 1712136265993 Api
V2

Figure 35. Doca Sync Event Components Diagram Version 1
Modificationdate 1711961903363 Api V2

Figure 36. Doca Sync Event Interaction Diagram Version 1
Modificationdate 1711961902633 Api V2

Figure 37. Sync Event Version 1 Modificationdate 1711961904130 Api
V2

Figure 38. Remote Net Export Flow Version 1 Modificationdate
1711961903650 Api V2

Figure 39. Architecture Diagram Version 1 Modificationdate 1707724249240 Api V2

Figure 40. Domains D1 Version 1 Modificationdate 1707724249723 Api V2

Figure 41. Domains D2 Version 1 Modificationdate 1707724249940 Api V2

Figure 42. Rfc2697 Version 1 Modificationdate 1707724244973 Api V2

Figure 43. Rfc2698 Version 1 Modificationdate 1707724245433 Api V2

Figure 44. Rfc4115 Version 1 Modificationdate 1707724245187 Api V2

Figure 45. Vnf Mode Diagram Version 1 Modificationdate 1707724249023 Api V2

Figure 46. Switch Mode Diagram Version 1 Modificationdate 1707724248740 Api V2

Figure 47. Image 2024 4 22 11 40 1 Version 2 Modificationdate 1714510012400 Api V2

Figure 48. Remote Vnf Mode Diagram Version 1 Modificationdate 1707724248433 Api V2

Figure 49. Pipe Illustration Version 1 Modificationdate 1707724248183 Api V2

Figure 50. Matching Diagram Version 1 Modificationdate 1707724245743 Api V2

Figure 51. Pipe Entry Queue Diagram Version 1 Modificationdate 1707724247967 Api V2

Figure 52. Miss Pipe Hw Table Structure Version 1 Modificationdate 1707724246460 Api V2

Figure 53. Packet Processing No Flow Version 1 Modificationdate
1707724247700 Api V2

Figure 54. Packet Processing W Flow Version 1 Modificationdate
1707724247437 Api V2

Figure 55. Arch Diagram Version 1 Modificationdate 1707982858940
Api V2

Figure 56. Autonomous Mode Diagram Version 1 Modificationdate
1707982858717 Api V2

Figure 57. Managed Mode Diagram Version 1 Modificationdate
1707982858340 Api V2

Figure 58. Image 2024 4 25 13 35 46 Version 2 Modificationdate
1714534639683 Api V2

Figure 59. Dpa Memory Hierarchy Diagram Version 1 Modificationdate
1704292970253 Api V2

Figure 60. Different Processes In System Version 1 Modificationdate
1710697862863 Api V2

Figure 61. Signed User Dpa Code Version 1 Modificationdate
1710697862613 Api V2

Figure 62.
Be2ad944364a59626dfbec77704b8a73946f1d8feac5bf2bc4d0530169a0

Figure 63. Rot Certificate Chain Including Nvidia Root And Customer
Certificate Chain Version 1 Modificationdate 1710697861247 Api V2

Figure 64. Elf File Structure Schematic Version 1 Modificationdate
1710697860663 Api V2

Figure 65. Signing Flow Version 1 Modificationdate 1710697860390 Api V2

Figure 66. Elf Cryptographic Data Section Layout Version 1 Modificationdate 1710697860080 Api V2

Figure 67. Hash Fields Big Endian Bytes Alignment Version 1 Modificationdate 1710697858933 Api V2

Figure 68. Image 2023 10 5 13 11 31 Version 1 Modificationdate 1711338004780 Api V2

Figure 69. Basic Initiator Target Version 1 Modificationdate 1712771078677 Api V2

Figure 70. Advanced Initiator Target Version 1 Modificationdate 1712777074427 Api V2

Figure 71. Host Dpu Library And Header Files Version 1 Modificationdate 1709127880633 Api V2

Figure 72. Image 2024 2 28 16 46 43 Version 1 Modificationdate 1709131603210 Api V2

Figure 73. Image 2024 2 28 16 42 13 Version 1 Modificationdate 1709131333427 Api V2

Figure 74. Image 2024 2 28 16 45 6 Version 1 Modificationdate 1709131506547 Api V2

Figure 75. Development Flow Version 2 Modificationdate 1709680085257 Api V2

Figure 76. Sys Design Version 2 Modificationdate 1709679022597 Api V2

Figure 77. DMA Local Copy Version 1 Modificationdate 1703662233270
Api V2

Figure 78. DMA Remote On DPU Version 1 Modificationdate
1703662233687 Api V2

Figure 79. DMA Remote On Host Version 1 Modificationdate
1703662233883 Api V2

Figure 80. Consumers Producers Version 3 Modificationdate
1713085812843 Api V2

Figure 81. Client Server Connection Version 1 Modificationdate
1713090420807 Api V2

Figure 82. Consumer Creation Flow Version 4 Modificationdate
1714485658890 Api V2

Figure 83. Msgq Overview Version 2 Modificationdate 1714619794027
Api V2

Figure 84. Establishing Connection Version 1 Modificationdate
1705079355063 Api V2

Figure 85. Urom Deployment Version 1 Modificationdate
1712872977443 Api V2

Figure 86. Image 2024 3 5 14 28 16 1 Version 1 Modificationdate
1709641694263 Api V2

Figure 87. Image 2024 3 5 14 58 49 Version 1 Modificationdate
1709643527180 Api V2

Figure 88. Doca Urom Library Components Version 1 Modificationdate
1712874399770 Api V2

Figure 89. Doca Urom Headers Version 1 Modificationdate
1712874552090 Api V2

Figure 90. Image 2024 4 15 18 53 9 1 Version 2 Modificationdate
1714617643890 Api V2

Figure 91.

Figure 92. Regular Receive Version 1 Modificationdate 1711355755860
Api V2

Figure 93. Cyclic Receive Version 1 Modificationdate 1711355756397
Api V2

Figure 94. Managed Memory Pool Receive Version 1 Modificationdate
1711355756950 Api V2

Figure 95. Working With Doca Flow Version 1 Modificationdate
1711355757257 Api V2

Figure 96. Regular Send Version 1 Modificationdate 1711355757910 Api
V2

Figure 97. Doca Eth Context Version 1 Modificationdate 1711355758403
Api V2

Figure 98. Image 2023 10 12 8 40 19 Version 1 Modificationdate
1711355750297 Api V2

Figure 99. Image2023 3 17 17 16 6 Version 1 Modificationdate
1708331893283 Api V2

Figure 100. Image2023 4 19 11 47 9 Version 1 Modificationdate
1708331896880 Api V2

Figure 101. Application On Host Cpu Diagram Version 1
Modificationdate 1708331891763 Api V2

Figure 102. Application On Dpu Converged Arm Cpu Diagram Version 1
Modificationdate 1708331891473 Api V2

Figure 103. Image 2024 1 12 12 24 42 Version 1 Modificationdate
1708331892010 Api V2

Figure 104. Image2023 4 18 12 6 25 Version 1 Modificationdate
1708331896663 Api V2

Figure 105. Image2023 4 18 11 49 40 Version 1 Modificationdate
1708331896383 Api V2

Figure 106. Image2023 4 3 18 18 20 Version 1 Modificationdate
1708331894060 Api V2

Figure 107. Image2023 4 4 12 13 32 Version 1 Modificationdate
1708331894257 Api V2

Figure 108. Image 2024 4 17 12 29 48 Version 1 Modificationdate
1713349788677 Api V2

Figure 109. Image 2024 3 26 15 19 21 Version 1 Modificationdate
1711462762163 Api V2

Figure 110. Dst Buf Src Buf Version 1 Modificationdate 1712750270287
Api V2

Figure 111. Sha Arch Diagram Version 1 Modificationdate
1702684755897 Api V2

Figure 112. Erasure Coding Transmission Version 1 Modificationdate
1707749454753 Api V2

Figure 113. Screenshot 2022 11 30 120911 Version 1 Modificationdate
1707749454193 Api V2

Figure 114. Screenshot 2022 11 24 101000 Version 1 Modificationdate
1707749453907 Api V2

Figure 115. Screenshot 2022 11 24 101058 Version 1 Modificationdate
1707749454473 Api V2

Figure 116.
87de05bf19da2337dcc4bee8c38d3d8cdd0b8c9bcf0d11026d110e1e347f

Figure 117.
848782870816eb781cba2690b24466dcbf20424df7c8be35ce154a46d3ac

Figure 118.
0963e85956a2487917c1c97b7e89644e0eb74703a28b3c776e7632ccf5d8

Figure 119.
Ff337263f67750682802455daa071f397667f901705ff3b3d668f82428f0b3

Figure 120.
436491c2113325308d67ec5aa37b6d07c6555a1ae88093dca5cc5f0411c8

Figure 121.
711108e36acb254c43c35ef7a01651e0d93a002c4808d5e2c7d876dc0b58

Figure 122.
848782870816eb781cba2690b24466dcbf20424df7c8be35ce154a46d3ac

Figure 123.
0963e85956a2487917c1c97b7e89644e0eb74703a28b3c776e7632ccf5d8

Figure 124. Image 2023 10 19 13 47 37 1 Version 1 Modificationdate
1704287901360 Api V2

Figure 125. Image 2023 10 19 13 47 47 1 Version 1 Modificationdate
1704287901137 Api V2

Figure 126. Image 2023 10 19 13 47 54 1 Version 1 Modificationdate 1704287900770 Api V2

Figure 127. TelemetryAgent Programmer Guide Version 1 Modificationdate 1710836657377 Api V2

Figure 128. Doca Schema Version 1 Modificationdate 1710836655790 Api V2

Figure 129. App Development Steps Version 1 Modificationdate 1710836656197 Api V2

Figure 130. Doca Device Emulation Version 1 Modificationdate 1713283959787 Api V2

Figure 131. Hotplug State Machine Version 2 Modificationdate 1714853926697 Api V2

Figure 132. PCI Header Version 2 Modificationdate 1713881072843 Api V2

Figure 133. Bar Overview Version 2 Modificationdate 1713885085617 Api V2

Figure 134. Io Mapped Version 2 Modificationdate 1713885189550 Api V2

Figure 135. Memory Mapped Version 2 Modificationdate 1713885236747 Api V2

Figure 136. Bar Region Overview Version 3 Modificationdate 1713888421243 Api V2

Figure 137. Stateful Region Read Version 3 Modificationdate 1713888488460 Api V2

Figure 138. Stateful Region Write Version 2 Modificationdate
1713888554040 Api V2

Figure 139. Stateful Default Values Version 3 Modificationdate
1713891703973 Api V2

Figure 140. DB Region Overview Version 3 Modificationdate
1713892226143 Api V2

Figure 141. DB Region By Offset Version 2 Modificationdate
1713893153033 Api V2

Figure 142. DB Region By Data Version 2 Modificationdate
1713893266403 Api V2

Figure 143. DOCA VFS Progress Devzone Version 2 Modificationdate
1715027248360 Api V2

Figure 144. DOCA Virtio FS IO Path Devzone Version 2 Modificationdate
1715027913957 Api V2

Figure 145. Architecture Diagram Version 1 Modificationdate
1705079608807 Api V2

Figure 146. Image2022 7 26 10 29 13 Version 1 Modificationdate
1715011248327 Api V2

Figure 147. Image2022 7 28 14 58 53 Version 1 Modificationdate
1715011248013 Api V2

Figure 148. Image2022 7 28 15 0 34 Version 1 Modificationdate
1715011246873 Api V2

Figure 149. Image2022 7 28 14 59 57 Version 1 Modificationdate
1715011247197 Api V2

Figure 150. Image2022 7 28 15 1 11 Version 1 Modificationdate
1715011246450 Api V2

Figure 151. Image2022 7 28 15 1 42 Version 1 Modificationdate
1715011246140 Api V2

Figure 152. Image2022 7 28 15 2 14 Version 1 Modificationdate
1715011245607 Api V2

Figure 153. Image2022 7 28 15 4 11 Version 1 Modificationdate
1715011245140 Api V2

Figure 154. Image2022 7 28 15 6 40 Version 1 Modificationdate
1715011244377 Api V2

Figure 155. Procedure Heading Icon Version 1 Modificationdate
1715011276433 Api V2

Figure 156. Procedure Heading Icon Version 1 Modificationdate
1715011276433 Api V2

Figure 157. Procedure Heading Icon Version 1 Modificationdate
1715011276433 Api V2

Figure 158. Procedure Heading Icon Version 1 Modificationdate
1715011276433 Api V2

Figure 159. Procedure Heading Icon Version 1 Modificationdate
1715011276433 Api V2

Figure 160. Procedure Heading Icon Version 1 Modificationdate
1715011287687 Api V2

Figure 161. Procedure Heading Icon Version 1 Modificationdate
1715011287687 Api V2

Figure 162. Procedure Heading Icon Version 1 Modificationdate
1715011287687 Api V2

Figure 163. Procedure Heading Icon Version 1 Modificationdate
1715011287687 Api V2

Figure 164. Worddavb2ee67a7eb9aae5c536610e39a37dcc5 Version 1
Modificationdate 1715011292287 Api V2

Figure 165. Worddav6931c32564b3b0c166f4a26788219144 Version 1
Modificationdate 1715011293317 Api V2

Figure 166. Procedure Heading Icon Version 1 Modificationdate
1715011293790 Api V2

Figure 167. Image2019 3 8 12 50 6 Version 1 Modificationdate
1715011294230 Api V2

Figure 168. Procedure Heading Icon Version 1 Modificationdate
1715011293790 Api V2

Figure 169. Procedure Heading Icon Version 1 Modificationdate
1715011293790 Api V2

Figure 170. Procedure Heading Icon Version 1 Modificationdate
1715011293790 Api V2

Figure 171. Procedure Heading Icon Version 1 Modificationdate
1715011293790 Api V2

Figure 172. Procedure Heading Icon Version 1 Modificationdate
1715011293790 Api V2

Figure 173. Procedure Heading Icon Version 1 Modificationdate
1715011293790 Api V2

Figure 174. Procedure Heading Icon Version 1 Modificationdate
1715011293790 Api V2

Figure 175. Procedure Heading Icon Version 1 Modificationdate
1715011293790 Api V2

Figure 176. Procedure Heading Icon Version 1 Modificationdate
1715011293790 Api V2

Figure 177. Procedure Heading Icon Version 1 Modificationdate
1715011306180 Api V2

Figure 178. Worddav336f9b6791fd85e08c8e6897697cd75b Version 1
Modificationdate 1715011307170 Api V2

Figure 179. Procedure Heading Icon Version 1 Modificationdate
1715011308847 Api V2

Figure 180. Procedure Heading Icon Version 1 Modificationdate
1715011308847 Api V2

Figure 181. Procedure Heading Icon Version 1 Modificationdate
1715011308847 Api V2

Figure 182. System Design Diagram Version 1 Modificationdate
1707753408817 Api V2

Figure 183. Offloaded Diagram Version 1 Modificationdate
1707753409247 Api V2

Figure 184. Non Offloaded Diagram Version 1 Modificationdate
1707753409467 Api V2

Figure 185. Offloaded Arch Version 1 Modificationdate 1707753409747
Api V2

Figure 186. Non Offloaded Arch Version 1 Modificationdate
1707753410017 Api V2

Figure 187. High Level Diagram Version 1 Modificationdate
1707756008163 Api V2

Figure 188. App Shield Arch Version 1 Modificationdate 1707756008450
Api V2

Figure 189. System Design Diagram Version 1 Modificationdate
1707755881397 Api V2

Figure 190. Application Architecture Diagram Version 1
Modificationdate 1707755881060 Api V2

Figure 191. System Design Diagram Version 1 Modificationdate
1707755743853 Api V2

Figure 192. All To All Non Blocking Version 1 Modificationdate
1707755742537 Api V2

Figure 193. System Design Diagram Version 1 Modificationdate
1707755629077 Api V2

Figure 194. Architecture Diagram Version 1 Modificationdate
1707755629413 Api V2

Figure 195. System Design Diagram Version 1 Modificationdate
1707755570657 Api V2

Figure 196. Application Architecture Diagram Version 1
Modificationdate 1707755571000 Api V2

Figure 197. Application Architecture Diagram Version 1
Modificationdate 1707755571000 Api V2

Figure 198. Image 2024 4 24 19 22 9 1 Version 1 Modificationdate 1715184477330 Api V2

Figure 199. Image 2024 4 25 18 4 10 1 Version 1 Modificationdate 1715184477773 Api V2

Figure 200. Image 2024 4 28 12 3 37 1 Version 1 Modificationdate 1715184478247 Api V2

Figure 201. Sys Design Version 1 Modificationdate 1707755427027 Api V2

Figure 202. App Arch Version 1 Modificationdate 1707755427900 Api V2

Figure 203. Sys Design Version 1 Modificationdate 1707755294790 Api V2

Figure 204. App Arch Version 1 Modificationdate 1707755295213 Api V2

Figure 205. Image 2024 1 12 12 37 6 Version 1 Modificationdate 1707755033600 Api V2

Figure 206. Image2023 4 11 10 33 44 Version 1 Modificationdate 1707755034360 Api V2

Figure 207. Image 2024 1 12 12 48 41 Version 1 Modificationdate 1707755033397 Api V2

Figure 208. Image 2024 1 12 13 0 40 Version 1 Modificationdate 1707755032807 Api V2

Figure 209. Image2023 4 11 12 30 28 Version 1 Modificationdate 1707755035100 Api V2

Figure 210. Ipcsec Mode Diagrams With Encryption Version 1 Modificationdate 1707754703443 Api V2

Figure 211. Image2023 4 2 12 21 37 Version 1 Modificationdate
1707754704347 Api V2

Figure 212. Image2023 4 2 12 22 22 Version 1 Modificationdate
1707754704607 Api V2

Figure 213. Image2023 4 2 12 24 42 Version 2 Modificationdate
1714616563293 Api V2

Figure 214. App Arch Dyn 2.7 Version 2 Modificationdate
1714616477517 Api V2

Figure 215. Vnf Pipes Enc 2.7 Version 1 Modificationdate
1714553673913 Api V2

Figure 216. Vnf Pipes Dec 2.7 Version 2 Modificationdate
1714616795657 Api V2

Figure 217. Switch Pipes 2.7 Version 1 Modificationdate 1714553981323
Api V2

Figure 218. END2END Version 1 Modificationdate 1707754705857 Api
V2

Figure 219. System Design Diagram Version 1 Modificationdate
1707754532033 Api V2

Figure 220. Static Mode Diagram Version 1 Modificationdate
1707754531803 Api V2

Figure 221. Dynamic Mode Diagram Version 1 Modificationdate
1707754531530 Api V2

Figure 222. Nat Offload Diagram Version 1 Modificationdate
1707754531173 Api V2

Figure 223. Sys Design Version 2 Modificationdate 1709680527063 Api V2

Figure 224. Image 2024 4 29 10 3 46

Figure 225. Image 2024 5 6 15 33 24 Version 2 Modificationdate 1715030824903 Api V2

Figure 226. Image 2024 4 23 14 21 16 Version 1 Modificationdate 1713900077087 Api V2

Figure 227. Image 2024 4 23 14 24 13 Version 1 Modificationdate 1713900253237 Api V2

Figure 228. Image 2024 4 23 14 50 36 Version 2 Modificationdate 1713993780690 Api V2

Figure 229. Image 2024 4 23 15 12 1 Version 2 Modificationdate 1713993897973 Api V2

Figure 230. Image 2024 4 23 15 30 41 Version 2 Modificationdate 1713993941697 Api V2

Figure 231. Image 2024 4 23 15 41 2 Version 2 Modificationdate 1713994016437 Api V2

Figure 232. Sys Design Version 1 Modificationdate 1707754226147 Api V2

Figure 233. Application Architecture Diagram Version 1 Modificationdate 1707754226610 Api V2

Figure 234. System Design Diagram Version 1 Modificationdate 1707754043223 Api V2

Figure 235. Initialization Process Illustration Version 1 Modificationdate 1707754042977 Api V2

Figure 236. Packet Processing Illustration Version 1 Modificationdate
1707754042603 Api V2

Figure 237. System Design Diagram 2 Version 1 Modificationdate
1707753872607 Api V2

Figure 238. System Design Diagram 1 Version 1 Modificationdate
1707753872300 Api V2

Figure 239. Application Architecture Diagram Version 1
Modificationdate 1707753872820 Api V2

Figure 240. Image 2024 3 10 10 56 44 1 Version 2 Modificationdate
1714279920303 Api V2

Figure 241. Image 2024 3 11 9 16 2 1 Version 2 Modificationdate
1713994699097 Api V2

Figure 242. Image 2024 3 10 11 42 4 1 Version 2 Modificationdate
1714280000753 Api V2

Figure 243. Image 2024 3 10 15 19 42 1 Version 2 Modificationdate
1713994773310 Api V2

Figure 244. Image 2024 3 11 8 46 48 1 Version 3 Modificationdate
1714280058960 Api V2

Figure 245. Image 2024 3 11 8 55 18 1 Version 2 Modificationdate
1714280253600 Api V2

Figure 246. System Design Diagram Version 1 Modificationdate
1707753636250 Api V2

Figure 247. Application Architecture Diagram Version 1
Modificationdate 1707753636607 Api V2

Figure 248. Image 2024 4 24 14 24 10 Version 2 Modificationdate
1714779249390 Api V2

Figure 249. Image 2024 4 16 12 47 57 Version 1 Modificationdate
1713268077330 Api V2

Figure 250. Image 2024 4 16 12 47 9 Version 1 Modificationdate
1713268029407 Api V2

Figure 251. Dpacc Offloading Version 1 Modificationdate
1702686392883 Api V2

Figure 252. DPACC Output Version 1 Modificationdate 1702686388197
Api V2

Figure 253. Dpa Lib Version 1 Modificationdate 1702686389523 Api V2

Figure 254. Dpacc Trajectory Version 1 Modificationdate
1702686393177 Api V2

Figure 255. Partition Control Diagram Version 1 Modificationdate
1704380748793 Api V2

Figure 256. Image 2024 1 25 12 27 13 Version 1 Modificationdate
1709237164957 Api V2

Figure 257. Image2023 4 9 16 37 54 Version 1 Modificationdate
1702686672617 Api V2

Figure 258. Containers Overview Version 1 Modificationdate
1707750414520 Api V2

Figure 259. Deployment Architecture Version 1 Modificationdate
1707750414103 Api V2

Figure 260. Blueman Health Version 1 Modificationdate 1702686893613
Api V2

Figure 261. Blueman Login Version 1 Modificationdate 1705445968150
Api V2

Figure 262. Arch Diagram Version 1 Modificationdate 1707750050417
Api V2

Figure 263. Monitor Arch Version 1 Modificationdate 1707750050000
Api V2

Figure 264. Flow Inspector Service Arch Version 1 Modificationdate
1702686961310 Api V2

Figure 265. Flow Of Service Graph Version 1 Modificationdate
1702686960717 Api V2

Figure 266. Hbn Architecture Version 1 Modificationdate
1710231732767 Api V2

Figure 267. System Overview Version 1 Modificationdate
1710231730197 Api V2

Figure 268. Image 2023 12 7 14 8 20 Version 1 Modificationdate
1710231732413 Api V2

Figure 269. Hbn Sfc Cni Hbn Interfaces Version 2 Modificationdate
1714756315320 Api V2

Figure 270. DVNI Fnn Version 5 Modificationdate 1715199920240 Api
V2

Figure 271. Gateway4 Version 2 Modificationdate 1715201359927 Api
V2

Figure 272. Screenshot 2024 04 07 095621 Version 3 Modificationdate
1714578008037 Api V2

Figure 273. Screenshot 2024 04 07 095501 Version 3 Modificationdate 1714578149817 Api V2

Figure 274. Doca Telemetry Service Overview Version 1 Modificationdate 1709738068343 Api V2

Figure 275. Configuration Diagram Version 1 Modificationdate 1709738068767 Api V2

Figure 276. Data Sources Version 1 Modificationdate 1709738071027 Api V2

Figure 277. Url Address Version 1 Modificationdate 1709738069077 Api V2

Figure 278. Explore Version 1 Modificationdate 1709738070393 Api V2

Figure 279. Grapgh Version 1 Modificationdate 1709738070137 Api V2

Figure 280. Image 2024 3 7 12 16 1 Version 1 Modificationdate 1709806560297 Api V2

Figure 281. Kernel Representors Model Version 1 Modificationdate 1715004628570 Api V2

Figure 282. Virtio Pfs Version 1 Modificationdate 1707421048803 Api V2

Figure 283. Virtio Vfs Version 1 Modificationdate 1707421049047 Api V2

Figure 284. Virtio Vf Pcie Devices For Vhost Acceleration Version 1 Modificationdate 1702447159680 Api V2

Figure 285. Vdpa Over Virtio Full Emulation Design Version 1 Modificationdate 1702390976047 Api V2

Figure 286. Embedded Mode Version 1 Modificationdate 1707758625503 Api V2

Figure 287. Bluefield Internal Cpu Configuration Version 1
Modificationdate 1714278946250 Api V2

Figure 288. Internal Cpu Offload Engine Version 1 Modificationdate
1714278947750 Api V2

Figure 289. Nic Mode Version 1 Modificationdate 1714278948610 Api
V2

Figure 290. Openssl Architecture Version 1 Modificationdate
1702687347637 Api V2

Figure 291. Scalable Functions Illustration Version 1 Modificationdate
1713970886143 Api V2

Figure 292. Sf Steps Version 1 Modificationdate 1713970887350 Api V2

Figure 293. Running Application Over Sf Version 1 Modificationdate
1713970887703 Api V2

Figure 294. Tls Setup Diagram Version 1 Modificationdate
1702687400643 Api V2

Figure 295. Tls Testing Setup Diagram Version 1 Modificationdate
1702687400027 Api V2

Figure 296. Eswitch Topology Version 1 Modificationdate
1714765863767 Api V2

DOCA Documentation v2.7.0

DOCA Overview

[This page](#) provides an overview of the structure of NVIDIA DOCA documentation.

Release Notes

[This page](#) contains information on new features, bug fixes, and known issues.

User Types

[This page](#) provides a quick introduction to the NVIDIA® BlueField® family of networking platforms (i.e., DPUs and SuperNICs), its DOCA software components, and BlueField user types.

NVIDIA DOCA EULA

[This page](#) provides the NVIDIA DOCA SDK end-user license agreement.

Quick Start

Developer Quick Start Guide

[This page](#) details the basic steps to bring up the NVIDIA DOCA development environment and to build and run the DOCA reference applications provided along with the DOCA

software framework package.

Installation and Setup

Profiles

[This page](#) provides an introduction to the various supported DOCA profiles.

Installation Guide for Linux

[This page](#) details the necessary steps to set up NVIDIA DOCA in your Linux environment.

Developer Guide

[This page](#) details the recommended steps to set up an NVIDIA DOCA development environment.

DOCA Programming Guides

[These pages](#) are intended for developers wishing to utilize DOCA SDK to develop application on top of NVIDIA® BlueField® networking platforms.

Applications

[This page](#) provides an overview of the example DOCA applications implemented on top of NVIDIA® BlueField®.

Allreduce

[This page](#) provides a DOCA Allreduce collective operation implementation on top of NVIDIA® BlueField® using UCX.

App Shield Agent

[This page](#) provides process introspection system implementation on top of NVIDIA® BlueField®.

DMA Copy

[This page](#) provides an example of a DMA Copy implementation on top of NVIDIA® BlueField®.

DPA All-to-all

[This page](#) explains the all-to-all collective operation example when accelerated using the DPA in NVIDIA® BlueField®-3.

DPA L2 Reflector

[This page](#) provides an L2 reflector implementation on top of the NVIDIA® BlueField®-3.

East-west Overlay Encryption

[This page](#) describes IPsec based strongSwan solution on top of NVIDIA® BlueField®.

File Compression

[This page](#) provides a file compression implementation on top of the NVIDIA® BlueField®.

File Integrity

[This page](#) provides a file integrity implementation on top of NVIDIA® BlueField®.

GPU Packet Processing

[This page](#) provides a description of the GPU packet processing application to demonstrate using the DOCA GPUNetIO, DOCA Ethernet, and DOCA Flow libraries to implement a GPU traffic analyzer.

IPsec Security Gateway

[This page](#) provides an IPsec security gateway implementation on top of NVIDIA® BlueField®.

NAT

[This page](#) provides a NAT implementation on top of NVIDIA® BlueField®.

PCC

[This page](#) provides a DOCA PCC implementation on top of NVIDIA® BlueField®.

PSP Gateway

[This page](#) describes the usage of the NVIDIA DOCA PSP Gateway sample application on top of an NVIDIA® BlueField® networking platform or NVIDIA® ConnectX® SmartNIC.

Secure Channel

[This page](#) provides a secure channel implementation on top of NVIDIA® BlueField®.

Simple Forward VNF

[This page](#) provides a Simple Forward implementation on top of NVIDIA® BlueField®.

Switch

[This page](#) provides an example of switch implementation on top of NVIDIA® BlueField® .

UROM RDMO

[This page](#) provides a DOCA Remote Direct Memory Operation implementation on top of NVIDIA® BlueField® using Unified Communication X (UCX) . .

YARA Inspection

[This page](#) provides YARA inspection implementation on top of NVIDIA® BlueField®.

Tools

[This page](#) provides an overview of the set of tools provided by DOCA and their purpose.

DOCA Bench

[This page](#) describes a tool which allows users to evaluate the performance of DOCA applications, with reasonable accuracy for real-world applications.

Capabilities Print Tool

[This page](#) provides instruction on the usage of the DOCA Capabilities Print Tool.

Comm Channel Admin Tool

[This page](#) provides instructions on the usage of the DOCA Comm Channel Admin Tool.

DPA Tools

[This page](#) lists a set of executables that enable the DPA application developer and the system administrator to manage and monitor DPA resources and to debug DPA applications.

PCC Counter Tool

[This page](#) provides instruction on the usage of the PCC Counter tool.

Socket Relay

[This page](#) describes DOCA Socket Relay architecture, usage, etc.

DOCA Services

[This page](#) provides an overview of the set of services provided by DOCA and their purpose.

Container Deployment

[This page](#) provides an overview and deployment configuration of DOCA containers for NVIDIA® BlueField®.

DOCA BlueMan Service

[This page](#) provides instructions on how to use the DOCA BlueMan service on top of NVIDIA® BlueField®.

DOCA Firefly Service

[This page](#) provides instructions on how to use the DOCA Firefly service container on top of NVIDIA® BlueField®.

DOCA Flow Inspector Service

[This page](#) provides instructions on how to use the DOCA Flow Inspector service container on top of NVIDIA® BlueField®.

DOCA HBN Service

[This page](#) provides instructions on how to use the DOCA HBN Service container on top of NVIDIA® BlueField®.

DOCA Management Service

[This page](#) provides instructions on how to use the DOCA Management Service on top of NVIDIA® BlueField® Networking Platform or ConnectX® Network Adapters.

DOCA Telemetry Service

[This page](#) provides instructions on how to use the DOCA Telemetry Service (DTS) container on top of NVIDIA® BlueField®.

DOCA UROM Service

[This page](#) provides instructions on how to use the DOCA Telemetry Service (DTS) container on top of NVIDIA® BlueField®.

DOCA Switching

[These pages](#) describe the extensive switching capabilities enabled by DOCA libraries and services on these platforms.

API References

DOCA Driver APIs

[This page](#) contains DOCA driver APIs.

DOCA Libraries APIs

[This page](#) contains DOCA libraries APIs.

Miscellaneous

Glossary

[This page](#) provides a list of terms and acronyms and in the DOCA documentation.

Crypto Acceleration

[This page](#) shows the ability of NVIDIA® BlueField® to accelerate crypto operations.

DOCA Services Fluent Logger

[This page](#) provides instructions on how to use the logging infrastructure for DOCA services on top of NVIDIA® BlueField®.

DPU CLI

[This page](#) provides quick access to a useful set of CLI commands and utilities on the NVIDIA® BlueField® environment.

Emulated Devices

[This page](#) describes the ability of NVIDIA® BlueField® to emulate and accelerate physical and virtual host functions.

Modes of Operation

[This page](#) describes the modes of operation available for NVIDIA® BlueField®.

OpenSSL

[This page](#) provides instructions on using DOCA SHA for OpenSSL implementations.

Scalable Functions (SFs)

[This page](#) provides an overview and configuration of scalable functions (sub-functions, or SFs) for NVIDIA® BlueField®.

TLS Offload

[This page](#) provides an overview and configuration steps of TLS hardware offloading via kernel-TLS, using hardware capabilities of NVIDIA® BlueField®.

Troubleshooting

[This page](#) provides troubleshooting information for common issues and misconfigurations encountered when using DOCA for NVIDIA® BlueField®.

Virtual Functions (VFs)

[This page](#) provides an overview and configuration of virtual functions for NVIDIA® BlueField® and demonstrates a use case for running the DOCA applications over x86 host.


Archive

LTS Versions

[This page](#) provides pointers to the DOCA long term support (LTS) releases.

Documentation Archives

This page provides pointers to archived documentation of previous DOCA software releases.

 **Info**

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

DOCA SDK v2.7.0

This section contains the following pages:

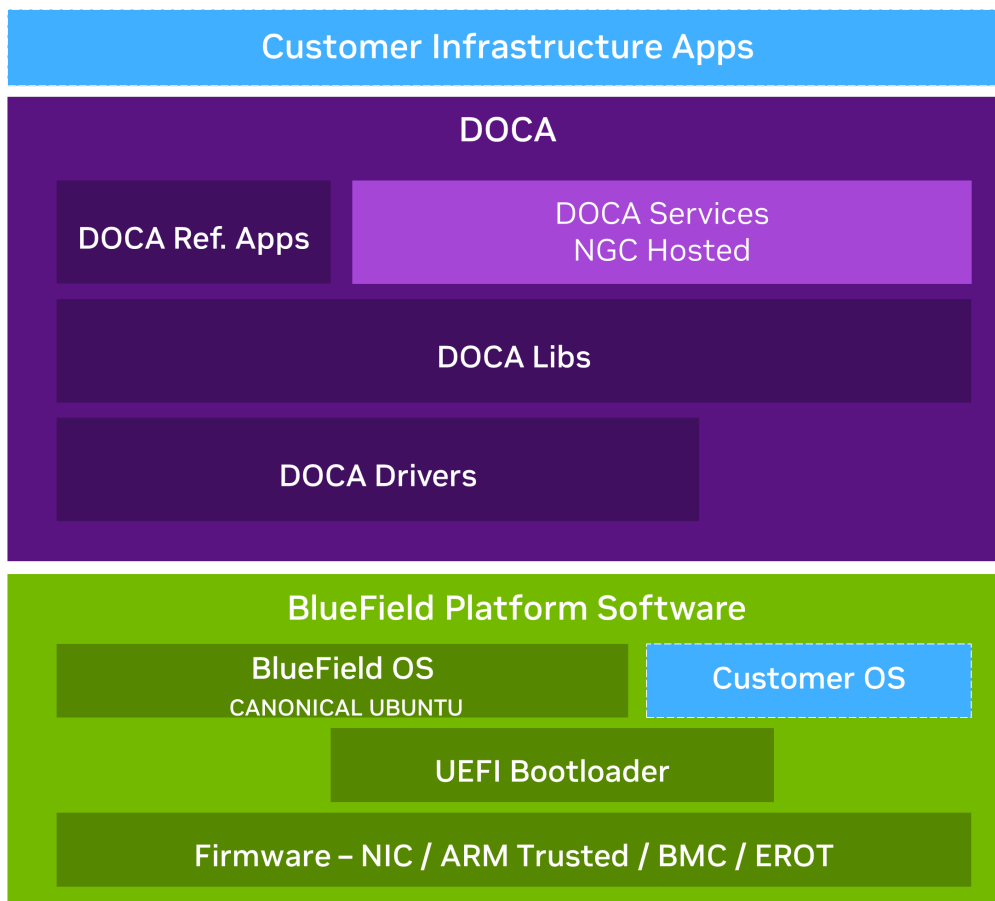
- [NVIDIA DOCA Overview](#)
- [NVIDIA DOCA Release Notes](#)
- [BlueField and DOCA User Types](#)
- [NVIDIA DOCA EULA](#)

NVIDIA DOCA Overview

This is an overview of the structure of NVIDIA DOCA documentation. It walks you through DOCA's developer zone portal which contains all the information about the DOCA toolkit from NVIDIA, providing all you need to develop NVIDIA® BlueField®-accelerated applications and the drivers for the host.

Introduction

The NVIDIA DOCA™ Framework enables rapidly creating and managing applications and services on top of the BlueField networking platform, leveraging industry-standard APIs. With DOCA, developers can deliver breakthrough networking, security, and storage performance by harnessing the power of NVIDIA's BlueField data-processing units (DPUs) and SuperNICs. Installing DOCA on your host provides all the necessary drivers and tools to manage NVIDIA® BlueField® and NVIDIA® ConnectX® devices.



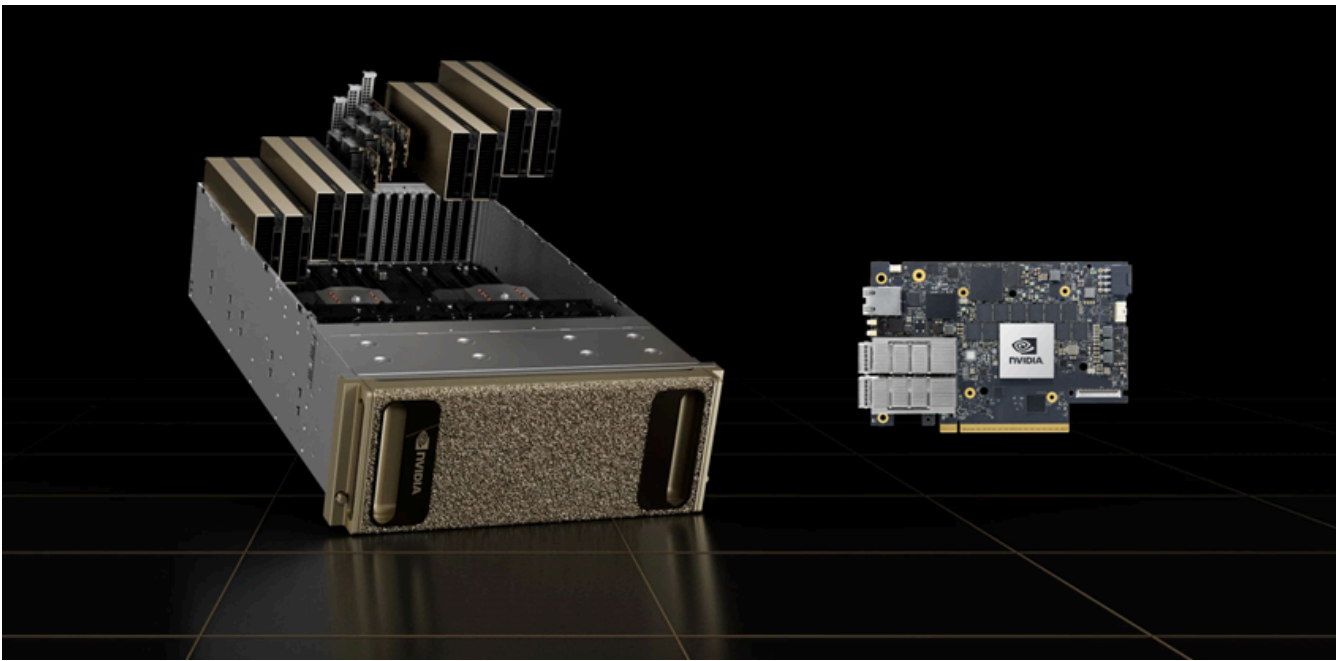
DOCA Framework includes the DOCA-Host package and the BlueField Software Bundle for BlueField Arm:

- BlueField Software Bundle (BF-Bundle) is the software package installed on the BlueField Arm cores
- DOCA-Host is the software package installed on the host server which includes different DOCA installation profiles

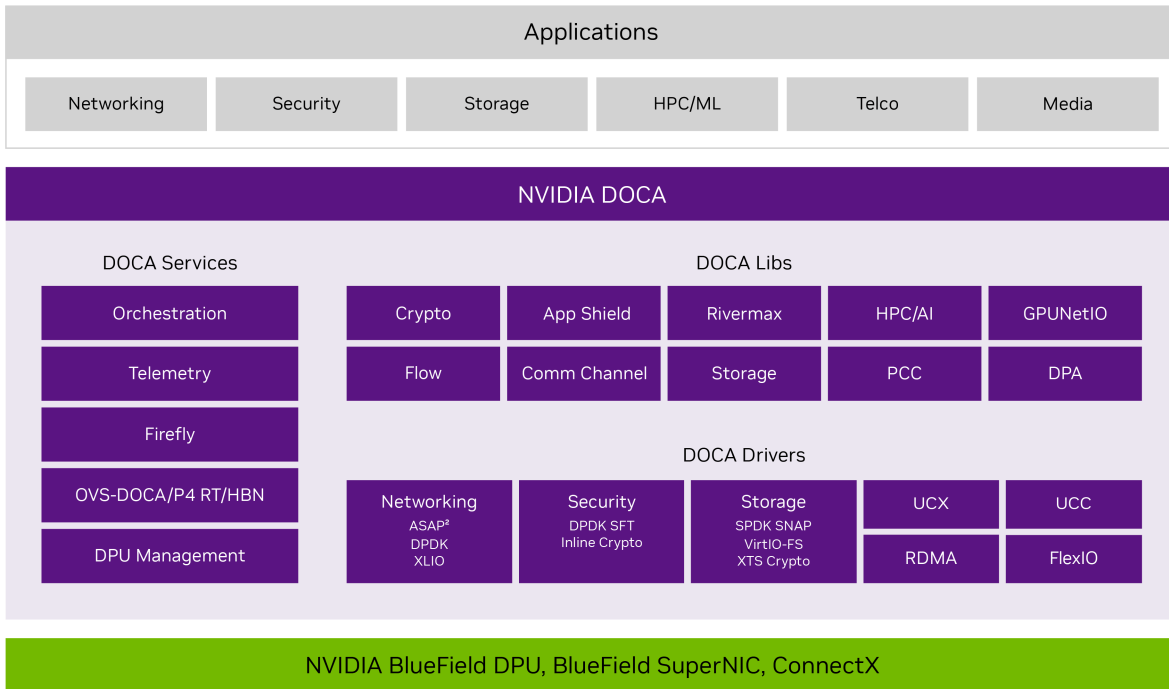
The BlueField Software Bundle includes:

- The DOCA runtime drivers and libs installed on top of the BlueField Platform
- The OS installed on the BlueField Platform
- The BlueField Platform Software (i.e., firmware and UEFI bootloader)

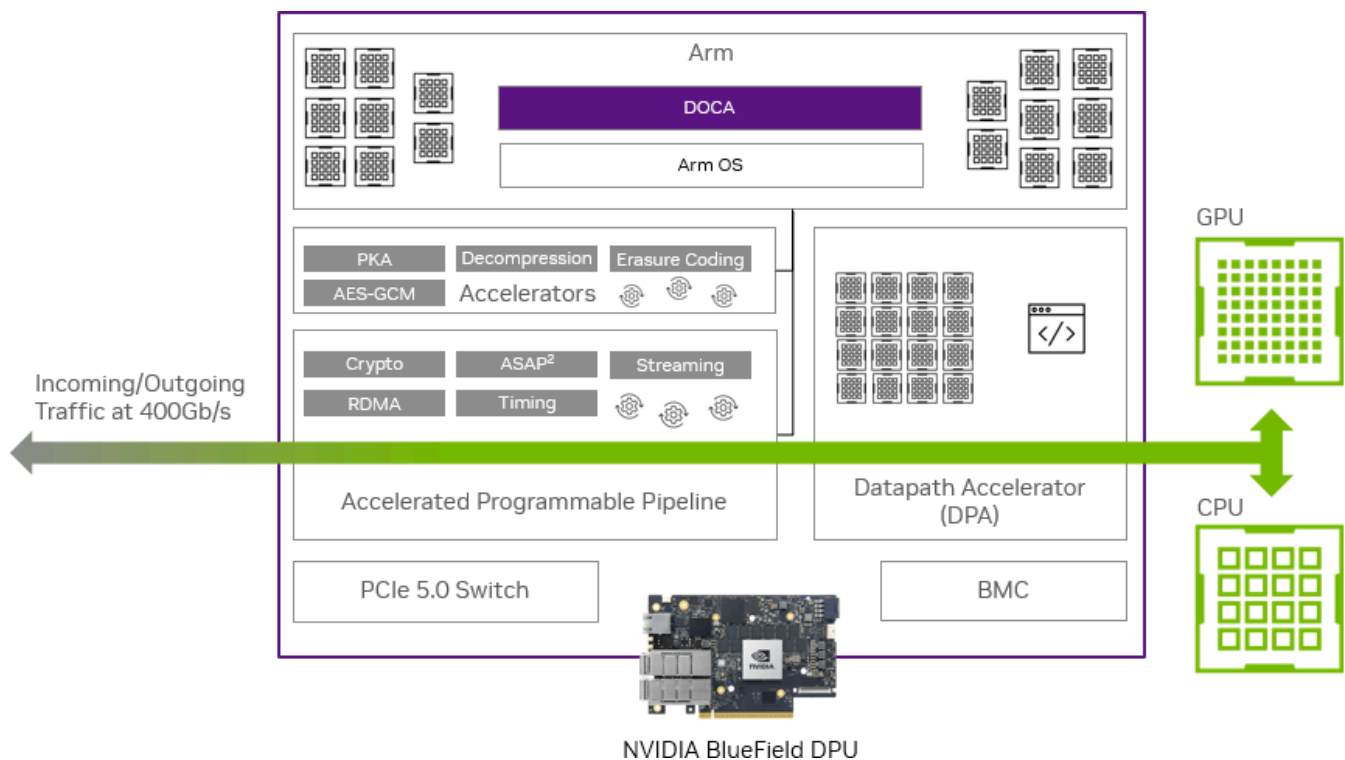
DOCA provides all the required libraries and drivers for hosts that include NVIDIA Networking platforms (i.e., BlueField and ConnectX) with a dedicated DOCA-Host package installation.



DOCA contains a runtime and development environment, including libraries and drivers for device management and programmability, for the host and as part of a BlueField Platform Software.



DOCA is the software infrastructure for BlueField's main hardware entities:



Installation

Installation instructions for both host and BlueField image can be found in the [NVIDIA DOCA Installation Guide for Linux](#).

Whether DOCA has been installed on the host or on the BlueField networking platform, one can find the different DOCA components under the `/opt/mellanox/doca` directory. These include the traditional SDK-related components (libraries, header files, etc.) as well as the DOCA samples, applications, tools and more, as described in this document.

API

The DOCA SDK is built around the different DOCA libraries designed to leverage the capabilities of BlueField. Under the [Programming Guide](#) section, one can find a detailed description of each DOCA library, its goals, and API. These guides document DOCA's API, aiming to help develop DOCA-based programs.

The [API References](#) section holds the Doxygen-generated documentation of DOCA's official API.

Programming Guides

DOCA programming guides provide the full picture of DOCA libraries and their APIs. Each guide includes an introduction, architecture, API overview, and other library-specific information.

Each library's programming guide includes code snippets for achieving basic DOCA-based tasks. It is recommended to review these samples while going over the programming guide of the relevant DOCA library to learn about its API. The samples provide an implementation example of a single feature of a given DOCA library.

For a more detailed reference of full DOCA-based programs that make use of multiple DOCA libraries, please refer to the [Reference Applications](#).

Applications

Applications are a higher-level reference code than the samples and demonstrate how a full DOCA-based program can be built. In addition to the supplied source code and compilation definitions, the applications are also shipped in their compiled binary form. This is to allow users an out-of-the-box interaction with DOCA-based programs without the hassle of a developer-oriented compilation process.

Many DOCA applications combine the functionality of more than one DOCA library and offer an example implementation for common scenarios of interest to users such as application recognition according to incoming/outgoing traffic, scanning files using the hardware RegEx acceleration, and much more.

For more information about DOCA applications, refer to [DOCA Applications](#).

Tools

Some of the DOCA libraries are shipped alongside helper tools for both runtime and development. These tools are often an extension to the library's own API and bridge the gap between the library's expected input format and the input available to the users.

For more information about DOCA tools, refer to [DOCA Tools](#).

Services

DOCA services are containerized DOCA-based programs that provide an end-to-end solution for a given use case. DOCA services are accessible as part of NVIDIA's container

catalog (NGC) from which they can be easily deployed directly to BlueField, and sometimes also to the host.

For more information about container-based deployment to the BlueField Platform, refer to the [NVIDIA BlueField Container Deployment Guide](#).

For more information about DOCA services, refer to the [DOCA Services](#).

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

NVIDIA DOCA Release Notes

NVIDIA DOCA SDK release notes containing information on new features, software interoperability, and known issues.

Introduction

DOCA 2.7.0 introduces NVIDIA® BlueField® networking platform enhancement for high-performance and secure AI bare-metal cloud and DOCA-Host updates for supported BlueField and NVIDIA® ConnectX® devices. With programmable congestion control (PCC) and data-path acceleration (DPA). DOCA SDK provides an extensive framework for developers.

New Features, Updates, and Enhancements

Note

BlueField-3 devices are not supported with MLNX_OFED as the host driver and are required to use DOCA-Host.

- Spectrum-X 1.0.1 with BlueField-3 SuperNIC
- DOCA PCC (GA) – Added new telemetry information to the PCC application
- DOCA Flow Enhancements, including DOCA Flow Tune Server and Pipeline Visualization for debugging (alpha support)
- DOCA Flow switch unified model supported

(i) Note

DOCA Flow switch has now unified all the representor ports to the switch manager port for traffic management. User should only manage the pipes on the switch manager port.

Applications must not call DPDK `rte_eth_dev_start/configure/stop` for VF/SF representors anymore. User should acquire packets (e.g., with DPDK but calling `rte_eth_rx/tx_burst()`) only with switch manager port.

- OVS-DOCA – Unified representor for multiple ports for better resource utilization with higher scale; OVS package rename for smoother installation
- Increased support for virtio-net VF devices on BlueField-3 networking platforms to 2K
- DOCA HBN Service 2.2.0 enhancements, including GA-level support for Local VRF Route Leaking, EVPN Downstream VNI (DVNI) for symmetric EVPN Route Leaking, Network-to-Network Hairpin routing support on BlueField uplinks
- SNAP Encryption at Rest with Zero-Copy: Available with BlueField-3 with SNAP 4.4.0
- DOCA Firefly Service enhancements, including new Telco profile, ptp4l update, new Firefly servo module
- Traffic Crypto - DOCA IPsec GA and merge into DOCA Flow, New Security protocol - PSP

- Alpha support for new DOCA Unified Resource and Offload Management (UROM) library
- Alpha support for DOCA Device Emulation (DevEmu) library – Emulate your own standard/non-standard devices on BlueField
- DOCA GPUNetIO new API to support RDMA
- DOCA Comm Channel (Comch) API update, Extend Comch to Arm DPA, host DPA.

(i) Note

Comch API is being updated, the old version is in deprecation process, DOCA 2.8 will be the last version to support it.

- DOCA Remote Direct Memory Operation (RDMO) reference application
- Alpha support for DOCA Management Service (DMS) – simplifying BlueField post-boot provisioning and configuration using standard configuration interfaces (API/CLI)
- DOCA NVQual – H20-BFx support, power stressors improvements
- DOCA NVCert – BlueField-3 SuperNIC support, SPC-X support using multi-DPU (and ConnectX) and GPU direct, RDMA/TCP-OVS/IPSec and VXLAN workloads
- Updated the default operation mode of SuperNICs to NIC mode (from DPU mode). This is relevant to the following SKUs:
 - 900-9D3B4-00CC-EA0
 - 900-9D3B4-00SC-EA0
 - 900-9D3B4-00CV-EA0
 - 900-9D3B4-00SV-EA0
 - 900-9D3B4-00EN-EA0

- 900-9D3B4-00PN-EA0
- 900-9D3D4-00EN-HA0
- 900-9D3D4-00NN-HA0

(i) Note

When upgrading one of these SuperNICs to 2.7.0, if its mode of operation was changed at any point in the past, then the last configured mode of operation will remain unchanged. Otherwise, the SuperNIC will rise in NIC operation mode.

- DOCA packaging – new BlueField firmware bundle package (bf-fwbundle-`<version>.prod.bfb`), a smaller image for Day 2 upgrades, without the OS and DOCA runtime. Includes ATF, UEFI, nic-fw, bmc-fw, and eROT only.
- BlueField-3 Firmware Components Upgrade – Upgrade all BlueField-3 firmware components in one upgrade flow through either `bfb-install` from the host (via RShim), or DPU BMC Redfish transfer BFB image
- Update BlueField NIC-Firmware automatically as part of `.bfb` image upgrade
- Improved BlueField BMC robustness –
 - Report LLDP for L2 discovery via Redfish
 - Improved BlueField DPU debuggability
- Compilation on top of DOCA's SDK
 - DOCA 2.7 – installation now includes additional `pkg-config (.pc)` definitions per DOCA SDK library on top of the general `doca.pc` file. This is part of a deprecation process for `doca.pc` and a focus on modularity of DOCA's SDK.
 - Please refer to DOCA SDK reference samples and applications for an example of using the per library `.pc` files.

Note

Starting with DOCA 2.8, the general `doca.pc` file will be removed from the release and only files per DOCA SDK library will remain.

- Added support for new [BlueField reset and reboot procedures](#) for loading new firmware and firmware configuration changes which replace previous need for server power cycle

Installation Notes

Note

The format of image filenames for the BF-Bundle and DOCA-Host have been updated to the following template:

- BF-Bundle image file format – `bf-bundle-<doca_ver.LTS#>-<build#>[BUILD-LABEL]-<yy.mm>-<OS_distro>-<#os_ver>[OS-LABEL]-<unsigned/dev/prod>.<bfb/iso>`
- DOCA-Host image file format – `doca-host-<doca_ver.LTS#>-<build#>[BUILD-LABEL]-<yy.mm>-<OS_distro>-<#os_ver>[OS-LABEL]-<arch>.<rpm/deb/iso>`

Where:

- `<doca_ver.LTS#>-<build#>` – the DOCA version with the NVIDIA build number in a x.y.z-abcd format (e.g., 2.7.0-1456). If it is an LTS release, it indicates which update number it is.
- `<yy.mm>` – the year and month the image was created/released (e.g., 24.04)
- `<OS_distro>-<#os_ver>` – the name and version of the operating system (e.g., ubuntu-22.04)

- `<arch>.<rpm/deb/iso>` – which processor architecture is supported and how the image is packaged (e.g., `x86.rpm` or `x86.deb`)
- `<unsigned/dev/prod>.<bf/iso>` – security signature of the image (no signature, development, production) and how the image is packaged (e.g., `prod.rpm` or `dev.rpm`)

Refer to the [NVIDIA DOCA Installation Guide for Linux](#) for information on:

- Setting up DOCA SDK on your BlueField networking platform or SmartNIC
- Supported BlueField platforms

Note

By default, installing DOCA profiles with standard Linux tools (`yum`, `apt`) installs both `doca-runtime` and `doca-devel` (previously `doca-sdk`).

- `doca-runtime` includes all the components, libs, drivers, and tools used in the production environment by the DOCA admin
- `doca-devel` includes all the components, libs, drivers, and tools used for development, including reference applications, compilers, etc.

Starting with DOCA 2.8, the default installation of BlueField-Bundle and DOCA-Host profiles will only include DOCA runtime. `doca-devel` can be installed manually as needed.

Embedded DOCA Libraries

Component	Version
<code>doca-apps</code>	2.7.0
<code>doca-grpc</code>	2.7.0

Component	Version
doca-libs	2.7.0
ucx	1.17.0-1.2404066
gpunetio	2.7.0

Embedded DOCA Firmware Components

Component	Version	Description
ATF	v2.2(release): 4.7.0-25- g5569834	Arm-trusted firmware is a reference implementation of secure world software for Arm architectures
UEFI	4.7.0-42- g13081ae	UEFI is a specification that defines the architecture of the platform firmware used for booting and its interface for interaction with the operating system
BlueField-3 NIC firmware	32.41.1000	Firmware is used to run user programs on the BlueField-3 which allow hardware to run
BlueField-2 NIC firmware	24.41.1000	Firmware is used to run user programs on the BlueField-2 which allow hardware to run
BMC firmware	24.04	BlueField BMC firmware
BlueField-3 eROT (Glacier)	cec_ota_BMG P-04.0f	BlueField-3 eROT firmware
BlueField-2 eROT (CEC)	00.02.0182.0 000.n02	BlueField-2 eROT firmware

Embedded DOCA Drivers

Component	Version	Description
collectx-clxapi	1.17.0-1	A library which exposes the CollectX API, which allows any 3 rd party to easily use CollectX functionality in their own programs
doca-base (MLNX_OFED)	24.04 - 0.6.6.0	NVIDIA® MLNX_OFED is a single software stack that operates across all NVIDIA network adapter solutions
dpacc	1.7.0-1	DPACC is a high-level compiler for the DPA processor which compiles code targeted for the data-path accelerator (DPA) processor into a device executable and generates a DPA program
dpcp	1.1.48-1.2404066	DPCP provides a unified flexible interface for programming IB devices using DevX
flexio	24.04.2148-0	FlexIO SDK exposes an API for managing the device and executing native code over the DPA processor
ibutils	2.1.1	ibdiagnet scans the fabric using directed route packets and extracts all the available information regarding its connectivity and devices.
libvma	9.8.60-1	The NVIDIA® VMA library accelerates latency-sensitive and throughput-demanding TCP and UDP socket-based applications by offloading traffic from the user-space directly to the NIC, without going through the kernel and the standard IP stack (kernel-bypass)
libxlio	3.30.5-1.2404066	The NVIDIA® XLIO software library boosts the performance of TCP/IP applications based on NGINX (CDN, DoH, etc.) and storage solutions as part of the SPDK

Component	Version	Description
MFT	4.28.0-92	NVIDIA® MFT is a set of firmware management and debug tools for NVIDIA devices
mlnx-dpdk	22.11.0-2404	Equivalent to DPDK upstream. The versioning of MLNX_DPDK indicates which upstream DPDK it is compatible with it (e.g., 22.11 is compatible with upstream DPDK 2022.11).
mlnx-libsnap	1.6.0-1	Libsnap is a common library designed to assist common tasks for applications wishing to interact with emulated hardware over BlueField and take the most advantage from hardware capabilities
mlnx-snap	3.8.0-3	BlueField SNAP for NVMe and virtio-blk enables hardware-accelerated virtualization of local storage
mlx-regex	1.2-ubuntu1	RegEx is a library that provides RegEx pattern matching to DOCA applications using the regular expression processor (RXP) or software-based engines when required
OpenSM	5.19.0	InfiniBand Subnet Manager and Subnet Administrator based on OpenSM
Rivermax	1.50.7	NVIDIA® Rivermax® is an optimized networking SDK for media and data streaming applications
RShim	2.0.27	The user-space driver to access the BlueField SoC via the RShim interface, providing ways to push boot stream, debug the target, or login via the virtual console or network interface
SHARP	3.7.0	Improves the performance of MPI and Machine Learning collective operation by offloading from CPUs and GPUs to the network and eliminating the need to send data multiple times between endpoints
SPDK	23.01.5-20	SPDK provides a set of tools and libraries for writing high performance, scalable, user-mode storage applications
Virtio-net-controller	1.9.17-1	Virtio-net-controller is a systemd service running on BlueField, with a user interface front-end to communicate with the background service
VMA	9.8.60-1	Accelerates latency-sensitive and throughput-demanding TCP and UDP socket-based applications by offloading traffic from the user-space

Component	Version	Description
		directly to the network interface card (NIC) or Host Channel Adapter (HCA)
XLIO	3.30.5	Boosts the performance of TCP/IP applications based on NGINX (CDN, DoH, etc.) and storage solutions as part of the SPDK

DOCA Packages

Device	Component	Version	Description
Host	DOCA Devel	2.7.0	Software development kit package and tools for developing host software
	DOCA Runtime	2.7.0	Runtime libraries and tools required to run DOCA-based software applications on host
	DOCA Extra	2.7.0	Contains helper scripts (doca-info, doca-kernel-support)
	DOCA OFED	2.7.0	Software stack which operates across all NVIDIA network adapter solutions
	Arm emulated (QEMU) development container	4.7.0	Linux-based BlueField Arm emulated container for developers
Target BlueField DPU (Arm)	BlueField BSP	4.7.0	BlueField image and firmware
	DOCA SDK	2.7.0	Software development kit packages and tools for developing Arm software
	DOCA Runtime	2.7.0	Runtime libraries and tools required to run DOCA-based software applications on Arm

Supported Host OS per DOCA-Host Installation Profile

Note

Starting with DOCA version 2.6.0 OSs with kernel versions lower than 4.18 will no longer be supported. DOCA 2.5.0 is the last version to support OS with kernel lower than 4.18.

The default operating system included with the BlueField Bundle (for DPU and SuperNIC) is Ubuntu 22.04.

The supported operating systems on the host machine per DOCA-Host installation profile are the following:

Note

Only the following generic kernel versions are supported for DOCA local repo package for host installation.

OS	OS Version	Default Kernel Version	Arch	doca-all	doca-networking	doca- ofed
Alinux	3.2	5.10.134-13.al8.x86_64	x86			
Anolis	8.6	5.10.134+	aarch			

			64			
			x86			
BCLinux	21.10SP2	4.19.90-2107.6.0.0098.oe1.bclinux.aarch64	aarch64			
		4.19.90-2107.6.0.0100.oe1.bclinux.x86_64	x86			
CTYunOS	2.0	4.19.90-2102.2.0.0062.ctl2.aarch64	aarch64			
		4.19.90-2102.2.0.0062.ctl2.x86_64	x86			
	3.0 (23.01)	5.10.0-136.12.0.86.ctl3.aarch64	aarch64			
		5.10.0-136.12.0.86.ctl3.x86_64	x86			
Debian	10.13	4.19.0-21-arm64	aarch64			
		4.19.0-21-amd64	x86			
	10.8	4.19.0-14-arm64	aarch64			
		4.19.0-14-amd64	x86			
	10.9	4.19.0-16-amd64	x86			
	11.3	5.10.0-13-arm64	aarch64			
		5.10.0-13-amd64	x86			
	12.1	6.1.0-10-arm64	aarch64			
6.1.0-10-amd64		x86				

EulerOS	2.0sp11	5.10.0-60.18.0.50.h323.eulerosv2r11.aarch64	aarch64			
		5.10.0-60.18.0.50.h323.eulerosv2r11.x86_64	x86			
	2.0sp12	5.10.0-136.12.0.86.h1032.eulerosv2r12.aarch64	aarch64			
		5.10.0-136.12.0.86.h1032.eulerosv2r12.x86_64	x86			
Kylin	10sp2	4.19.90-24.4.v2101.ky10.aarch64	aarch64			
		4.19.90-24.4.v2101.ky10.x86_64	x86			
	10sp3	4.19.90-52.22.v2207.ky10.aarch64	aarch64			
		4.19.90-52.22.v2207.ky10.x86_64	x86			
Mariner	2.0	5.15.118.1-1.cm2.x86_64	x86			
Oracle Linux	7.9	5.4.17-2011.6.2.el7uek.x86_64	x86			
	8.4	5.4.17-2102.201.3.el8uek.x86_64	x86			
	8.6	5.4.17-2136.307.3.1.el8uek.x86_64	x86			

	8.7	5.15.0-3.60.5.1.el8uek.x86_64	x86			
	8.8	5.15.0-101.103.2.1.el8uek.x86_64	x86			
	9.1	5.15.0-3.60.5.1.el9uek.x86_64	x86			
	9.2	5.15.0-101.103.2.1.el9uek.x86_64	x86			
openEuler	20.03sp3	4.19.90-2112.8.0.0131.oe1.aarch64	aarch64			
		4.19.90-2112.8.0.0131.oe1.x86_64	x86			
	22.03	5.10.0-60.18.0.50.oe2203.aarch64	aarch64			
		5.10.0-60.18.0.50.oe2203.x86_64	x86			
RHEL/CentOS	8.0	4.18.0-80.el8.aarch64	aarch64			
		4.18.0-80.el8.x86_64	x86			
	8.1	4.18.0-147.el8.aarch64	aarch64			
		4.18.0-147.el8.x86_64	x86			
	8.2	4.18.0-193.el8.aarch64	aarch64			
		4.18.0-193.el8.x86_64	x86			
	8.3	4.18.0-240.el8.aarch64	aarch64			

		4.18.0-240.el8.x86_64	x86			
	8.4	4.18.0-305.el8.aarch64	aarch64			
		4.18.0-305.el8.x86_64	x86			
RHEL/Rocky	8.5	4.18.0-348.el8.aarch64	aarch64			
		4.18.0-348.el8.x86_64	x86			
	8.6	4.18.0-372.41.1.el8_6.aarch64	aarch64			
		4.18.0-372.41.1.el8_6.x86_64	x86			
	8.7	4.18.0-425.14.1.el8_7.aarch64	aarch64			
		4.18.0-425.14.1.el8_7.x86_64	x86			
	8.8	4.18.0-477.10.1.el8_8.aarch64	aarch64			
		4.18.0-477.10.1.el8_8.x86_64	x86			
	8.9	4.18.0-513.5.1.el8_9.aarch64	aarch64			
		4.18.0-513.5.1.el8_9.x86_64	x86			
	9.0	5.14.0-70.46.1.el9_0.aarch64	aarch64			
		5.14.0-70.46.1.el9_0.x86_64	x86			
	9.1	5.14.0-162.19.1.el9_1.aarch64	aarch64			
		5.14.0-162.19.1.el9_1.x86_64	x86			

	9.2	5.14.0-284.11.1.el9_2.aarch64	aarch64			
		5.14.0-284.11.1.el9_2.x86_64	x86			
	9.3	5.14.0-362.8.1.el9_3.aarch64	aarch64			
		5.14.0-362.8.1.el9_3.x86_64	x86			
RHEL/Rocky	9.4	5.14.0-427.13.1.el9_4.aarch64	aarch64			
		5.14.0-427.13.1.el9_4.x86_64	x86			
SLES	15sp2	5.3.18-22-default	aarch64			
			x86			
	15sp3	5.3.18-57-default	aarch64			
			x86			
	15sp4	5.14.21-150400.22-default	aarch64			
			x86			
	15sp5	5.14.21-150500.53-default	aarch64			
			x86			
TKLinux	3.3	5.4.119-19.0009.39	aarch64			
		5.4.119-19.0009.39	x86			
Ubuntu	20.04	5.4.0-26-generic	aarch64			
			x86			
	22.04	5.15.0-25-generic	aarch64			

			x86			
	24.04	6.8.0-31-generic	aarch64			
			x86			
UOS	20.1060a	5.10.0-46.uelc20.aarch64	aarch64			
		5.10.0-46.uelc20.x86_64	x86			
	20.1060e	5.10.0-46.uel20.aarch64	aarch64			
		5.10.0-46.uel20.x86_64	x86			

BFB Version Upgrade/Downgrade

The following table provides a matrix for the supported upgrade/downgrade of BFBs across different versions.

Version	Upgrade to	Downgrade to
1.0.0	1.1.0; 1.1.1	N/A
1.1.0	1.1.1; 1.2.0	1.0.0
1.1.1	1.2.0; 1.3.0	1.1.0; 1.0.0
1.2.0	1.3.0; 1.4.0	1.1.1; 1.1.0
1.3.0	1.4.0; 1.5.0	1.2.0; 1.1.1
1.4.0	1.5.0; 2.0.2	1.3.0; 1.2.0
1.5.0	2.0.2; 2.2.0; 1.5.1; 1.5.2; 1.5.3	1.4.0; 1.3.0

Version	Upgrade to	Downgrade to
1.5.1	1.5.2	1.5.0
1.5.2	1.5.3	1.5.1; 1.5.0
1.5.3	N/A	1.5.2; 1.5.0
2.0.2	2.2.0; 2.5.0	1.5.0; 1.4.0
2.2.0	2.5.0; 2.6.0	N/A
2.2.1	2.5.0; 2.6.0	N/A
2.5.0	2.5.1; 2.6.0	2.2.1 for BlueField-3; 2.2.0 for BlueField-2
2.5.1	N/A	2.5.0
2.6.0	2.7.0	2.5.0; 2.2.1 for BlueField-3; 2.2.0 for BlueField-2
2.7.0	N/A	2.6.0; 2.5.0; 2.2.1 for BlueField-3; 2.2.0 for BlueField-2

Supported DOCA Version Upgrade Using Standard Linux Tools on BlueField

Version	Upgrade to
2.5.0	2.5.1; 2.6.0
2.5.1	N/A
2.6.0	2.7.0

Technical Support

Customers who purchased NVIDIA products directly from NVIDIA are invited to contact us through the following methods:

- E-mail: enterprisesupport@nvidia.com

- Enterprise Support page: <https://www.nvidia.com/en-us/support/enterprise>

Customers who purchased NVIDIA M-1 Global Support Services, please see your contract for details regarding Technical Support.

Customers who purchased NVIDIA products through an NVIDIA-approved reseller should first seek assistance through their reseller.

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

Known Issues

The following table lists the known issues and limitations for this release of DOCA SDK.

Reference	Description
3882794	<p>Description: When working with <code>doca_pcc_np</code> context, the return value from the API <code>doca_pcc_get_max_num_threads()</code> is incorrect. The function has an output parameter that indicates the maximum number of threads allowed for a <code>doca_pcc_np</code> context. The correct value that the library expects is 16 instead of the returned 64.</p> <p>Workaround: N/A</p> <p>Keyword: PCC; threads</p> <p>Reported in version: 2.7.0</p>
388674	<p>Description: Installing <code>doca-all</code> and other DOCA metapackages does not install the <code>mlnx-nvme</code> driver.</p> <p>Workaround: <code>mlnx-nvme</code> is only needed for NVMe-over-RDMA remote storage support. If you wish to install it, add the <code>mlnx-nvme</code> package to the install command.</p> <ul style="list-style-type: none"> • On RHEL:

Reference	Description
	<pre>apt install doca-all mlnx-nvme-modules</pre> <ul style="list-style-type: none"> On Ubuntu: <pre>dnf install doca-all-kmod-mlnx-nvme</pre> <p>Keyword: NVMe; DOCA profile</p> <p>Reported in version: 2.7.0</p>
38 85 93 0	<p>Description: When installing DOCA-Host on a system using NVMe storage (typically local NVMe disk), and the script <code>doca-kernel-support</code> is used to rebuild and install kernel modules, unloading the <code>mlx5</code> drivers is only possible after also unmounting the NVMe storage, which would typically necessitate a reboot.</p> <p>Workaround: N/A</p> <p>Keyword: NVMe; <code>doca-kernel-support</code>; DOCA for host</p> <p>Reported in version: 2.7.0</p>
38 86 31 5	<p>Description: To reset or shut down the BlueField Arm, it is mandatory to specify the <code>--sync 0</code> argument with reset level 1 and reset type 3 or 4. For example:</p> <pre>mlxfwreset -d <device> -l 1 -t 4 --sync 0 r</pre> <p>Workaround: N/A</p> <p>Keyword: Arm; shutdown</p> <p>Reported in version: 2.7.0</p>
38 37 25 5	<p>Description: When running Arm shutdown from the host OS it is expected to get the message <code>-E- Failed to send Register MRSI</code>. This message should be ignored.</p> <p>Workaround: Wait 2 more minutes before rebooting the host. Before proceeding with host OS reboot, it is recommended to query the operational state of the BlueField Arm cores from the BlueField BMC to verify that shutdown state has been reached. Run the following command:</p> <pre>ipmitool -C 17 -I lanplus -H <bmc_ip> -U root -P <password> raw 0x32 0xA3</pre> <p>Expected output is "06".</p>

Reference	Description
	Keyword: Host OS; reboot; error
	Reported in version: 2.7.0
38 81 94 1	<p>Description: When working with RShim 2.0.28, PCIe host crash may rarely occur at the beginning of BFB push after the Arm reset.</p> <p>Workaround: Downgrade to RShim 2.0.27 or upgrade to RShim 2.0.29.</p> <p>Keyword: RShim; driver</p> <p>Reported in version: 2.7.0</p>
38 44 70 5	<p>Description: In OpenEuler 20.03, the Linux Kernel version 4.19.90 is affected by an issue that impacts the discard/trim functionality for the BlueField eMMC device which may cause degraded performance of the BlueField eMMC over time.</p> <p>Workaround: Upgrade to Linux Kernel version 5.10 or later.</p> <p>Keyword: eMMC discard; trim functionality</p> <p>Reported in version: 2.7.0</p>
38 77 72 5	<p>Description: During BFB installation in NIC mode on BlueField-3, too much information is added into RShim log which fills it, causing the Linux installation progress log to not appear in the RShim log.</p> <pre>echo "DISPLAY_LEVEL 2" > /dev/rshim0/misc cat /dev/rshim0/misc</pre> <p>Workaround: Monitor the BlueField-3 Arm's UART console to check whether BFB installation has completed or not for NIC mode.</p> <pre>[13:58:39] INFO: Installation finished ... [14:01:53] INFO: Rebooting...</pre> <p>Keyword: NIC mode; BFB install</p> <p>Reported in version: 2.7.0</p>
38 55	<p>Description: Trying to jump from a steering level in the hardware to a lower level using software steering is not supported on rdma-core lower than 48.x.</p>

Reference	Description
702	<p>Workaround: N/A</p> <p>Keyword: RDMA; SWS</p> <p>Reported in version: 2.7.0</p>
3855485	<p>Description: When enabling the PCI_SWITCH_EMULATION_ENABLE NVconfig, the mlx devices, and potentially the RShim devices disappear. Also, looking at the kernel logs using dmesg shows the following messages:</p> <pre data-bbox="232 642 1466 835">pci 0000:29:00.0: BAR 0: no space for [mem size 0x0200 0000 64bit pref] pci 0000:29:00.0: BAR 2: no space for [mem size 0x0080 0000 64bit pref] ...</pre> <p>Workaround: N/A</p> <p>Keyword: NVconfig; RShim; dmsg</p> <p>Reported in version: 2.7.0</p>
3831230	<p>Description: In OpenEuler 20.03, the Linux Kernel version 4.19.90 is affected by an issue that impacts the discard/trim functionality for BlueField eMMC device which may cause degraded performance of BlueField eMMC over time.</p> <p>Workaround: Upgrade to Linux Kernel version 5.10 or later.</p> <p>Keyword: eMMC discard; trim functionality</p> <p>Reported in version: 2.7.0</p>
3743879	<p>Description: mlxfwreset could timeout on servers where the RShim driver is running and INTx is not supported. The following error message is printed: BF reset flow encountered a failure due to a reset state error of negotiation timeout.</p> <p>Workaround: Set PCIE_HAS_VFIO=0 and PCIE_HAS_UIO=0 in /etc/rshim.conf and restart the RShim driver. Then re-run the mlxfwreset command. If host Linux kernel lockdown is enabled, then manually unbind the RShim driver before mlxfwreset and bind it back after mlxfwreset:</p> <pre data-bbox="232 1730 1466 1919">echo "DROP_MODE 1" > /dev/rshim0/misc mlxfwreset <arguments> echo "DROP_MODE 0" > /dev/rshim0/misc</pre>

Reference	Description
	Keyword: Timeout; mlxfwreset; INTx
	Reported in version: 2.7.0
38	Description: Users cannot use --job-output-buffer-size 0 when using remote output memory (--use-remote-output-buffers).
69	Workaround:
63	Keyword: DOCA Bench
9	Reported in version: 2.7.0
38	Description: An issue occurs when submitting tasks with DOCA SHA with the following error.
72	<pre>[DOCA][ERR][doca_pe.cpp:177][task_submit] Task 0xaaaaf4865bf0: Failed to submit task: task is already submitted</pre>
65	Workaround: Reattempt the submit using a different --data-provider-job-count value. This workaround may also fail.
4	Keyword: DOCA Bench
	Reported in version: 2.7.0
38	Description: Multi-threaded tests using DOCA Comch may hang or emit an infinite amount of log messages. Single-threaded tests are less likely to cause this issue.
59	Workaround: N/A
82	Keyword: DOCA Bench; DOCA Comch
3	Reported in version: 2.7.0
38	Description: Send tasks on DOCA RDMA may fail.
57	Workaround: N/A
09	Keyword: DOCA Bench; DOCA RDMA; send
5	Reported in version: 2.7.0
38	Description: DOCA RDMA tests cannot be launched from BlueField side.
57	Workaround: N/A

Reference	Description
097	Keyword: DOCA Bench; DOCA RDMA Reported in version: 2.7.0
3849701	Description: DOCA Comch tests can not be launched from BlueField side . Workaround: N/A Keyword: DOCA Bench; DOCA Comch Reported in version: 2.7.0
3840230	Description: Order of cores specified in --core-list is not respected. Cores are picked in ascending order instead. Workaround: N/A Keyword: DOCA Bench Reported in version: 2.7.0
3665070	Description: Virtio-net controller fails to load if DPA_AUTHENTICATION is enabled. Workaround: N/A Keyword: Virtio-net; DPA Reported in version: 2.5.0
3678069	Description: If using BlueField with NVMe and mmcbld and configured to boot from mmcbld, users must create bf.cfg file with device=/dev/mmcbld0, then install the *.bfb as normal. Workaround: N/A Keyword: NVMe Reported in version: 2.5.0
3680538	Description: When using strongSwan or OVS-IPsec as explained in the NVIDIA BlueField DPU BSP , the IPSec Rx data path is not offloaded to hardware and occurs in software running on the Arm cores. As a result, bandwidth performance is substantially low. Workaround: N/A Keyword: IPsec

Reference	Description
	Reported in version: 2.5.0
N/A	Description: Execution unit partitions are still not implemented and would be added in a future release.
	Workaround: N/A
	Keyword: EU tool
	Reported in version: 2.5.0
3666160	Description: Installing BFB using bfb-install when mlxconfig PF_TOTAL_SF>1700, triggers server reboot immediately.
	Workaround: Change PF_TOTAL_SF to 0, perform a <u>graceful shutdown</u> , power cycle, then installing BFB.
	Keyword: SF; PF_TOTAL_SF; BFB installation
	Reported in version: 2.2.1
3594836	Description: When enabling Flex IO SDK tracer at high rates, a slow-down in processing may occur and/or some traces may be lost.
	Workaround: Keep tracing limited to ~1M traces per second to avoid a significant processing slow-down. Use tracer for debug purposes and consider disabling it by default.
	Keyword: Tracer FlexIO
	Reported in version: 2.2.1
3592080	Description: When using UEK8 on the host in DPU mode, creating a VF on the host consumes about 100MB memory on BlueField
	Workaround: N/A
	Keyword: UEK; VF
	Reported in version: 2.2.1
3566042	Description: Virtio hotplug is not supported in GPU-HOST mode on the NVIDIA Converged Accelerator.
	Workaround: N/A

Reference	Description
	Keyword: Virtio; Converged Accelerator
	Reported in version: 2.2.0
35	Description: PXE boot over ConnectX interface might not work due to an invalid MAC address in the UEFI boot entry.
46	Workaround: On BlueField, create <code>/etc/bf.cfg</code> file with the relevant PXE boot entries, then run the command <code>bfcfg</code> .
47	
4	Keyword: PXE; boot; MAC
	Reported in version: 2.2.0
35	Description: Running <code>mlxfwreset sync 1</code> on NVIDIA Converged Accelerators may be reported as supported although it is not. Executing the reset will fail.
61	Workaround: N/A
72	Keywords: <code>mlxfwreset</code>
3	Reported in version: 2.2.0
35	Description: After rebooting a BlueField-3 DPU running Rocky Linux 8.6 BFB, the kernel log shows the following error:
46	<pre>[3.787135] mlxbf_gige MLNXBF17:00: Error getting PHY irq. Use polling instead</pre>
20	This message indicates that the Ethernet driver will function normally in all aspects, except that PHY polling is enabled.
2	Workaround: N/A
	Keywords: Linux; PHY; kernel
	Reported in version: 2.2.0
33	Description: When performing longevity tests (e.g., <code>mlxfwreset</code> , DPU reboot, burning of new BFBs), a host running an Intel CPU may observe errors related to "CPU 0: Machine Check Exception".
06	
48	Workaround: Add <code>intel_idle.max_cstate=1</code> entry to the kernel command line.
9	Keywords: Longevity; <code>mlxfwreset</code> ; DPU reboot

Reference	Description
	Reported in version: 2.2.0
35 29 29 7	<p>Description: Enhanced NIC mode is not supported on BlueField-2.</p> <p>Workaround: N/A</p> <p>Keywords: Operation; mode</p> <p>Reported in version: 2.2.0</p>
35 38 48 6	<p>Description: When removing LAG configuration from BlueField, a kernel warning for <code>uverbs_destroy_ufile_hw</code> is observed if <code>virtio-net-controller</code> is still running.</p> <p>Workaround: Stop <code>virtio-net-controller</code> service before cleaning up bond configuration.</p> <p>Keywords: Virtio-net; LAG</p> <p>Reported in version: 2.2.0</p>
35 34 21 9	<p>Description: On BlueField-3 devices, from DOCA 2.2.0 to 32.37.1306 (or lower), the host crashes when executing partial Arm reset (e.g., Arm reboot; BFB push; <code>mlxfwreset</code>).</p> <p>Workaround: Before downgrading the firmware:</p> <ol style="list-style-type: none"> Run: <pre data-bbox="313 1262 1463 1356">echo 0 > /sys/bus/platform/drivers/mlxbf-bootctl/large_icm</pre> Reboot Arm. <p>Keyword: BlueField-3; downgrade</p> <p>Reported in version: 2.2.0</p>
34 62 63 0	<p>When trying to perform a PXE installation when UEFI Secure Boot is enabled, the following error messages may be observed:</p> <pre data-bbox="233 1675 1463 1822">error: shim_lock protocol not found. error: you need to load the kernel first.</pre> <p>Workaround: Download a Grub EFI binary from the Ubuntu website. For further information on Ubuntu UEFI Secure Boot PXE Boot, please visit Ubuntu's official</p>

Reference	Description
	<p>website.</p> <p>Keyword: PXE; UEFI Secure Boot</p> <p>Reported in version: 2.0.2</p>
34 48 84 1	<p>Description: While running CentOS 8.2, switchdev Ethernet BlueField runs in "shared" RDMA net namespace mode instead of "exclusive".</p> <p>Workaround: Use <code>ib_core</code> module parameter <code>netns_mode=0</code>. For example:</p> <pre>echo "options ib_core netns_mode=0" >> /etc/modprobe.d/mlnx-bf.conf</pre> <p>Keyword: RDMA; isolation; Net NS</p> <p>Reported in version: 2.0.2</p>
27 06 80 3	<p>Description: When an NVMe controller, SoC management controller, and DMA controller are configured, the maximum number of VFs is limited to 124.</p> <p>Workaround: N/A</p> <p>Keyword: VF; limitation</p> <p>Reported in version: 2.0.2</p>
32 73 43 5	<p>Description: Changing the mode of operation between NIC and DPU modes results in different capabilities for the host driver which might cause unexpected behavior.</p> <p>Workaround: Reload the host driver or reboot the host.</p> <p>Keyword: Modes of operation; driver</p> <p>Reported in version: 2.0.2</p>
32 64 74 9	<p>Description: In Rocky and CentOS 8.2 inbox-kernel BFBs, RegEx requires the following extra huge page configuration for it to function properly:</p> <pre>sudo hugeadm --pool-pages-min DEFAULT:2048M sudo systemctl start mlx-regex.service systemctl status mlx-regex.service</pre> <p>If these commands have executed successfully you should see <code>active (running)</code> in the last line of the output.</p>

Reference	Description
	Workaround: N/A
	Keyword: RegEx; hugepages
	Reported in version: 1.5.1
3240153	Description: DOCA kernel support only works on a non-default kernel.
	Workaround: N/A
	Keyword: Kernel
	Reported in version: 1.5.0
3217627	Description: The <code>doca_devinfo_rep_list_create</code> API returns success on the host instead of Operation not supported.
	Workaround: N/A
	Keyword: DOCA core; InfiniBand
	Reported in version: 1.5.0

BlueField and DOCA User Types

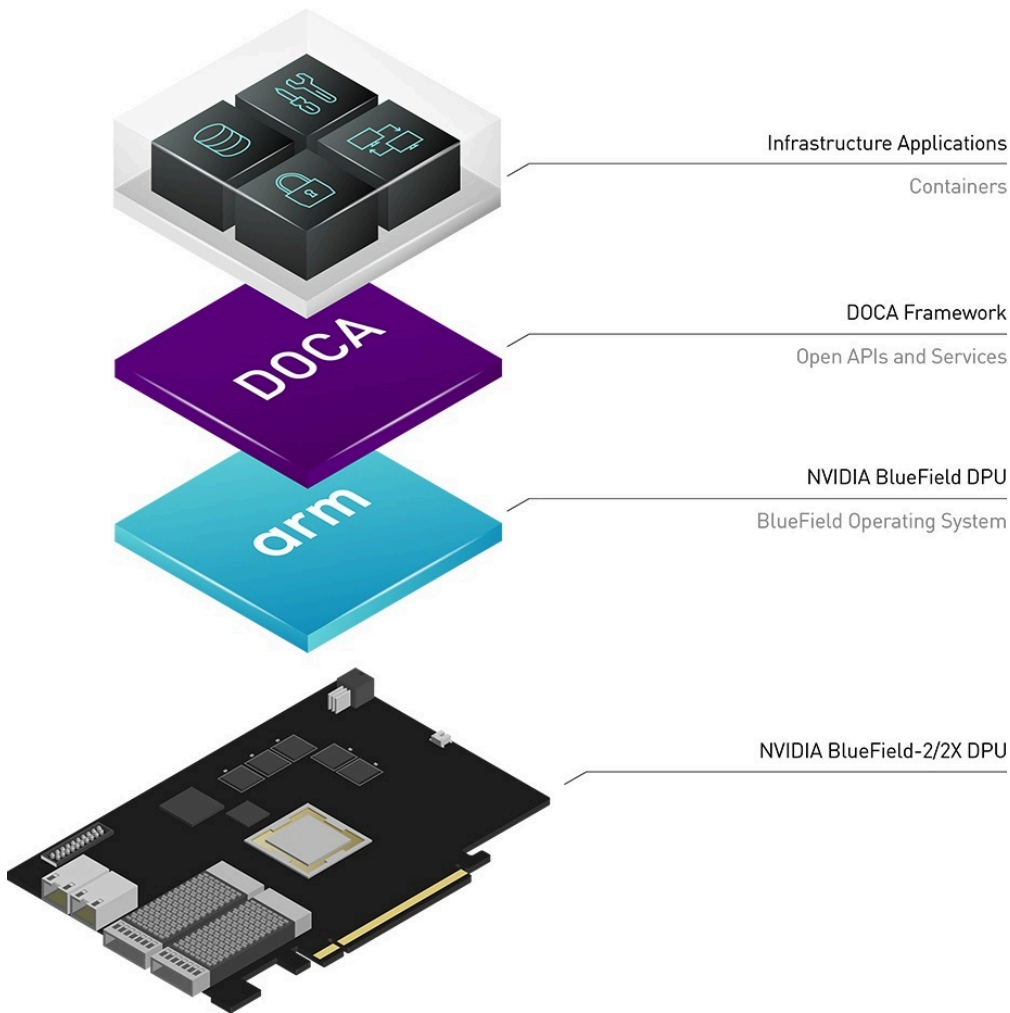
This guide provides a quick introduction to the NVIDIA® BlueField® networking platform, its DOCA software components, and BlueField user types.

Introduction

The BlueField family of networking platforms includes data processing units (DPUs) and SuperNICs, and is optimized for traditional enterprise, high-performance computing (HPC), and modern cloud workloads, delivering a broad set of accelerated software-defined networking, storage, security, and management services. BlueField enables organizations to transform their IT infrastructures into state-of-the-art data centers that are accelerated, fully programmable, and armed with zero-trust security to prevent data breaches and cyber-attacks.

NVIDIA DOCA™ brings together a wide range of powerful APIs, libraries, and frameworks for programming and acceleration of the modern data center infrastructure. Like

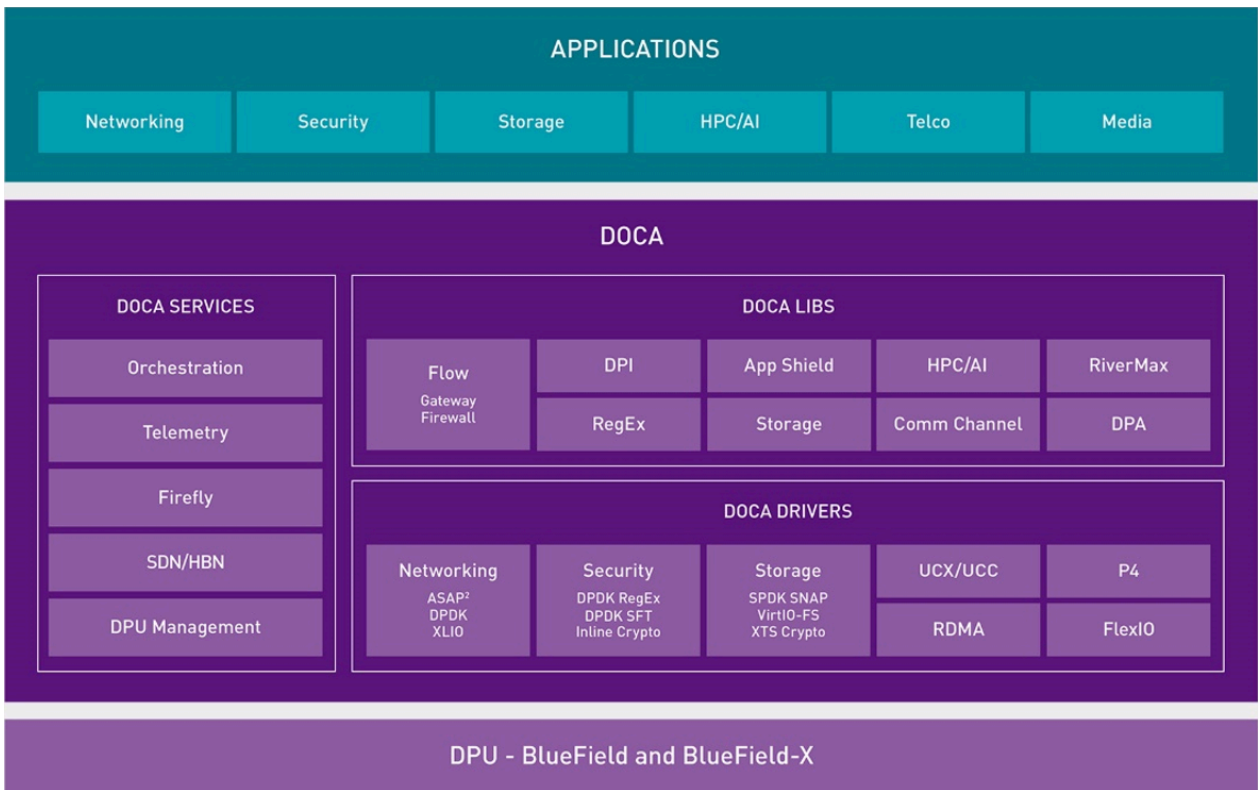
NVIDIA® CUDA® for GPUs, DOCA is a consistent and essential resource across all existing and future generations of BlueField products.



DOCA Components

DOCA software consists of a development and a runtime environment.

- DOCA-Devel provides industry-standard open APIs and frameworks, including Data Plane Development Kit (DPDK) and P4 for networking and security, and the Storage Performance Development Kit (SPDK) for storage. The frameworks simplify application offload with integrated NVIDIA acceleration packages. The Devel environment supports a range of operating systems and distributions and includes drivers, libraries, tools, documentation, and reference applications.

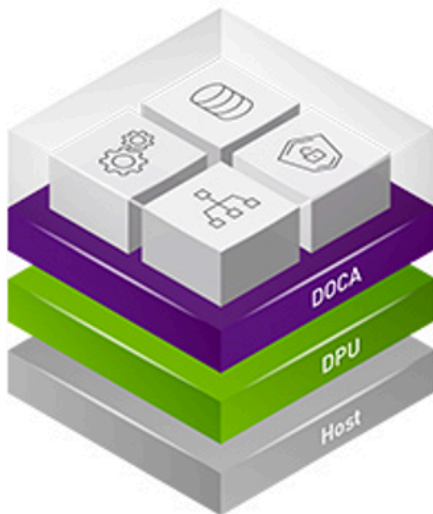


- **DOCA runtime** includes tools for provisioning, deploying, and orchestrating containerized services on BlueField Platforms in bulk across the data center.



Business Application Domain

Functional Isolation



Infrastructure Services Domain

BlueField Networking Platform User Types

BlueField Administrator

A BlueField administrator can be a system admin, an IT specialist, a security operations specialist, or anyone managing data center servers and their functionality. The admin would usually be interfacing with BlueField configuration and DOCA services and applications running on the BlueField Platform.

Common operations performed by the BlueField admin:

- Updating the BlueField image
- Running reference applications on the BlueField Platform
- Running DOCA services on the BlueField Platform

For more information, please visit [BlueField Administrator Quick Start Guide](#).

DOCA Developer

A DOCA developer creates the services and applications that run on top of the BlueField Platform and usually interfaces with DOCA libraries and drivers to create the necessary workflow and functionality.

Common operations performed by the DOCA developer:

- Developing DOCA applications using DOCA libraries and drivers
- Compiling DOCA reference applications
- Using DOCA sample code to create a new workflow

For more information, please refer to the [NVIDIA DOCA Developer Quick Start Guide](#).

NVIDIA DOCA EULA

NVIDIA DOCA SDK end-user license agreement.

End-User License Agreement

This license is a legal agreement between you and Mellanox Technologies, Ltd. ("NVIDIA Mellanox") and governs the use of the NVIDIA DOCA software and materials provided hereunder ("SOFTWARE").

This license can be accepted only by an adult of legal age of majority in the country in which the SOFTWARE is used. If you are under the legal age of majority, you must ask your parent or legal guardian to consent to this license. If you are entering this license on behalf of a company or other legal entity, you represent that you have legal authority and "you" will mean the entity you represent.

By using the SOFTWARE, you affirm that you have reached the legal age of majority, you accept the terms of this license, and you take legal and financial responsibility for the actions of your permitted users.

You agree to use the SOFTWARE only for purposes that are permitted by (a) this license, and (b) any applicable law, regulation or generally accepted practices or guidelines in the relevant jurisdictions.

1. LICENSE. Subject to the terms of this license, NVIDIA Mellanox hereby grants you a non-exclusive, non-transferable license, without the right to sublicense (except as expressly provided in this license) to:

1. Install and use the SOFTWARE,
2. Modify and create derivative works of sample or reference source code delivered in the SOFTWARE, and
3. Distribute the following portions of the SOFTWARE as incorporated in object code format into a software application, subject to the distribution requirements indicated in this license: API headers, drivers, libraries and sample applications.

BlueField SNAP software and materials, if delivered to you under this license, are licensed only for use in BlueField DPUs and subject to license fees Per DPU. "Per DPU" license means a license that allows concurrent authorized users to use the SOFTWARE in a single DPU under the license, and in some cases the SKU or documentation will indicate the maximum number of concurrent authorized users or virtual machines per DPU. Notwithstanding contrary terms in Section 1 above, you may not use or copy BlueField SNAP software without the necessary licenses.

2. DISTRIBUTION REQUIREMENTS. These are the distribution requirements for you to exercise the grants above:

1. An application must have material additional functionality, beyond the included portions of the SOFTWARE.
2. The following notice shall be included in modifications and derivative works of source code distributed: "This software contains source code provided by Mellanox Technologies Ltd."
3. You agree to distribute the SOFTWARE subject to the terms at least as protective as the terms of this license, including (without limitation) terms

relating to the license grant, license restrictions and protection of NVIDIA Mellanox's intellectual property rights. Additionally, you agree that you will protect the privacy, security and legal rights of your application users.

4. You agree to notify NVIDIA Mellanox in writing of any known or suspected distribution or use of the SOFTWARE not in compliance with the requirements of this license, and to enforce the terms of your agreements with respect to the distributed portions of the SOFTWARE.
3. **AUTHORIZED USERS.** You may allow employees and contractors of your entity or of your subsidiary(ies) to access and use the SOFTWARE from your secure network to perform work on your behalf. If you are an academic institution you may allow users enrolled or employed by the academic institution to access and use the SOFTWARE from your secure network. You are responsible for the compliance with the terms of this license by your authorized users.
4. **LIMITATIONS.** Your license to use the SOFTWARE is restricted as follows:
 1. The SOFTWARE is licensed for you to develop applications only for their use in systems with NVIDIA DPUs or adapter products or related adapter products.
 2. Except as provided in this Agreement, you may not modify, reverse engineer, decompile or disassemble, or remove copyright or other proprietary notices from any portion of the SOFTWARE or copies of the SOFTWARE.
 3. You may not disclose the results of benchmarking, competitive analysis, regression or performance data relating to the SOFTWARE without the prior written permission from NVIDIA Mellanox.
 4. Except as expressly provided in this license, you may not copy, sell, rent, sublicense, transfer, distribute, modify, or create derivative works of any portion of the SOFTWARE. For clarity, unless you have an agreement with NVIDIA Mellanox for this purpose you may not distribute or sublicense the SOFTWARE as a stand-alone product.
 5. Unless you have an agreement with NVIDIA Mellanox for this purpose, you may not indicate that an application created with the SOFTWARE is sponsored or endorsed by NVIDIA Mellanox.
 6. You may not bypass, disable, or circumvent any technical limitation, encryption, security, digital rights management or authentication mechanism in the SOFTWARE.

7. You may not replace any NVIDIA Mellanox software components in the SOFTWARE that are governed by this license with other software that implements NVIDIA Mellanox APIs.
 8. You may not use the SOFTWARE in any manner that would cause it to become subject to an open-source software license. As examples, licenses that require as a condition of use, modification, and/or distribution that the SOFTWARE be:
 - (i) disclosed or distributed in source code form;
 - (ii) licensed for the purpose of making derivative works; or
 - (iii) redistributable at no charge.
 9. Unless you have an agreement with NVIDIA Mellanox for this purpose, you may not use the SOFTWARE with any system or application where the use or failure of the system or application can reasonably be expected to threaten or result in personal injury, death, or catastrophic loss. Examples include use in avionics, navigation, military, medical, life support or other life critical applications. NVIDIA Mellanox does not design, test or manufacture the SOFTWARE for these critical uses and NVIDIA Mellanox shall not be liable to you or any third party, in whole or in part, for any claims or damages arising from such uses.
 10. You agree to defend, indemnify and hold harmless NVIDIA Mellanox and its affiliates, and their respective employees, contractors, agents, officers and directors, from and against any and all claims, damages, obligations, losses, liabilities, costs or debt, fines, restitutions and expenses (including but not limited to attorney's fees and costs incident to establishing the right of indemnification) arising out of or related to your use of the SOFTWARE outside of the scope of this license, or not in compliance with its terms.
5. UPDATES. NVIDIA Mellanox may, at its option, make available patches, workarounds or other updates to this SOFTWARE. Unless the updates are provided with their separate governing terms, they are deemed part of the SOFTWARE licensed to you as provided in this license. You agree that the form and content of the SOFTWARE that NVIDIA Mellanox provides may change without prior notice to you. While NVIDIA Mellanox generally maintains compatibility between versions, NVIDIA Mellanox may in some cases make changes that introduce incompatibilities in future versions of the SOFTWARE.
6. PRE-RELEASE VERSIONS. SOFTWARE versions identified as alpha, beta, preview, early access or otherwise as pre-release may not be fully functional, may contain errors or design flaws, and may have reduced or different security, privacy, availability, and reliability standards relative to commercial versions of NVIDIA

Mellanox software and materials. You may use a pre-release SOFTWARE version at your own risk, understanding that these versions are not intended for use in production or business-critical systems. NVIDIA Mellanox may choose not to make available a commercial version of any pre-release SOFTWARE. NVIDIA Mellanox may also choose to abandon development and terminate the availability of a pre-release SOFTWARE at any time without liability.

7. COMPONENTS UNDER OTHER LICENSES. The SOFTWARE may include NVIDIA Mellanox or third party components with separate legal notices or terms as may be described in proprietary notices accompanying the SOFTWARE, such as components governed by open source software licenses. If and to the extent there is a conflict between the terms in this license and the license terms associated with a component, the license terms associated with the components control only to the extent necessary to resolve the conflict.

8. OWNERSHIP

1. NVIDIA Mellanox reserves all rights, title and interest in and to the SOFTWARE not expressly granted to you under this license. NVIDIA Mellanox and its suppliers hold all rights, title and interest in and to the SOFTWARE, including their respective intellectual property rights. The SOFTWARE is copyrighted and protected by the laws of the United States and other countries, and international treaty provisions.

2. Subject to the rights of NVIDIA Mellanox and its suppliers in the SOFTWARE, you hold all rights, title and interest in and to your applications and your derivative works of the sample source code delivered in the SOFTWARE including their respective intellectual property rights.

9. FEEDBACK. You may, but are not obligated to, provide to NVIDIA Mellanox Feedback. "Feedback" means suggestions, fixes, modifications, feature requests or other feedback regarding the SOFTWARE. Feedback, even if designated as confidential by you, shall not create any confidentiality obligation for NVIDIA Mellanox. NVIDIA Mellanox and its designees have a perpetual, non-exclusive, worldwide, irrevocable license to use, reproduce, publicly display, modify, create derivative works of, license, sublicense, and otherwise distribute and exploit Feedback as NVIDIA Mellanox sees fit without payment and without obligation or restriction of any kind on account of intellectual property rights or otherwise.

10. NO WARRANTIES. THE SOFTWARE IS PROVIDED AS-IS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW NVIDIA MELLANOX AND ITS AFFILIATES EXPRESSLY

DISCLAIM ALL WARRANTIES OF ANY KIND OR NATURE, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. NVIDIA MELLANOX DOES NOT WARRANT THAT THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION THEREOF WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT ALL ERRORS WILL BE CORRECTED.

11. LIMITATIONS OF LIABILITY. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW NVIDIA MELLANOX AND ITS AFFILIATES SHALL NOT BE LIABLE FOR ANY SPECIAL, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES, OR FOR ANY LOST PROFITS, PROJECT DELAYS, LOSS OF USE, LOSS OF DATA OR LOSS OF GOODWILL, OR THE COSTS OF PROCURING SUBSTITUTE PRODUCTS, ARISING OUT OF OR IN CONNECTION WITH THIS LICENSE OR THE USE OR PERFORMANCE OF THE SOFTWARE, WHETHER SUCH LIABILITY ARISES FROM ANY CLAIM BASED UPON BREACH OF CONTRACT, BREACH OF WARRANTY, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR ANY OTHER CAUSE OF ACTION OR THEORY OF LIABILITY, EVEN IF NVIDIA MELLANOX HAS PREVIOUSLY BEEN ADVISED OF, OR COULD REASONABLY HAVE FORESEEN, THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT WILL NVIDIA MELLANOX'S AND ITS AFFILIATES TOTAL CUMULATIVE LIABILITY UNDER OR ARISING OUT OF THIS LICENSE EXCEED US\$10.00. THE NATURE OF THE LIABILITY OR THE NUMBER OF CLAIMS OR SUITS SHALL NOT ENLARGE OR EXTEND THIS LIMIT.
12. TERMINATION. Your rights under this license will terminate automatically without notice from NVIDIA Mellanox if you fail to comply with any term and condition of this license or if you commence or participate in any legal proceeding against NVIDIA Mellanox with respect to the SOFTWARE. NVIDIA Mellanox may terminate this license with advance written notice to you, if NVIDIA Mellanox decides to no longer provide the SOFTWARE in a country or, in NVIDIA Mellanox's sole discretion, the continued use of it is no longer commercially viable. Upon any termination of this license, you agree to promptly discontinue use of the SOFTWARE and destroy all copies in your possession or control. Your prior distributions in accordance with this license are not affected by the termination of this license. All provisions of this license will survive termination, except for the license granted to you.
13. APPLICABLE LAW. This license will be governed in all respects by the laws of the United States and of the State of Delaware, without regard to the conflicts of laws principles. The United Nations Convention on Contracts for the International Sale of Goods is specifically disclaimed. You agree to all terms of this license in the English language. The state or federal courts residing in Santa Clara County, California shall have exclusive jurisdiction over any dispute or claim arising out of this license.

Notwithstanding this, you agree that NVIDIA Mellanox shall still be allowed to apply for injunctive remedies or urgent legal relief in any jurisdiction.

14. NO ASSIGNMENT. This license and your rights and obligations thereunder may not be assigned by you by any means or operation of law without NVIDIA Mellanox's permission. Any attempted assignment not approved by NVIDIA Mellanox in writing shall be void and of no effect. NVIDIA Mellanox may assign, delegate or transfer this license and its rights and obligations, and if to a non-affiliate you will be notified.
15. EXPORT. The SOFTWARE is subject to United States export laws and regulations. You agree to comply with all applicable U.S. and international export laws, including the Export Administration Regulations (EAR) administered by the U.S. Department of Commerce and economic sanctions administered by the U.S. Department of Treasury's Office of Foreign Assets Control (OFAC). These laws include restrictions on destinations, end-users and end-use. By accepting this license, you confirm that you are not currently residing in a country or region currently embargoed by the U.S. and that you are not otherwise prohibited from receiving the SOFTWARE.
16. GOVERNMENT USE. The SOFTWARE is, and shall be treated as being, "Commercial Items" as that term is defined at 48 CFR § 2.101, consisting of "commercial computer software" and "commercial computer software documentation", respectively, as such terms are used in, respectively, 48 CFR § 12.212 and 48 CFR §§ 227.7202 & 252.227-7014(a)(1). Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions in this license pursuant to 48 CFR § 12.212 or 48 CFR § 227.7202. In no event shall the US Government user acquire rights in the SOFTWARE beyond those specified in 48 C.F.R. 52.227 19(b)(1) (2).
17. NOTICES. Please direct your legal notices or other correspondence to NVIDIA Corporation, 2788 San Tomas Expressway, Santa Clara, California 95051, United States of America, Attention: Legal Department and NBU legal_notices@exchange.nvidia.com.
18. ENTIRE AGREEMENT. This license is the final, complete and exclusive agreement between the parties relating to the subject matter of this license and supersedes all prior or contemporaneous understandings and agreements relating to this subject matter, whether oral or written. If any court of competent jurisdiction determines that any provision of this license is illegal, invalid or unenforceable, the remaining provisions will remain in full force and effect. Any amendment or waiver under this license shall be in writing and signed by representatives of both parties.

19. LICENSING. If the distribution terms in this license are not suitable for your organization, or for any questions regarding this license, please contact NVIDIA Mellanox at doca_license@nvidia.com.

Last updated: May 10, 2022

Quick Start for BlueField Developers

This section contains the following pages:

- [NVIDIA DOCA Developer Quick Start Guide](#)

NVIDIA DOCA Developer Quick Start Guide

This guide details the basic steps to bring up the NVIDIA DOCA development environment and to build and run the DOCA reference applications provided along with the DOCA software framework package.

Introduction

NVIDIA DOCA brings together a wide range of powerful APIs, libraries, and frameworks for programming and accelerating modern data center infrastructures. Like NVIDIA® CUDA® for GPUs, DOCA is a consistent and essential resource across all existing and future generations of BlueField DPU and SuperNIC products.

This document is intended for those wishing to develop applications using the DOCA framework.

Note

Not sure which installation type to use? To expand on different DOCA user types and the relevant installation for each, see [BlueField and DOCA User Types](#).

Install BlueField Networking Platform

Install the BlueField networking platform into your host according to the installation instructions in the [BlueField's hardware user guide](#). The steps include installing BlueField into the PCIe slot and properly securing it in the chassis. Make sure your host OS is listed under the [supported operating systems](#) section.

Install DOCA Software Package

A detailed step-by-step process for downloading and installing the required development software on both the host and BlueField can be found in the [NVIDIA DOCA Installation Guide for Linux](#).

During installation, you must change the default password, ubuntu, to access the NVIDIA® BlueField® networking platform.

Access BlueField

After a successful installation, on the host, the RShim driver exposes a virtual Ethernet device called `tmfifo_net0`.

1. Configure the host side of the `tmfifo_net0` with a static IP to enable IPv4-based communication to the BlueField OS according to the instructions on "[Host-side Interface Configuration](#)" in the *NVIDIA BlueField DPU BSP* document.
2. Log into BlueField's Ubuntu-based OS by running the following command from the host:

```
host# ssh ubuntu@192.168.100.2
```

Use the BlueField networking platform password you defined during the installation process.

At this stage DOCA is installed on BlueField and the host server.

Run Reference DOCA Application

DOCA package assets (e.g., references, tools) are located on Bluefield and on the host under `/opt/mellanox/doca/`.

The DOCA package includes a set of reference applications to facilitate developer onboarding. Please refer to the [DOCA Reference Applications](#) and [DOCA Programming Guide](#) for more information.

To run the [DOCA Secure Channel](#) reference application which demonstrates accelerated and secure message transmission between the host and BlueField over the Comm Channel interface:

1. Run the application as server on the BlueField networking platform using the following command (all parameters are available in the [secure channel application guide](#)):

```
# /opt/mellanox/doca/applications/secure_channel/bin/doca_secure_channel -s 256 -n 10 -p 03:00.0 -r 3b:00.0
```

2. Run the application as client on the host using the following command (all parameters are available in the [secure channel application guide](#)):

```
# /opt/mellanox/doca/applications/secure_channel/bin/doca_secure_channel -s 256 -n 10 -p 3b:00.0
```

More Information

To learn more about NVIDIA BlueField networking platforms, refer to the [NVIDIA BlueField Hardware Manuals](#).

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

Installation and Setup

This section contains the following page:

- [NVIDIA DOCA Profiles](#)
- [NVIDIA DOCA Installation Guide for Linux](#)
- [NVIDIA DOCA Developer Guide](#)

NVIDIA DOCA Profiles

The following document provides an introduction to the various supported DOCA-Host profiles.

Introduction

NVIDIA DOCA™ can be installed on the host and used by a variety of customers who have different workloads and requirements. The DOCA-Host package includes drivers, libraries, and tools to support NVIDIA® BlueField® Networking Platform and NVIDIA® ConnectX® SmartNIC, Ethernet and InfiniBand, with both kernel and user-space components. Depending on their specific needs, customers may choose not to install the full DOCA-Host package on their host server but only the subset of components and tools relevant for their use case (whether to have a smaller installation size, lower integration/validation effort, etc).

To support the different use cases, DOCA includes DOCA-Host Installation Profiles, which are a subset of the full DOCA installation. DOCA-Host profiles are validated and tested installation packages. The following are the available DOCA profiles:

- doca-all
- doca-networking
- doca-Ofed

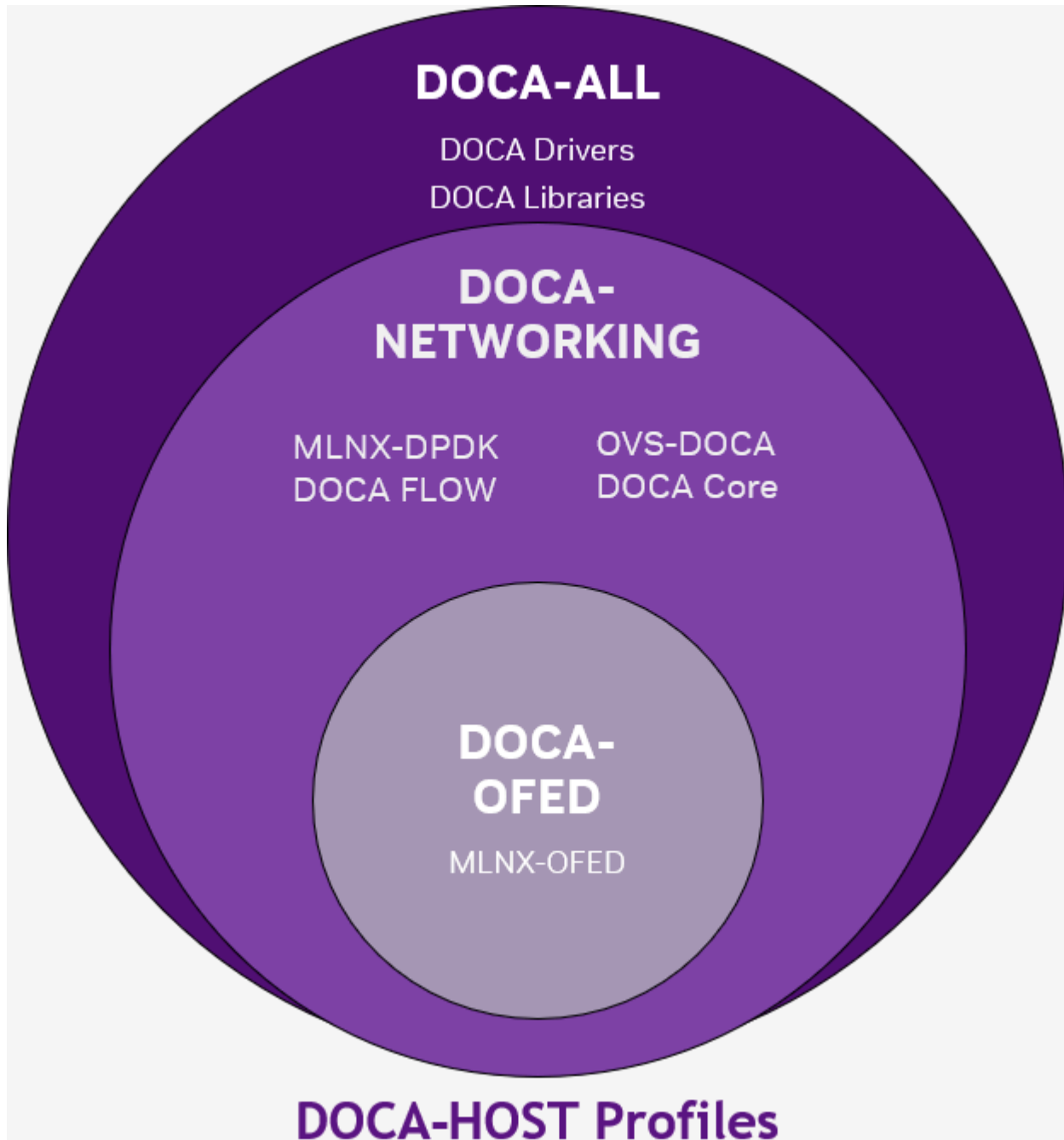
DOCA-Host supports the following NVIDIA devices:

- BlueField-3
- BlueField-2
- ConnectX-7
- ConnectX-6 DX
- ConnectX-6 LX
- ConnectX-6
- ConnectX-5
- ConnectX-4 LX
- ConnectX-4

For hardware details on these devices, refer to the following pages:

- [BlueField devices](#)
- [ConnectX devices](#)

DOCA functionality is limited by the specific device capabilities.



doca-all

The full DOCA-Host installation is intended for users who wish to utilize the full extent of DOCA libs and drivers.

This profile is the super-set of components, which also includes the content of `doca-oped` and `doca-networking`.

All DOCA libraries, drivers and tools are included in doca-all.

Info

When installing doca-all on host, BlueField Platforms can utilize all DOCA libs and drivers whereas ConnectX devices can utilize only doca-ofed and doca-networking subset of functions from within the super-set of doca-all, depending on the device's capabilities.

doca-networking

The doca-networking profile is intended for users who wish to benefit only from the networking functionality of DOCA.

The content of the doca-networking package is the following:

- MLNX_OFED
- DOCA Core
- MLNX-DPDK
- OVS-DOCA
- DOCA Flow

Info

BlueField DPUs, BlueField SuperNICs, and ConnectX devices can utilize all included libs and drivers in the doca-networking profile, based on the device's capabilities.

doca-ofed

This profile is intended for users who wish to have the same user experience and content as MLNX_OFED but with DOCA package. doca-ofed installs the MLNX_OFED drivers and

tools and does not include any other DOCA components.

The content of the `doca-oped` package is:

- `MLNX_OFED` drivers and tools

Info

BlueField Platforms and ConnectX devices can utilize only the drivers in `doca-oped`, based on the device's capabilities. No added DOCA libs are supported with any of the devices with `doca-oped` profile installation.

Which Profile to Install?

Selecting the right DOCA-Host installation profile is important to fully utilize the capabilities of your BlueField Platforms or ConnectX.

The functionality of DOCA-Host is limited by the device capabilities (e.g., ConnectX devices cannot utilize DOCA libs such as DPA, even if `doca-all` is installed on the host).

For BlueField devices:

- It is recommended to use `doca-all`
- If you require the smallest installation package for networking-only purposes, use `doca-networking`
- For `MLNX_OFED`-like installation, use `doca-oped` (no additional DOCA functionality)

For ConnectX devices:

- It is recommended to use `doca-networking`
- For future-proof and mixed BlueField/ConnectX deployments, use `doca-all`
- For `MLNX_OFED`-like installation use `doca-oped` (no additional DOCA functionality)

DOCA-Host Profile Installation

DOCA-Host can be installed on specific host OSs. Each of the Host Installation Profiles has specific OSs on which it can be installed as specified in section "[Supported Host OS per DOCA-Host Installation Profile](#)".

Follow the instructions under section "[Installing Software on Host](#)" in the *NVIDIA DOCA Installation Guide for Linux*.

Supported Host OS per DOCA-Host Installation Profile

The default operating system included with the BlueField Bundle (for DPU and SuperNIC) is Ubuntu 22.04.

The supported operating systems on the host machine per DOCA-Host installation profile are the following:

Note

Only the following generic kernel versions are supported for DOCA local repo package for host installation.

OS	OS Version	Default Kernel Version	Arch	doca-all	doca-networking	doca-oped
Alinux	3.2	5.10.134-13.al8.x86_64	x86			
Anolis	8.6	5.10.134+	aarch64			
			x86			
BCLinux	21.10SP2	4.19.90-2107.6.0.0098.oe1.bclinux.aarch64	aarch64			
		4.19.90-2107.6.0.0100.oe1.bclinux.x86_64	x86			

OS	OS Version	Default Kernel Version	Arch	doca-all	doca-networking	doca-oped
		64				
CTYunOS	2.0	4.19.90-2102.2.0.0062.ctl2.aarch64	aarch64			
		4.19.90-2102.2.0.0062.ctl2.x86_64	x86			
	3.0 (23.01)	5.10.0-136.12.0.86.ctl3.aarch64	aarch64			
		5.10.0-136.12.0.86.ctl3.x86_64	x86			
Debian	10.13	4.19.0-21-arm64	aarch64			
		4.19.0-21-amd64	x86			
	10.8	4.19.0-14-arm64	aarch64			
		4.19.0-14-amd64	x86			
	10.9	4.19.0-16-amd64	x86			
	11.3	5.10.0-13-arm64	aarch64			
		5.10.0-13-amd64	x86			
	12.1	6.1.0-10-arm64	aarch64			
6.1.0-10-amd64		x86				
EulerOS	2.0sp11	5.10.0-60.18.0.50.h323.eulerosv2r11.aarch64	aarch64			
		5.10.0-60.18.0.50.h323.eulerosv2r11.x86_64	x86			
	2.0sp12	5.10.0-136.12.0.86.h1032.eulerosv2r12.aarch64	aarch64			

OS	OS Version	Default Kernel Version	Arch	doca-all	doca-networking	doca-oped
		5.10.0-136.12.0.86.h1032.eulerosv2r12.x86_64	x86			
Kylin	10sp2	4.19.90-24.4.v2101.ky10.aarch64	aarch64			
		4.19.90-24.4.v2101.ky10.x86_64	x86			
	10sp3	4.19.90-52.22.v2207.ky10.aarch64	aarch64			
		4.19.90-52.22.v2207.ky10.x86_64	x86			
Mariner	2.0	5.15.118.1-1.cm2.x86_64	x86			
Oracle Linux	7.9	5.4.17-2011.6.2.el7uek.x86_64	x86			
	8.4	5.4.17-2102.201.3.el8uek.x86_64	x86			
	8.6	5.4.17-2136.307.3.1.el8uek.x86_64	x86			
	8.7	5.15.0-3.60.5.1.el8uek.x86_64	x86			
	8.8	5.15.0-101.103.2.1.el8uek.x86_64	x86			
	9.1	5.15.0-3.60.5.1.el9uek.x86_64	x86			
	9.2	5.15.0-101.103.2.1.el9uek.x86_64	x86			
openEuler	20.03sp3	4.19.90-2112.8.0.0131.oe1.aarch64	aarch64			
		4.19.90-2112.8.0.0131.oe1.x86_64	x86			
	22.03	5.10.0-60.18.0.50.oe2203.aarch64	aarch64			
		5.10.0-60.18.0.50.oe2203.x86_64	x86			

OS	OS Version	Default Kernel Version	Arch	doca-all	doca-networking	doca-oped	
RHEL/CentOS	8.0	4.18.0-80.el8.aarch64	aarch64				
		4.18.0-80.el8.x86_64	x86				
	8.1	4.18.0-147.el8.aarch64	aarch64				
		4.18.0-147.el8.x86_64	x86				
	8.2	4.18.0-193.el8.aarch64	aarch64				
		4.18.0-193.el8.x86_64	x86				
	8.3	4.18.0-240.el8.aarch64	aarch64				
		4.18.0-240.el8.x86_64	x86				
	8.4	4.18.0-305.el8.aarch64	aarch64				
		4.18.0-305.el8.x86_64	x86				
	RHEL/Rocky	8.5	4.18.0-348.el8.aarch64	aarch64			
			4.18.0-348.el8.x86_64	x86			
8.6		4.18.0-372.41.1.el8_6.aarch64	aarch64				
		4.18.0-372.41.1.el8_6.x86_64	x86				
8.7		4.18.0-425.14.1.el8_7.aarch64	aarch64				
		4.18.0-425.14.1.el8_7.x86_64	x86				
8.8		4.18.0-477.10.1.el8_8.aarch64	aarch64				
		4.18.0-477.10.1.el8_8.x86_64	x86				
8.9		4.18.0-513.5.1.el8_9.aarch64	aarch64				

OS	OS Version	Default Kernel Version	Arch	doca-all	doca-networking	doca-oped
	9.0	4.18.0-513.5.1.el8_9.x86_64	x86			
		5.14.0-70.46.1.el9_0.aarch64	aarch64			
	9.1	5.14.0-70.46.1.el9_0.x86_64	x86			
		5.14.0-162.19.1.el9_1.aarch64	aarch64			
	9.2	5.14.0-162.19.1.el9_1.x86_64	x86			
		5.14.0-284.11.1.el9_2.aarch64	aarch64			
	9.3	5.14.0-284.11.1.el9_2.x86_64	x86			
		5.14.0-362.8.1.el9_3.aarch64	aarch64			
SLES	15sp2	5.3.18-22-default	aarch64			
			x86			
	15sp3	5.3.18-57-default	aarch64			
			x86			
	15sp4	5.14.21-150400.22-default	aarch64			
			x86			
	15sp5	5.14.21-150500.53-default	aarch64			
			x86			
TKLinux	3.3	5.4.119-19.0009.39	aarch64			
		5.4.119-19.0009.39	x86			

OS	OS Version	Default Kernel Version	Arch	doca-all	doca-networking	doca-oped
Ubuntu	20.04	5.4.0-26-generic	aarch64			
			x86			
	22.04	5.15.0-25-generic	aarch64			
			x86			
	24.04	6.8.0-31-generic	aarch64			
			x86			
UOS	20.1060a	5.10.0-46.uelc20.aarch64	aarch64			
		5.10.0-46.uelc20.x86_64	x86			
	20.1060e	5.10.0-46.uel20.aarch64	aarch64			
		5.10.0-46.uel20.x86_64	x86			

NVIDIA DOCA Installation Guide for Linux

This guide details the necessary steps to set up NVIDIA DOCA in your Linux environment.

Introduction

Installation of the NVIDIA® BlueField® networking platform (DPU or SuperNIC) software requires following the following [step-by-step procedure](#).

Supported Platforms

Supported BlueField Platforms

The following NVIDIA® BlueField® Platforms are supported with DOCA:

NVIDIA SKU	Legacy OPN	PSID	Description
900-9D3B6-00CV-AA0	N/A	MT_000000884	BlueField-3 B3220 P-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B6-00SV-AA0	N/A	MT_000000965	BlueField-3 B3220 P-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3B6-00CC-AA0	N/A	MT_000001024	BlueField-3 B3210 P-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B6-00SC-AA0	N/A	MT_000001025	BlueField-3 B3210 P-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Disabled
900-9D219-0086-ST1	MBF2 M516A - CECOT	MT_000000375	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto and Secure Boot Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0086-ST0	MBF2 M516A - EECOT	MT_000000376	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto and Secure Boot Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0056-ST1	MBF2 M516A - EENOT	MT_000000377	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D206-	MBF2 H332A	MT_000000539	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; HHHL

NVIDIA SKU	Legacy OPN	PSID	Description
0053-SQ0	- AENOT		
900-9D206-0063-ST2	MBF2 H332A - AEEOT	MT_0 00000 0540	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D206-0083-ST3	MBF2 H332A - AECOT	MT_0 00000 0541	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto and Secure Boot Enabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D206-0083-ST1	MBF2 H322A - AECOT	MT_0 00000 0542	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto and Secure Boot Enabled; 8GB on-board DDR; 1GbE OOB management; HHHL
900-9D206-0063-ST1	MBF2 H322A - AEEOT	MT_0 00000 0543	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto Enabled; 8GB on-board DDR; 1GbE OOB management; HHHL
900-9D219-0066-ST0	MBF2 M516A -EEEOT	MT_0 00000 0559	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0056-SN1	MBF2 M516A - CENOT	MT_0 00000 0560	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0066-ST2	MBF2 M516A - CEEOT	MT_0 00000 0561	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0006-ST0	MBF2 H516A - CEEOT	MT_0 00000 0702	BlueField-2 DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto; 16GB on-board DDR; 1GbE OOB management; FHHL

NVIDIA SKU	Legacy OPN	PSID	Description
900-9D219-0056-ST2	MBF2 H516A - CENOT	MT_0 00000 0703	BlueField-2 DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0066-ST3	MBF2 H516A -EEEOT	MT_0 00000 0704	BlueField-2 DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0056-SQ0	MBF2 H516A - EENOT	MT_0 00000 0705	BlueField-2 DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D250-0038-ST1	MBF2 M345A - HESOT	MT_0 00000 0715	BlueField-2 E-Series DPU; 200GbE/HDR single-port QSFP56; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D250-0048-ST1	MBF2 M345A - HECOT	MT_0 00000 0716	BlueField-2 E-Series DPU; 200GbE/HDR single-port QSFP56; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D218-0073-ST1	MBF2 H512C - AESOT	MT_0 00000 0723	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D218-0083-ST2	MBF2 H512C - AECOT	MT_0 00000 0724	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0086-ST4	MBF2 M516C - EECOT	MT_0 00000 0728	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-	MBF2 H516C	MT_0 00000	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto

NVIDIA SKU	Legacy OPN	PSID	Description
0086-SQ0	- CECOT	0729	Enabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0076-ST5	MBF2 M516C - CESOT	MT_0 00000 0731	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0076-ST6	MBF2 M516C -EESOT	MT_0 00000 0732	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0086-ST3	MBF2 M516C - CECOT	MT_0 00000 0733	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0076-ST2	MBF2 H516C -EESOT	MT_0 00000 0737	BlueField-2 P-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0076-ST1	MBF2 H516C - CESOT	MT_0 00000 0738	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D218-0083-ST4	MBF2 H532C - AECOT	MT_0 00000 0765	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Enabled; 32GB on-board DDR; 1GbE OOB management; FHHL
900-9D218-0073-ST0	MBF2 H532C - AESOT	MT_0 00000 0766	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Disabled; 32GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0076-ST3	MBF2 H536C - CESOT	MT_0 00000 0767	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 32GB on-board DDR; 1GbE OOB management; FHHL

NVIDIA SKU	Legacy OPN	PSID	Description
900-9D208-0086-ST2	MBF2 H536C - CECOT	MT_0 00000 0768	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 32GB on-board DDR; 1GbE OOB management; FHHL
900-9D218-0073-ST4	MBF2 H512C - AEUOT	MT_0 00000 0972	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled with UEFI disabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management
900-9D208-0076-STA	MBF2 H516C - CEUOT	MT_0 00000 0973	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled with UEFI disabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management
900-9D208-0076-STB	MBF2 H536C - CEUOT	MT_0 00000 1008	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56, integrated BMC, PCIe Gen4 x16, Secure Boot Enabled with UEFI Disabled, Crypto Disabled, 32GB on-board DDR, 1GbE OOB management, Tall Bracket, FHHL
P1004/699210040230	N/A	NVDO 00000 0015	BlueField-2 A30X, P1004 SKU 205, Generic, GA100, 24GB HBM2e, PCIe passive Dual Slot 230W GEN4, DPU Crypto ON W/ Bkt, 1 Dongle, Black, HF, VCPD
P4028/699140280000	N/A	NVDO 00000 0020	ZAM / NAS

Supported ConnectX NICs

The NVIDIA® ConnectX® NICs supported with DOCA-Host can be found in: [NVIDIA DOCA Profiles](#)

Hardware Prerequisites

For BlueField Platform users, this guide assumes that a BlueField device has been installed in a server according to the instructions detailed in your [DPU's hardware user guide](#).

DOCA Packages

Device	Component	Version	Description
Host	DOCA Devel	2.7.0	Software development kit package and tools for developing host software
	DOCA Runtime	2.7.0	Runtime libraries and tools required to run DOCA-based software applications on host
	DOCA Extra	2.7.0	Contains helper scripts (doca-info, doca-kernel-support)
	DOCA OFED	2.7.0	Software stack which operates across all NVIDIA network adapter solutions
	Arm emulated (QEMU) development container	4.7.0	Linux-based BlueField Arm emulated container for developers
Target BlueField DPU (Arm)	BlueField BSP	4.7.0	BlueField image and firmware
	DOCA SDK	2.7.0	Software development kit packages and tools for developing Arm software
	DOCA Runtime	2.7.0	Runtime libraries and tools required to run DOCA-based software applications on Arm

Supported Host OS per DOCA-Host Installation Profile

The default operating system included with the BlueField Bundle (for DPU and SuperNIC) is Ubuntu 22.04.

The supported operating systems on the host machine per DOCA-Host installation profile are the following:

Note

Only the following generic kernel versions are supported for DOCA local repo package for host installation.

OS	OS Version	Default Kernel Version	Arch	doca-all	doca-networking	doca-oped
Alinux	3.2	5.10.134-13.al8.x86_64	x86			
Anolis	8.6	5.10.134+	aarch64			
			x86			
BCLinux	21.10SP2	4.19.90-2107.6.0.0098.oe1.bclin ux.aarch64	aarch64			
		4.19.90-2107.6.0.0100.oe1.bclin ux.x86_64	x86			

CTYunOS	2.0	4.19.90-2102.2.0.0062.ctl2.aarch64	aarch64			
		4.19.90-2102.2.0.0062.ctl2.x86_64	x86			
	3.0 (23.01)	5.10.0-136.12.0.86.ctl3.aarch64	aarch64			
		5.10.0-136.12.0.86.ctl3.x86_64	x86			
Debian	10.13	4.19.0-21-arm64	aarch64			
		4.19.0-21-amd64	x86			
	10.8	4.19.0-14-arm64	aarch64			
		4.19.0-14-amd64	x86			
	10.9	4.19.0-16-amd64	x86			
	11.3	5.10.0-13-arm64	aarch64			
		5.10.0-13-amd64	x86			
	12.1	6.1.0-10-arm64	aarch64			
6.1.0-10-amd64		x86				
EulerOS	2.0sp11	5.10.0-60.18.0.50.h323.eulerosv2r11.aarch64	aarch64			
		5.10.0-60.18.0.50.h323.eulerosv2r11.x86_64	x86			
	2.0sp12	5.10.0-136.12.0.86.h1032.eulerosv2r12.aarch64	aarch64			

		5.10.0-136.12.0.86.h1032.eulerosv2r12.x86_64	x86			
Kylin	10sp2	4.19.90-24.4.v2101.ky10.aarch64	aarch64			
		4.19.90-24.4.v2101.ky10.x86_64	x86			
	10sp3	4.19.90-52.22.v2207.ky10.aarch64	aarch64			
		4.19.90-52.22.v2207.ky10.x86_64	x86			
Mariner	2.0	5.15.118.1-1.cm2.x86_64	x86			
Oracle Linux	7.9	5.4.17-2011.6.2.el7uek.x86_64	x86			
	8.4	5.4.17-2102.201.3.el8uek.x86_64	x86			
	8.6	5.4.17-2136.307.3.1.el8uek.x86_64	x86			
	8.7	5.15.0-3.60.5.1.el8uek.x86_64	x86			
	8.8	5.15.0-101.103.2.1.el8uek.x86_64	x86			
	9.1	5.15.0-3.60.5.1.el9uek.x86_64	x86			
	9.2	5.15.0-101.103.2.1.el9uek.x86_64	x86			

openEuler	20.03sp3	4.19.90-2112.8.0.0131.oe1.aarch64	aarch64			
		4.19.90-2112.8.0.0131.oe1.x86_64	x86			
	22.03	5.10.0-60.18.0.50.oe2203.aarch64	aarch64			
		5.10.0-60.18.0.50.oe2203.x86_64	x86			
RHEL/CentOS	8.0	4.18.0-80.el8.aarch64	aarch64			
		4.18.0-80.el8.x86_64	x86			
	8.1	4.18.0-147.el8.aarch64	aarch64			
		4.18.0-147.el8.x86_64	x86			
	8.2	4.18.0-193.el8.aarch64	aarch64			
		4.18.0-193.el8.x86_64	x86			
	8.3	4.18.0-240.el8.aarch64	aarch64			
		4.18.0-240.el8.x86_64	x86			
8.4	4.18.0-305.el8.aarch64	aarch64				
	4.18.0-305.el8.x86_64	x86				
RHEL/Rocky	8.5	4.18.0-348.el8.aarch64	aarch64			
		4.18.0-348.el8.x86_64	x86			
	8.6	4.18.0-372.41.1.el8_6.aarch64	aarch64			

		4.18.0-372.41.1.el8_6.x86_64	x86			
	8.7	4.18.0-425.14.1.el8_7.aarch64	aarch64			
		4.18.0-425.14.1.el8_7.x86_64	x86			
	8.8	4.18.0-477.10.1.el8_8.aarch64	aarch64			
		4.18.0-477.10.1.el8_8.x86_64	x86			
	8.9	4.18.0-513.5.1.el8_9.aarch64	aarch64			
		4.18.0-513.5.1.el8_9.x86_64	x86			
	9.0	5.14.0-70.46.1.el9_0.aarch64	aarch64			
		5.14.0-70.46.1.el9_0.x86_64	x86			
	9.1	5.14.0-162.19.1.el9_1.aarch64	aarch64			
		5.14.0-162.19.1.el9_1.x86_64	x86			
	9.2	5.14.0-284.11.1.el9_2.aarch64	aarch64			
		5.14.0-284.11.1.el9_2.x86_64	x86			
	9.3	5.14.0-362.8.1.el9_3.aarch64	aarch64			
		5.14.0-362.8.1.el9_3.x86_64	x86			
RHEL/Rocky	9.4	5.14.0-427.13.1.el9_4.aarch64	aarch64			

		5.14.0-427.13.1.el9_4.x86_64	x86			
SLES	15sp2	5.3.18-22-default	aarch64			
			x86			
	15sp3	5.3.18-57-default	aarch64			
			x86			
	15sp4	5.14.21-150400.22-default	aarch64			
			x86			
	15sp5	5.14.21-150500.53-default	aarch64			
			x86			
TKLinux	3.3	5.4.119-19.0009.39	aarch64			
		5.4.119-19.0009.39	x86			
Ubuntu	20.04	5.4.0-26-generic	aarch64			
			x86			
	22.04	5.15.0-25-generic	aarch64			
			x86			
	24.04	6.8.0-31-generic	aarch64			
			x86			
UOS	20.1060a	5.10.0-46.uelc20.aarch64	aarch64			
		5.10.0-46.uelc20.x86_64	x86			

	20.1060e	5.10.0-46.uel20.aarch64	aarch64			
		5.10.0-46.uel20.x86_64	x86			

BlueField Networking Platform Image Installation

This guide provides the minimal instructions for setting up DOCA on a standard system.

Note

Make sure to follow the instructions in this section sequentially. Make sure to update DOCA on the host side first before installing the BFB Bundle on the BlueField.

Installation Files

Device	Component	OS	Arch	Link
Host	These files contain the following components suitable for their respective OS version. <ul style="list-style-type: none"> • DOCA Devel v2.7.0 • DOCA Runtime v2.7.0 • DOCA Extra v2.7.0 • DOCA OFED v2.7.0 	Alinux 3.2	x86	doca-host-2.7.0-204000_24.04_alinux32.x86_64.rpm
		Anolis	aarch64	doca-host-2.7.0-204000_24.04_anolis86.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_anolis86.x86_64.rpm
		BCLinux 21.10 SP2	aarch64	doca-host-2.7.0-204000_24.04_bclinux2110sp2.aarch64.rpm

Device	Component	OS	Arch	Link
			x86	doca-host-2.7.0-204000_24.04_bclinux2110sp2.x86_64.rpm
		CTyun OS 2.0	arch64	doca-host-2.7.0-204000_24.04_ctyunos20.arch64.rpm
			x86	doca-host-2.7.0-204000_24.04_ctyunos20.x86_64.rpm
		CTyun OS 23.01	arch64	doca-host-2.7.0-204000_24.04_ctyunos2301.arch64.rpm
			x86	doca-host-2.7.0-204000_24.04_ctyunos2301.x86_64.rpm
		Debian 10.13	arch64	doca-host_2.7.0-204000-24.04-debian1013_arm64.deb
			x86	doca-host_2.7.0-204000-24.04-debian1013_amd64.deb
		Debian 10.8	arch64	doca-host_2.7.0-204000-24.04-debian108_arm64.deb
			x86	doca-host_2.7.0-204000-24.04-debian108_amd64.deb
		Debian 10.9	x86	doca-host_2.7.0-204000-24.04-debian109_amd64.deb
		Debian 11.3	arch64	doca-host_2.7.0-204000-24.04-debian113_arm64.deb

Device	Component	OS	Arch	Link
			x86	doca-host_2.7.0-204000-24.04-debian113_amd64.deb
		Debian 12.1	arch64	doca-host_2.7.0-204000-24.04-debian121_arm64.deb
			x86	doca-host_2.7.0-204000-24.04-debian121_amd64.deb
		EulerOS 20 SP11	arch64	doca-host-2.7.0-204000_24.04_euleros20sp11.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_euleros20sp11.x86_64.rpm
		EulerOS 20 SP12	arch64	doca-host-2.7.0-204000_24.04_euleros20sp12.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_euleros20sp12.x86_64.rpm
		Fedora 32	x86	doca-host-2.7.0-204000_24.04_fc32.x86_64.rpm
		Kylin 1.0 SP2	arch64	doca-host-2.7.0-204000_24.04_kylin10sp2.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_kylin10sp2.x86_64.rpm
		Kylin 1.0 SP3	arch64	doca-host-2.7.0-204000_24.04_kylin10sp3.aarch64.rpm

Device	Component	OS	Arch	Link
			x86	doca-host-2.7.0-204000_24.04_kylin10sp3.x86_64.rpm
		Mariner 2.0	x86	doca-host-2.7.0-204000_24.04_mariner20.x86_64.rpm
		Oracle Linux 7.9	x86	doca-host-2.7.0-204000_24.04_ol79.x86_64.rpm
		Oracle Linux 8.4	x86	doca-host-2.7.0-204000_24.04_ol84.x86_64.rpm
		Oracle Linux 8.6	x86	doca-host-2.7.0-204000_24.04_ol86.x86_64.rpm
		Oracle Linux 8.7	x86	doca-host-2.7.0-204000_24.04_ol87.x86_64.rpm
		Oracle Linux 8.8	x86	doca-host-2.7.0-204000_24.04_ol88.x86_64.rpm
		Oracle Linux 9.0	x86	doca-host-2.7.0-204000_24.04_ol90.x86_64.rpm
		Oracle Linux 9.1	x86	doca-host-2.7.0-204000_24.04_ol91.x86_64.rpm
		Oracle Linux 9.2	x86	doca-host-2.7.0-204000_24.04_ol92.x86_64.rpm
		openEuler	aarch64	doca-host-2.7.0-204000_24.04_openeuler2003sp3.aarch64.rpm

Device	Component	OS	Arch	Link
		20.03 SP3	x86	doca-host-2.7.0-204000_24.04_openeuler2003sp3.x86_64.rpm
		openEuler 22.03	aarch64	doca-host-2.7.0-204000_24.04_openeuler2203.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_openeuler2203.x86_64.rpm
		RHEL/CentOS 8.0	aarch64	doca-host-2.7.0-204000_24.04_rhel80.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_rhel80.x86_64.rpm
		RHEL/CentOS 8.1	aarch64	doca-host-2.7.0-204000_24.04_rhel81.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_rhel81.x86_64.rpm
		RHEL/CentOS 8.2	aarch64	doca-host-2.7.0-204000_24.04_rhel82.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_rhel82.x86_64.rpm
		RHEL/CentOS 8.3	aarch64	doca-host-2.7.0-204000_24.04_rhel83.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_rhel83.x86_64.rpm

Device	Component	OS	Arch	Link
		RHEL/CentOS 8.4	aarch64	doca-host-2.7.0-204000_24.04_rhel84.aarch64.rpm
			x86_64	doca-host-2.7.0-204000_24.04_rhel84.x86_64.rpm
		RHEL/CentOS 8.5	aarch64	doca-host-2.7.0-204000_24.04_rhel85.aarch64.rpm
			x86_64	doca-host-2.7.0-204000_24.04_rhel85.x86_64.rpm
		RHEL/Rocky 8.6	aarch64	doca-host-2.7.0-204000_24.04_rhel86.aarch64.rpm
			x86_64	doca-host-2.7.0-204000_24.04_rhel86.x86_64.rpm
		RHEL/Rocky 8.7	aarch64	doca-host-2.7.0-204000_24.04_rhel87.aarch64.rpm
			x86_64	doca-host-2.7.0-204000_24.04_rhel87.x86_64.rpm
		RHEL/Rocky 8.8	aarch64	doca-host-2.7.0-204000_24.04_rhel88.aarch64.rpm
			x86_64	doca-host-2.7.0-204000_24.04_rhel88.x86_64.rpm
		RHEL/Rocky 8.9	aarch64	doca-host-2.7.0-204000_24.04_rhel89.aarch64.rpm

Device	Component	OS	Arch	Link
			x86	doca-host-2.7.0-204000_24.04_rhel89.x86_64.rpm
		RHEL/Rocky 8.10	aarch64	doca-host-2.7.0-204000_24.04_rhel810.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_rhel810.x86_64.rpm
		RHEL/Rocky 9.0	aarch64	doca-host-2.7.0-204000_24.04_rhel90.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_rhel90.x86_64.rpm
		RHEL/Rocky 9.1	aarch64	doca-host-2.7.0-204000_24.04_rhel91.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_rhel91.x86_64.rpm
		RHEL/Rocky 9.2	aarch64	doca-host-2.7.0-204000_24.04_rhel92.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_rhel92.x86_64.rpm
		RHEL/Rocky 9.3	aarch64	doca-host-2.7.0-204000_24.04_rhel93.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_rhel93.x86_64.rpm

Device	Component	OS	Arch	Link
		RHEL/Rocky 9.4	aarch64	doca-host-2.7.0-204000_24.04_rhel94.aarch64.rpm
			x86_64	doca-host-2.7.0-204000_24.04_rhel94.x86_64.rpm
		SLES 15 SP2	aarch64	doca-host-2.7.0-204000_24.04_sles15sp2.aarch64.rpm
			x86_64	doca-host-2.7.0-204000_24.04_sles15sp2.x86_64.rpm
		SLES 15 SP3	aarch64	doca-host-2.7.0-204000_24.04_sles15sp3.aarch64.rpm
			x86_64	doca-host-2.7.0-204000_24.04_sles15sp3.x86_64.rpm
		SLES 15 SP4	aarch64	doca-host-2.7.0-204000_24.04_sles15sp4.aarch64.rpm
			x86_64	doca-host-2.7.0-204000_24.04_sles15sp4.x86_64.rpm
		SLES 15 SP5	aarch64	doca-host-2.7.0-204000_24.04_sles15sp5.aarch64.rpm
			x86_64	doca-host-2.7.0-204000_24.04_sles15sp5.x86_64.rpm
		SLES 15 SP6	x86_64	doca-host-2.7.0-204000_24.04_sles15sp6.x86_64.rpm

Device	Component	OS	Arch	Link
		TencentOS 3.3	arch64	doca-host-2.7.0-204000_24.04_tencentos33.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_tencentos33.x86_64.rpm
		Ubuntu 20.04	arch64	doca-host_2.7.0-204000-24.04-ubuntu2004_arm64.deb
			x86	doca-host_2.7.0-204000-24.04-ubuntu2004_amd64.deb
		Ubuntu 22.04	arch64	doca-host_2.7.0-204000-24.04-ubuntu2204_arm64.deb
			x86	doca-host_2.7.0-204000-24.04-ubuntu2204_amd64.deb
		Ubuntu 24.04	arch64	doca-host_2.7.0-204000-24.04-ubuntu2404_arm64.deb
			x86	doca-host_2.7.0-204000-24.04-ubuntu2404_amd64.deb
		UOS20.1060	arch64	doca-host-2.7.0-204000_24.04_uos201060.aarch64.rpm
			x86	doca-host-2.7.0-204000_24.04_uos201060.x86_64.rpm
		UOS20.1060A	arch64	doca-host-2.7.0-204000_24.04_uos201060a.aarch64.rpm

Device	Component	OS	Arch	Link
			x86	doca-host-2.7.0-204000_24.04_uos201060a.x86_64.rpm
		XenServer 8.2	x86	doca-host-2.7.0-204000_24.04_xenserver82.x86_64.rpm
Target BlueField Platform (Arm)	BlueField Software v 4.7.0	Ubuntu 22.04	arch64	bf-bundle-2.7.0-33_24.04_ubuntu-22.04_prod.bfb
	DOCA SDK v2.7.0	Ubuntu 22.04	arch64	doca-dpu-repo-ubuntu2204-local_2.7.0085-1.24.04.0.6.6.0.bf.4.7.0.13127_arm64.deb
	DOCA Runtime v2.7.0			

Uninstalling Software from Host

If an older DOCA (or MLNX_OFED) software version is installed on your host, make sure to uninstall it before proceeding with the installation of the new version:

Deb-based	<pre>\$ for f in \$(dpkg --list grep doca awk '{print \$2}'); do echo \$f ; apt remove --purge \$f -y ; done \$ /usr/sbin/ofed_uninstall.sh --force \$ sudo apt-get autoremove</pre>
RPM-based	<pre>host# for f in \$(rpm -qa grep -i doca) ; do yum -y remove \$f; done host# /usr/sbin/ofed_uninstall.sh --force host# yum autoremove host# yum makecache</pre>

Then perform the following steps:

1. Download NVIDIA's RPM-GPG-KEY-Mellanox-SHA256 key:

```
# wget http://www.mellanox.com/downloads/ofed/RPM-GPG-KEY-Mellanox-
SHA256
--2018-01-25 13:52:30-- http://www.mellanox.com/downloads/ofed/RPM-GPG-
KEY-Mellanox-SHA256
Resolving www.mellanox.com... 72.3.194.0
Connecting to www.mellanox.com|72.3.194.0|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1354 (1.3K) [text/plain]
Saving to: ?RPM-GPG-KEY-Mellanox-SHA256?

100%[=====>] 1,354 --.-
K/s in 0s

2018-01-25 13:52:30 (247 MB/s) - ?RPM-GPG-KEY-Mellanox-SHA256? saved
[1354/1354]
```

2. Install the key:

```
# sudo rpm --import RPM-GPG-KEY-Mellanox-SHA256
warning: rpmts_HdrFromFdno: Header V3 DSA/SHA1 Signature, key ID
6224c050: NOKEY
Retrieving key from file:///repos/MLNX_OFED//RPM-GPG-KEY-Mellanox
Importing GPG key 0x6224C050:
Userid: "Mellanox Technologies (Mellanox Technologies - Signing Key v2) "
From : /repos/MLNX_OFED//RPM-GPG-KEY-Mellanox-SHA256
Is this ok [y/N]:
```

3. Verify that the key was successfully imported:

```
# rpm -q gpg-pubkey --qf '%{NAME}-%{VERSION}-%{RELEASE}\t%
{SUMMARY}\n' | grep Mellanox
gpg-pubkey-a9e4b643-520791ba gpg(Mellanox Technologies )
```

Installing Prerequisites on Host for Target BlueField

Install RShim to manage and flash the BlueField Platform.

OS	Procedure
Deb-based	<ol style="list-style-type: none">1. Download the DOCA host repo package from the "Installation Files" section.2. Unpack the deb repo. Run:<pre>host# sudo dpkg -i <repo_file></pre>3. Perform apt update. Run:<pre>host# sudo apt-get update</pre>4. Run apt install for RShim:<pre>host# sudo apt install rshim</pre>
RPM-based	<ol style="list-style-type: none">1. Download the DOCA host repo package from the "Installation Files" section.2. Unpack the RPM repo. Run:<pre>host# sudo rpm -Uvh <repo_file></pre>3. Enable new dnf repos. Run:<pre>host# sudo dnf makecache</pre>4. Run dnf install to install RShim:<pre>host# sudo dnf install rshim</pre>

Note

Skip section "[Installing Software on Host](#)" to proceed without the DOCA local repo package for host.

Determining BlueField Device ID

It is important to learn your BlueField's device-id to perform some of the software installations or upgrades in this guide.

To determine the device ID of the BlueField Platform on your setup, run:

```
host# mst start
host# mst status -v
```

Example output:

```
MST modules:
-----
MST PCI module is not loaded
MST PCI configuration module loaded
PCI devices:
-----
DEVICE_TYPE MST PCI RDMA NET NUMA
BlueField2(rev:1) /dev/mst/mt41686_pciconf0.1 3b:00.1 mlx5_1 net-ens1f1 0

BlueField2(rev:1) /dev/mst/mt41686_pciconf0 3b:00.0 mlx5_0 net-ens1f0 0

BlueField3(rev:1) /dev/mst/mt41692_pciconf0.1 e2:00.1 mlx5_1 net-
ens7f1np1 4

BlueField3(rev:1) /dev/mst/mt41692_pciconf0 e2:00.0 mlx5_0 net-
ens7f0np0 4
```

Info

The device IDs for the BlueField-2 and BlueField-3 networking platforms in this example are `/dev/mst/mt41686_pciconf0` and

/dev/mst/mt41692_pciconf0 respectively.

Installing Software on Host

Note

Skip this section if you intend to update only the BlueField software (*.bfb).

Note


Make sure to have followed the instructions under "[Installing Prerequisites on Host for Target DPU](#)".

1. Install DOCA local repo package for host:

Info

The following table provides instructions for installing the DOCA host repo on your device depending on your OS and desired profile.

OS	Profile	Instructions
Deb-base	doca-all	1. Download the DOCA host repo from section " Installation Files " for the host.

OS	Profile	Instructions
d		<p>2. Unpack the deb repo. Run:</p> <pre data-bbox="540 260 1463 352">host# dpkg -i <repo_file></pre> <p>3. Perform apt update. Run:</p> <pre data-bbox="540 396 1463 489">host# apt-get update</pre> <p>4. If the kernel version on your host is not supported (not shown under "Supported Operating System Distributions"), refer to section "DOCA Extra Package".</p> <p>5. Ensure that the kernel headers installed match the version of the currently running kernel.</p> <div data-bbox="540 768 1463 1041" style="background-color: #ffffcc; padding: 10px; margin: 10px 0;"> <p> Info If the build directory exists in under <code>/lib/modules/\$(uname -r)/build</code>, then the kernel headers are installed.</p> </div> <p>6. Run apt install for DOCA SDK and DOCA runtime:</p> <pre data-bbox="540 1146 1463 1239">host# sudo apt install -y doca-all mlnx-fw-updater</pre>
	doca-networking	<p>1. Download the DOCA host repo from section "Installation Files" for the host.</p> <p>2. Unpack the deb repo. Run:</p> <pre data-bbox="540 1388 1463 1480">host# dpkg -i <repo_file></pre> <p>3. Perform apt update. Run:</p> <pre data-bbox="540 1524 1463 1617">host# apt-get update</pre> <p>4. If the kernel version on your host is not supported (not shown under "Supported Operating System Distributions"), refer to section "DOCA Extra Package".</p> <p>5. Ensure that the kernel headers installed match the version of the currently running kernel.</p>

OS	Profile	Instructions
		<p>i Info If the build directory exists in under <code>/lib/modules/\$(uname -r)/build</code>, then the kernel headers are installed.</p> <p>6. Run apt install for DOCA SDK and DOCA runtime:</p> <pre>host# sudo apt install -y doca-networking mlnx-fw-updater</pre>
	doca- ofed	<ol style="list-style-type: none"> 1. Download the DOCA host repo from section "Installation Files" for the host. 2. Unpack the deb repo. Run: <pre>host# sudo dpkg -i <repo_file></pre> 3. Perform apt update. Run: <pre>host# sudo apt-get update</pre> 4. If the kernel version on your host is not supported (not shown under "Supported Operating System Distributions"), refer to section "DOCA Extra Package". 5. Ensure that the kernel headers installed match the version of the currently running kernel. <p>i Info If the build directory exists in under <code>/lib/modules/\$(uname -r)/build</code>, then the kernel headers are installed.</p> <p>6. Install doca-ofed. Run:</p> <pre>host# sudo apt install -y doca-ofed mlnx-fw-updater</pre>
RPM -	doca- all	<ol style="list-style-type: none"> 1. Download the DOCA host repo from section "Installation Files" for the host .

OS	Profile	Instructions
base d		<p>2. Unpack the rpm repo. Run:</p> <pre>host# rpm -Uvh <repo_file>.rpm</pre> <p>3. Perform yum update. Run:</p> <pre>host# sudo yum makecache</pre> <p>4. If the kernel version on your host is not supported (not shown under "Supported Operating System Distributions"), refer to section "DOCA Extra Package".</p> <p>5. Run yum install for DOCA SDK and DOCA runtime:</p> <pre>host# sudo yum install -y doca-all mlnx-fw-updater</pre>
	doca- netwo rking	<p>1. Download the DOCA host repo from section "Installation Files" for the host .</p> <p>2. Unpack the rpm repo. Run:</p> <pre>host# rpm -Uvh <repo_file>.rpm</pre> <p>3. Perform yum update. Run:</p> <pre>host# sudo yum makecache</pre> <p>4. If the kernel version on your host is not supported (not shown under "Supported Operating System Distributions"), refer to section "DOCA Extra Package".</p> <p>5. Run yum install for DOCA SDK and DOCA runtime:</p> <pre>host# sudo yum install -y doca-networking mlnx-fw-updater</pre>
	doca- ofed	<p>1. Download the DOCA host repo from section "Installation Files" for the host.</p> <p>2. Unpack the RPM repo. Run:</p> <pre>host# sudo rpm -Uvh <repo_file>.rpm</pre> <p>3. Perform yum update. Run:</p> <pre>host# sudo yum makecache</pre> <p>4. If the kernel version on your host is not supported (not shown under "Supported Operating System Distributions"), refer to</p>

OS	Profile	Instructions
		section " DOCA Extra Package ". 5. Install doca-ofed. Run: <pre>host# sudo yum install -y doca-ofed mlnx-fw-updater</pre>

2. Load the drivers:

```
host# sudo /etc/init.d/openibd restart
```

3. Initialize MST. Run:

```
host# sudo mst restart
```

4. Skip this step if your BlueField Platform is Ethernet only. Please refer to [Supported Platforms](#) to learn your Bluefield type.

If you have a VPI-capable BlueField, the default link type of the ports will be configured to IB. To verify your link type, run:

```
host# sudo mst start
host# sudo mlxconfig -d <device-id> -e q | grep -i link_type
Configurations: Default Current Next Boot
* LINK_TYPE_P1 IB(1) ETH(2) IB(1)
* LINK_TYPE_P2 IB(1) ETH(2) IB(1)
```

Note

If your BlueField is Ethernet capable only, then the `sudo mlxconfig -d <device>` command will not provide an output.

If the current link type is set to IB, run the following command to change it to Ethernet:

```
host# sudo mlxconfig -d <device-id> s LINK_TYPE_P1=2 LINK_TYPE_P2=2
```

5. Verify that RShim is active.

```
host# sudo systemctl status rshim
```

This command is expected to display `active (running)`. If RShim service does not launch automatically, run:

```
host# sudo systemctl enable rshim  
host# sudo systemctl start rshim
```

6. Assign a dynamic IP to `tmfifo_net0` interface (RShim host interface).

Note

Skip this step if you are [installing the DOCA image on multiple DPUs](#).

```
host# ifconfig tmfifo_net0 192.168.100.1 netmask 255.255.255.252 up
```

DOCA Extra Package

If the kernel version on your host is not supported (not shown under "[Supported Operating System Distributions](#)"), two options are available:

- Switch to a compatible kernel.
- Install `doca-extra` package:

1. Run:

```
host# sudo apt/yum install -y doca-extra
```

2. Execute the `doca-kernel-support` script which rebuilds and installs the DOCA-Host kernel modules with the running kernel:

```
host# sudo /opt/mellanox/doca/tools/doca-kernel-support
```

3. Install user-space packages:

```
host# sudo apt/yum install -y doca-ofed-userspace
```

Note

`doca-kernel-support` does not support customized or unofficial kernels.

Installing Software on DPU

Users have two options for installing DOCA on BlueField DPU or SuperNIC:

- Upgrading the full DOCA image on BlueField (recommended) – this option overwrites the entire boot partition with an Ubuntu 22.04 installation and updates BlueField and NIC firmware.
- Upgrading DOCA local repo package on BlueField – this option upgrades DOCA components without overwriting the boot partition. Use this option to preserve configurations or files on BlueField itself.

Installing Full DOCA Image on DPU via Host

Warning

This step overwrites the entire boot partition.

Note

This installation sets up the OVS bridge.

Note

If you are installing DOCA on multiple BlueField platforms, skip to section [Installing Full DOCA Image on Multiple BlueField Platforms](#).

Option 1 – No Pre-defined Password

Note

To change the default Ubuntu password during the BFB bundle installation, proceed to Option 2.

BFB installation is executed as follows:

```
host# sudo bfb-install --rshim rshim<N> --bfb <image_path.bfb>
```

Where `rshim<N>` is `rshim0` if you only have one Bluefield. You may run the following command to verify:

```
host# ls -la /dev/ | grep rshim
```

Option 2 – Set Pre-defined Password

Ubuntu users can provide a unique password that will be applied at the end of the BlueField BFB bundle installation. This password needs to be defined in a `bf.cfg` configuration file.

To set the password for the "ubuntu" user:

1. Create password hash. Run:

```
host# openssl passwd -1  
Password:  
Verifying - Password:  
$1$3B0RlrfX$TIHry93NFUJzg3Nya00rE1
```

2. Add the password hash in quotes to the `bf.cfg` file:

```
host# sudo vim bf.cfg  
ubuntu_PASSWORD='$1$3B0RlrfX$TIHry93NFUJzg3Nya00rE1'
```

When running the installation command, use the `--config` flag to provide the file containing the password:

```
host# sudo bfb-install --rshim rshim<N> --bfb <image_path.bfb> --config bf.cfg
```

Note

Optionally, to upgrade the BlueField integrated BMC firmware using BFB bundle, please provide the current BMC root credentials in a `bf.cfg` file, as shown in the following:


```
BMC_PASSWORD="<root password>"
BMC_USER="root"
BMC_REBOOT="yes"
```

Unless previously changed, the default BMC root password is OpenBmc.

Note

If `--config` is not used, then upon first login to the BlueField device, users will be prompted to update the default 'ubuntu' password.

The following is an example of Ubuntu-22.04 BFB bundle installation (Release version may vary in the future).

```
host# sudo bfb-install --rshim rshim0 --bfb bf-bundle-2.7.0_24.04_ubuntu-
22.04_prod.bfb --config bf.cfg
Pushing bfb 1.41GiB 0:02:02 [11.7MiB/s] [ <=> ]
Collecting BlueField booting status. Press Ctrl+C to stop
INFO[PSC]: PSC BL1 START
INFO[BL2]: start
INFO[BL2]: boot mode (rshim)
INFO[BL2]: VDDQ: 1120 mV
INFO[BL2]: DDR POST passed
INFO[BL2]: UEFI loaded
INFO[BL31]: start
INFO[BL31]: lifecycle GA Secured
INFO[BL31]: VDD: 850 mV
INFO[BL31]: runtime
INFO[BL31]: MB ping success
INFO[UEFI]: eMMC init
```

```
INFO[UEFI]: eMMC probed
INFO[UEFI]: UPVS valid
INFO[UEFI]: PMI: updates started
INFO[UEFI]: PMI: total updates: 1
INFO[UEFI]: PMI: updates completed, status 0
INFO[UEFI]: PCIe enum start
INFO[UEFI]: PCIe enum end
INFO[UEFI]: UEFI Secure Boot
INFO[UEFI]: PK configured
INFO[UEFI]: Redfish enabled
INFO[UEFI]: exit Boot Service
INFO[MISC]: Found bf.cfg
INFO[MISC]: Ubuntu installation started
INFO[MISC]: Installing OS image
INFO[MISC]: Changing the default password for user ubuntu
INFO[MISC]: Ubuntu installation completed
INFO[MISC]: Updating NIC firmware...
INFO[MISC]: NIC firmware update done
INFO[MISC]: Installation finished
```

To verify the BlueField has completed booting up, allow additional 90 seconds then perform the following:

```
host# sudo cat /dev/rshim<N>/misc
...
INFO[MISC]: Linux up
INFO[MISC]: DPU is ready
```

Installing Full DOCA Image on Multiple BlueField Platforms

On a host with multiple BlueField devices, the BFB image can be installed on all of them using the multi-bfb-install [script](#).

```
host# ./multi-bfb-install --bfb <image_path.bfb> --password <password>
```

This script detects the number of RShim devices and configures them statically.

- For Ubuntu – the script creates a configuration file `/etc/netplan/20-tmfifo.yaml`
- For CentOS/RH 8.0 and 8.2 – the script installs the `bridge-utils` package to use the `brctl` command, creates the `tm-br` bridge, and connects all RShim interfaces to it

After the installation is complete, the configuration of the bridge and each RShim interface can be observed using `ifconfig`. The expected result is to see the IP on the `tm-br` bridge configured to `192.168.100.1` with subnet `255.255.255.0`.

Note

To log into BlueField with `rshim0`, run:

```
ssh ubuntu@192.168.100.2
```

For each RShim after that, add 1 to the fourth octet of the IP address (e.g., `ubuntu@192.168.100.3` for `rshim1`, `ubuntu@192.168.100.4` for `rshim2`, etc).

The script burns a new MAC address to each BlueField and configures a new IP, `192.168.100.x`, as described earlier.

Installing DOCA Local Repo Package on BlueField

Note

If you have already [installed BlueField image](#), be aware that the DOCA SDK, Runtime, and Tools are already contained in the BFB, and this

installation is not mandatory. If you have not installed the BlueField image and wish to update DOCA Local Repo package, proceed with the following procedure.

Note

Before installing DOCA on the target BlueField, make sure the out-of-band interface (mgmt) is connected to the internet.

1. Download the DOCA SDK and DOCA Runtime package from section [Installation Files](#).
2. Copy deb repo package into BlueField. Run:

```
host# sudo scp -r doca-repo-aarch64-ubuntu2204-local_<version>_arm64.deb  
ubuntu@192.168.100.2:/tmp/
```

3. Unpack the deb repo. Run:

```
dpu# sudo dpkg -i doca-dpu-repo-ubuntu2204-local_<version>_arm64.deb
```

4. Run apt update.

```
dpu# sudo apt-get update
```

5. Run apt install for DOCA Runtime and DOCA SDK:

```
dpu# sudo apt install doca-runtime doca-sdk
```

Upgrading Firmware

Note

This operation is only required if the user skipped NIC firmware update during BFB bundle installation using the parameter `WITH_NIC_FW_UPDATE=no` in the `bf.cfg` file.

This section explains how to update the NIC firmware on a DOCA installed BlueField OS.

Note

If multiple BlueFields are installed, the following steps must be performed on all of them after [BFB installation](#).

An up-to-date NIC firmware image is provided in BlueField BFB bundle and copied to the BlueField filesystem during BFB installation.

To upgrade firmware in the BlueField Arm OS:

1. SSH to your BlueField Arm OS by any means available.

The following instructions enable to login to the BlueField Arm OS from the host OS over the RShim virtual interface, `tmfifo_net<N>` and do not require LAN connectivity with the BlueField OOB network port.

Note

This operation can be performed over the host's `tmfifo_net0` IPv4, 192.168.100.1 (preconfigured) with BlueField Arm OS at 192.168.100.2 (default).

If multiple BlueField DPUs were updated using the multi-bfb-install script, as explained above, then each target BlueField OS IPv4 address changes in its last octate according to the underlying RShim interface number: 192.168.100.3 for rshim1, 192.168.100.4 for rshim2, etc.

The default credentials for Ubuntu are as follows:

Username	Password
ubuntu	ubuntu

For example, to log into BlueField Arm OS over IPv6:

```
host]# systemctl restart rshim
// Wait 10 seconds

host]# ssh -6 fe80::21a:caff:feff:ff01%tmfifo_net<N>
Password: <configured-password>
```

2. Upgrade firmware in BlueField. Run:

```
dpu# sudo /opt/mellanox/mlnx-fw-updater/mlnx_fw_updater.pl --force-fw-update
```

Example output:

```
Device #1:
-----

Device Type: BlueField-2
[...]
Versions: Current Available
FW <Old_FW> <New_FW>
```

3. For the firmware upgrade to take effect perform a [BlueField system reboot](#).

Post-installation Procedure

1. Restart the driver. Run:

```
host# sudo /etc/init.d/openibd restart
Unloading HCA driver: [ OK ]
Loading HCA driver and Access Layer: [ OK ]
```

2. Configure the physical function (PF) interfaces.

```
host# sudo ifconfig <interface-1> <network-1/mask> up
host# sudo ifconfig <interface-2> <network-2/mask> up
```

For example:

```
host# sudo ifconfig p2p1 192.168.200.32/24 up
host# sudo ifconfig p2p2 192.168.201.32/24 up
```

Pings between the source and destination should now be operational.

Upgrading BlueField Using Standard Linux Tools

This dpu-upgrade procedure enables upgrading DOCA components using standard Linux tools (e.g., apt update and yum update). This process utilizes native package manager repositories to upgrade DPUs without the need for a full installation, and has the following benefits :

- Only updates components that include modifications
 - Configurable – user can select specific components (e.g., UEFI-ATF, NIC-FW)

- Includes upgrade of:
 - DOCA drivers and libraries
 - DOCA reference applications
 - BSP (UEFI/ATF) upgrade while maintaining the configuration
 - NIC firmware upgrade while maintaining the configuration
- Does not:
 - Impact user binaries
 - Upgrade non-Ubuntu OS kernels
 - Upgrade DPU BMC firmware
- After completion of DPU upgrade:
 - If NIC firmware was not updated, perform DPU Arm reset (software reset / reboot DPU)
 - If NIC firmware was updated, perform firmware reset (mlxfwreset) or perform a graceful shutdown and power cycle

OS	Action	Instructions
Ubuntu/ Debian	Remove mlxbf-bootimages package	<pre><dpu> \$ apt remove --purge mlxbf-bootimages* -y</pre>
	Install the the GPG key	<pre><dpu> \$ apt update <dpu> \$ apt install gnupg2</pre>
	Export the desired distribution	Export DOCA_REPO with the relevant URL. The following is an example for Ubuntu 22.04: <pre><dpu> \$ export DOCA_REPO="https://linux.mellanox.com/public/repo/doca/2.7.0/ubuntu22.04/dpu-arm64"</pre>

OS	Action	Instructions
		<ul style="list-style-type: none"> • Ubuntu 22.04 – https://linux.mellanox.com/public/repo/doca/2.7.0/ubuntu22.04/dpu-arm64 • Ubuntu 20.04 – https://linux.mellanox.com/public/repo/doca/2.7.0/ubuntu20.04/dpu-arm64 • Debian 12 – https://linux.mellanox.com/public/repo/doca/2.7.0/debian12/dpu-arm64
	Add GPG key to APT trusted keyring	<pre data-bbox="475 695 1461 846"><dpu> \$ curl \$DOCA_REPO/GPG-KEY-Mellanox.pub gpg --dearmor > /etc/apt/trusted.gpg.d/GPG-KEY-Mellanox.pub</pre>
	Add DOCA online repository	<pre data-bbox="475 894 1461 1066"><dpu> \$ echo "deb [signed-by=/etc/apt/trusted.gpg.d/GPG-KEY-Mellanox.pub] \$DOCA_REPO ." > /etc/apt/sources.list.d/doca.list</pre>
	Update index	<pre data-bbox="475 1108 1461 1178"><dpu> \$ apt update</pre>
	Upgrade UEFI/ATF firmware	<p data-bbox="475 1220 545 1251">Run:</p> <pre data-bbox="475 1262 1461 1331"><dpu> \$ apt install mlxbf-bootimages-signed</pre> <p data-bbox="475 1352 1146 1383">Then initiate upgrade for UEFI/ATF firmware:</p> <pre data-bbox="475 1394 1461 1520"><dpu> \$ apt install mlxbf-scripts <dpu> \$ bfrec</pre>
	Upgrade BlueField DPU NIC firmware	<p data-bbox="475 1566 545 1598">Run:</p> <pre data-bbox="475 1608 1461 1698"><dpu> \$ apt install mlx-fw-updater-signed.aarch64</pre> <div data-bbox="475 1755 1461 1942" style="background-color: #ffffcc; padding: 10px;"> <p data-bbox="505 1797 672 1839">i Note</p> <p data-bbox="578 1850 1271 1881">This immediately starts NIC firmware upgrade.</p> </div>

OS	Action	Instructions
		<p>To prevent automatic upgrade, run:</p> <pre><dpu> \$ export RUN_FW_UPDATER=no</pre>
	Remove old metapackages	<pre><dpu> \$ apt-get remove doca-tools doca-sdk doca-runtime -y</pre>
	Install new metapackages	<pre><dpu> \$ apt-get install doca-runtime doca-devel -y</pre>
	Upgrade system	<pre><dpu> \$ apt upgrade</pre>
	Apply the new changes, NIC firmware, and UEFI/ATF	<p>For the upgrade to take effect, perform BlueField system reboot as explained in the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page.</p> <div style="background-color: #ffffcc; padding: 10px;"> <p>Note This step triggers immediate reboot of the BlueField Arm cores.</p> </div>
Cent OS/RHEL/Anolis/Rocky	Remove mlxbf-bootimages package	<pre><dpu> \$ yum -y remove mlxbf-bootimages* <dpu> \$ yum makecache</pre>
	Export the desired distribution	<p>Export DOCA_REPO with the relevant URL. The following is an example for Rocky Linux 8.6:</p> <pre><dpu> \$ export DOCA_REPO="https://linux.mellanox.com/public/repo/doca/2.7.0/rhel8.6/dpu-arm64/"</pre> <ul style="list-style-type: none"> AnolisOS 8.6 – https://linux.mellanox.com/public/repo/doca/2.7.0/anolis8.6/

OS	Action	Instructions
		<p>dpu-arm64/</p> <ul style="list-style-type: none"> • OpenEuler 20.03 sp1 – https://linux.mellanox.com/public/repo/doca/2.7.0/openeuler20.03sp1/dpu-arm64/ • CentOS 7.6 with 4.19 kernel – https://linux.mellanox.com/public/repo/doca/2.7.0/rhel7.6-4.19/dpu-arm64/ • CentOS 7.6 with 5.10 kernel – https://linux.mellanox.com/public/repo/doca/2.7.0/rhel7.6-5.10/dpu-arm64/ • CentOS 7.6 with 5.4 kernel – https://linux.mellanox.com/public/repo/doca/2.7.0/rhel7.6/dpu-arm64/ • Rocky Linux 8.6 – https://linux.mellanox.com/public/repo/doca/2.7.0/rhel8.6/dpu-arm64/
	Add DOCA online repository	<pre>echo "[doca] name=DOCA Online Repo baseurl=\$DOCA_REPO enabled=1 gpgcheck=0 priority=10 cost=10" > /etc/yum.repos.d/doca.repo</pre> <p>A file is created under <code>/etc/yum.repos.d/doca.repo</code> .</p>
	Update index	<pre><dpu> \$ yum makecache</pre>
	Upgrade UEFI/ATF firmware	<p>Run:</p> <pre><dpu> \$ yum install mlxbf-bootimages-signed.aarch64 mlxbf-bfscripts</pre> <p>Then i nitiate the upgrade for UEFI/ATF firmware:</p> <pre><dpu> \$ bfrec</pre>

OS	Action	Instructions
	Upgrade BlueField DPU NIC firmware	<p>The following command updates the firmware package and automatically attempts to flash the firmware to the NIC:</p> <pre data-bbox="477 310 1461 407"><dpu> \$ yum install mlnx-fw-updater-signed.aarch64</pre> <p>i Info This step can be used as a standalone firmware update. In any case, it is performed as part of the upgrade flow.</p> <p>i Note To prevent automatic flashing of the firmware to the NIC, run the following first:</p> <pre data-bbox="581 974 1273 1071"><dpu> \$ export RUN_FW_UPDATER=no 0000018f-9ce9-dc04-a5ff-9ffdda38000a</pre>
	Remove old metapackages	<pre data-bbox="477 1262 1461 1358"><dpu> \$ yum -y remove doca-tools doca-sdk doca-runtime</pre>
	Install new metapackages	<pre data-bbox="477 1411 1461 1507"><dpu> \$ yum -y install doca-runtime doca-devel</pre>
	Upgrade system	<pre data-bbox="477 1566 1461 1642"><dpu> \$ yum upgrade --nobest</pre>
	Apply the new changes, NIC firmware,	<p>For the upgrade to take effect, perform BlueField system reboot as explained in the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page.</p> <p>i Note</p>

OS	Action	Instructions
	and UEFI/ATF	This step triggers immediate reboot of the BlueField Arm cores.

Building Your Own BFB Installation Image

Users wishing to build their own customized BlueField OS image can use the BFB build environment. Please refer to the bfb-build project in [this GitHub webpage](#) for more information.

Note

For a customized BlueField OS image to boot on the UEFI secure-boot-enabled BlueField (default BlueField secure boot setting), the OS must be either signed with an existing key in the UEFI DB (e.g., the Microsoft key), or UEFI secure boot must be disabled. Please refer to the "Secure Boot" page under [NVIDIA BlueField DPU Platform Operating System Documentation](#) for more details.

Setting Up Build Environment for Developers

For full instructions about setting up a development environment, refer to the [NVIDIA DOCA Developer Guide](#).

Additional SDKs for DOCA

Installing CUDA on NVIDIA Converged Accelerator

NVIDIA® CUDA® is a parallel computing platform and programming model developed by NVIDIA for general computing GPUs.

This section details the necessary steps to set up CUDA on your environment. This section assumes that a BFB image has already been installed on your environment. To install CUDA on your converged accelerator:

1. Download and install the latest NVIDIA Data Center GPU driver.
2. Download and install CUDA

Note

The CUDA version tested to work with DOCA SDK is 11.8.0.

Note

Downloading CUDA includes the latest NVIDIA Data Center GPU driver and CUDA toolkit. For more information about CUDA and driver compatibility, refer to the [NVIDIA CUDA Toolkit Release Notes](#).

Configuring Operation Mode

There are two modes that the NVIDIA Converged Accelerator may operate in:

- Standard mode (default) – the BlueField and the GPU operate separately
- BlueField-X mode – the GPU is exposed to BlueField and is no longer visible on the host

To verify which mode the system is operating in, run:

```
host# sudo mst start  
host# sudo mlxconfig -d <device-id> q PCI_DOWNSTREAM_PORT_OWNER[4]
```

Note

To learn your BlueField Platform's device ID, refer to section "[Determining BlueField Device ID](#)".

- Standard mode output:

```
Device #1:  
[...]  
Configurations: Next Boot  
PCI_DOWNSTREAM_PORT_OWNER[4] DEVICE_DEFAULT(0)
```

- BlueField-X mode output:

```
Device #1:  
[...]  
Configurations: Next Boot  
PCI_DOWNSTREAM_PORT_OWNER[4] EMBEDDED_CPU(15)
```

To configure BlueField-X mode, run:

```
host# mlxconfig -d <device-id> s PCI_DOWNSTREAM_PORT_OWNER[4]=0xF
```

To configure standard mode, run:

```
host# mlxconfig -d <device-id> s PCI_DOWNSTREAM_PORT_OWNER[4]=0x0
```

Note

To learn your BlueField Platform's device ID, refer to section "[Determining BlueField Device ID](#)".

Power cycle is required for configuration to take effect. For power cycle the host run:

```
host# ipmitool power cycle
```

Downloading and Installing CUDA Toolkit and Driver

This section details the necessary steps to set up CUDA on your environment. It assumes that a BFB image has already been installed on your environment.

1. Install CUDA by visiting the [CUDA Toolkit Downloads](#) webpage.

Note

Select the Linux distribution and version relevant for your environment.

Note

This section shows the native compilation option either on x86 or aarch64 hosts.

2. Test that the driver installation completed successfully. Run:

```
dpu# nvidia-smi
```


Tue Apr 5 13:37:59 2022

```
+-----+
| NVIDIA-SMI 510.47.03 Driver Version: 510.47.03 CUDA Version: 11.8 |
|-----+-----+-----+
| GPU Name Persistence-M | Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap | Memory-Usage | GPU-Util Compute M. |
| | | MIG M. |
|=====+=====+=====+
| 0 NVIDIA BF A10 Off | 00000000:06:00.0 Off | 0 |
| 0% 43C P0 N/A / 225W | 0MiB / 23028MiB | 0% Default |
| | | N/A |
+-----+-----+-----+
+-----+
| Processes: |
| GPU GI CI PID Type Process name GPU Memory |
| ID ID Usage |
|=====+=====+=====+
| No running processes found |
+-----+
```


3. Verify that the installation completed successfully.

1. Download CUDA samples repo. Run:

```
dpu# git clone https://github.com/NVIDIA/cuda-samples.git
```

2. Build and run vectorAdd CUDA sample. Run:

```
dpu# cd cuda-samples/Samples/0_Introduction/vectorAdd
dpu# make
dpu# ./vectorAdd
```

 **Note**

If the `vectorAdd` sample works as expected, it should output "Test Passed".

Note

If it seems that the GPU is slow or stuck, stop execution and run:

```
dpu# sudo setpci -v -d ::0302 800.L=201 # CPL_VCO = 32
```

GPUDirect RDMA

For information on GPUDirect RDMA and more, refer to [DOCA GPUNetIO](#) documentation.

Installing Rivermax on BlueField

NVIDIA Rivermax offers a unique IP-based solution for any media and data streaming use case.

This section provides the steps to install Rivermax assuming that a BFB image has already been installed on your environment.

Downloading Rivermax Driver

1. Navigate to the [NVIDIA Rivermax SDK](#) product page.
2. Register to be able to download the driver package using the JOIN button at the top of the page.
3. Download the appropriate driver package according to your BFB under the "Linux" subsection. For example, for Ubuntu 22.04 BFB, download `rivermax_ubuntu2204_<version>.tar.gz`.

Installing Rivermax Driver

1. Copy the .tgz file to BlueField:

```
host# sudo scp -r rivermax_ubuntu2204_<version>.tar.gz
ubuntu@192.168.100.2:/tmp/
```

2. Extract the Rivermax file:

```
dpu# sudo tar xzf rivermax_ubuntu2204_<version>.tar.gz
```

3. Install the Rivermax driver package:

```
dpu# cd <rivermax-version>/Ubuntu.22.04/deb-dist/aarch64/
dpu# sudo dpkg -i rivermax_<version>.deb
```

Installing Rivermax Libraries from DOCA

Rivermax libraries are compatibles with DOCA components and can be found inside the doca-dpu-repo.

1. Unpack the doca-dpu-repo:

```
dpu# sudo dpkg -i doca-dpu-repo-ubuntu2204-local_<version>_arm64.deb
```

2. Run apt update:

```
dpu# sudo apt-get update
```

3. Install the Rivermax libraries:

```
dpu# sudo apt install doca-rmax-libs
```

```
dpu# sudo apt install libdoca-rmax-libs-dev
```

For additional details and guidelines, please visit the [NVIDIA Rivermax SDK](#) product page.

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

NVIDIA DOCA Developer Guide

This guide details the recommended steps to set up an NVIDIA DOCA development environment.

Introduction

This guide is intended for software developers aiming to modify existing NVIDIA DOCA applications or develop their own DOCA-based software.

Instructions for installing DOCA on the NVIDIA® BlueField® Networking Platform (i.e., DPU or SuperNIC) can be found in the [NVIDIA DOCA Installation Guide for Linux](#).

This guide focuses on the recommended flow for developing DOCA-based software, and will address the following scenarios:

- BlueField is accessible and can be used during the development and testing process
 - [Working within a development container](#)
- BlueField is inaccessible, and the development happens on the host or on a different server

- [Cross-compilation from the host](#)
- [Working within a development container on top of QEMU running on the host](#)

It is recommended to follow the instructions for the first scenario, leveraging BlueField during the development and testing process.

This guide recommends using DOCA's development container during the development process on BlueField Platforms or on the host. Deploying development containers allows multiple developers to work simultaneously on the same device (host or BlueField Platform) in an isolated manner and even across multiple different DOCA SDK versions. This can allow multiple developers to work on the BlueField Platform itself, for example, without needing to have a dedicated BlueField per developer.

Another benefit of this container-based approach is that the development container allows developers to create and test their DOCA-based software in a user-friendly environment that comes pre-shipped with a set of handy development tools. The development container is focused on improving the development experience and is designed for that purpose, whereas the BlueField software is meant to be an efficient runtime environment for DOCA products.

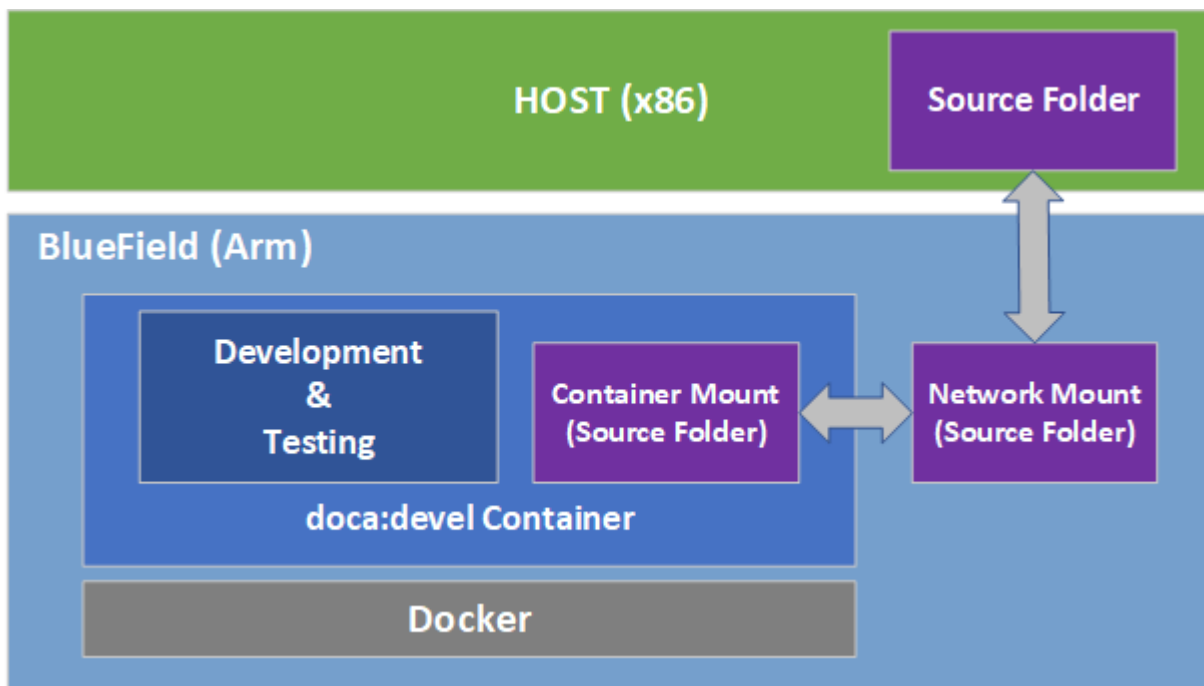
Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

Developing Using BlueField Networking Platform

Setup

DOCA's base image containers include a DOCA development container for the BlueField (doca:devel) which can be found on [NGC](#). It is recommended to deploy this container on top of BlueField when preparing a development setup.



The recommended approach for working using DOCA's development container on top of the BlueField, is by using [docker](#), which is already included in the supplied BFB image.

1. Make sure the docker service is started. Run:

```
sudo systemctl daemon-reload
sudo systemctl start docker
```

2. Pull the container image:

1. Visit the [NGC page](#) of the DOCA base image.
2. Under the "Tags" menu, select the desired development tag for BlueField.
3. The container tag for the docker pull command is copied to your clipboard once selected. Example docker pull command using the selected tag:

```
sudo docker pull nvcr.io/nvidia/doca/doca:1.5.1-devel
```

3. Once loaded locally, you may find the image's ID using the following command:

```
sudo docker images
```

Example output:

```
REPOSITORY TAG IMAGE ID CREATED SIZE
nvcv.io/nvidia/doca/doca 1.5.1-devel 931bd576eb49 10 months ago 1.49GB
```

4. Run the docker image:

```
sudo docker run -v <source-code-folder>:/doca_devel -v
/dev/hugepages:/dev/hugepages --privileged --net=host -it <image-name/ID>
```

For example, to map a source folder named `my_sources` into the same container tag from the example above, the command should look like this:

```
sudo docker run -v my_sources:/doca_devel -v
/dev/hugepages:/dev/hugepages --privileged --net=host -it
nvcv.io/nvidia/doca/doca:1.5.1-devel
```

After running the command, you get a shell inside the container where you can build your project using the regular build commands:

- From the container's perspective, the mounted folder will be named `/doca_devel`

Note

Make sure to map a folder with write privileges to everyone. Otherwise, the docker would not be able to write the output files to it.

- `--net=host` ensures the container has network access, including visibility to SFs and VFs as allocated on BlueField
- `-v /dev/hugepages:/dev/hugepages` ensures that allocated huge pages are accessible to the container

Development

It is recommended to do the development within the `doca:devel` container. That said, some developers prefer different integrated development environments (IDEs) or development tools, and sometimes prefer working using a graphical IDE until it is time to compile the code. As such, the recommendation is to mount a network share to BlueField (refer to [NVIDIA DOCA DPU CLI](#) for more information) and to the container.

Note

Having the same code folder accessible from the IDE and the container helps prevent edge cases where the compilation fails due to a typo in the code, but the typo is only fixed locally within the container and not propagated to the main source folder.

Testing

The container is marked as "privileged", hence it can directly access the hardware capabilities of the BlueField Platform. This means that once the tested program compiles successfully, it can be directly tested from within the container without the need to copy it to BlueField and running it there.

Publishing

Once the program passes the testing phase, it should be prepared for deployment. While some proof-of-concept (POC) programs are just copied "as-is" in their binary form, most deployments will probably be in the form of a package (`.deb/.rpm`) or a container.

Construction of the binary package can be done as-is inside the current `doca:devel` container, or as part of a CI pipeline that will leverage the same development container

as part of it.

For the construction of a container to ship the developed software, it is recommended to use a [multi-staged build](#) that ships the software on top of the runtime-oriented DOCA base images:

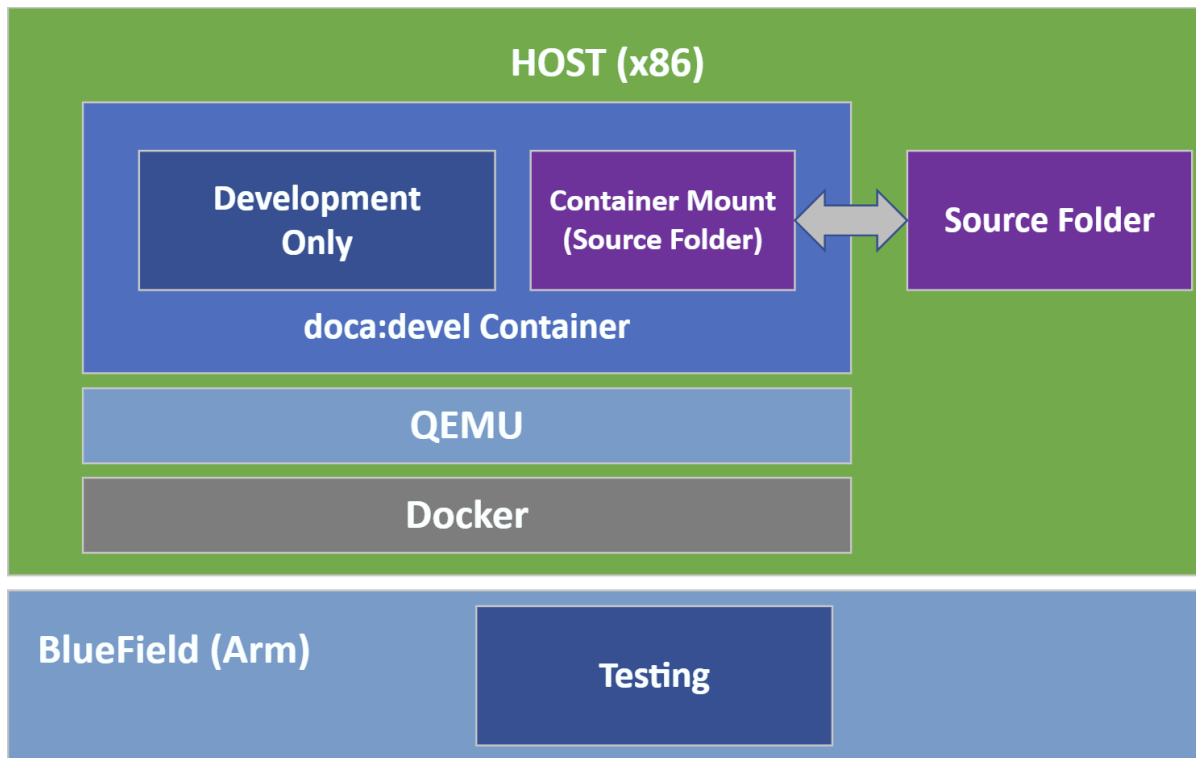
- `doca:base-rt` – slim DOCA runtime environment
- `doca:full-rt` – full DOCA runtime environment similar to the BlueField image

The runtime DOCA base images, alongside more details about their structure, can be found under the same [NGC page](#) that hosts the `doca:devel` image.

For a multi-staged build, it is recommended to compile the software inside the `doca:devel` container, and later copy it to one of the runtime container images. All relevant images must be pulled directly from NGC (using `docker pull`) to the container registry of BlueField.

Developing Without BlueField Networking Platform

If the development process needs to be done without access to a BlueField Platform, the recommendation is to use a QEMU-based deployment of a container on top of a regular x86 server. The development container for the host will be the same `doca:devel` image we mentioned previously.



Setup

1. Make sure Docker is installed on your host. Run:

```
docker version
```

If it is not installed, visit the official [Install Docker Engine](#) webpage for installation instructions.

2. Install QEMU on the host.

Note

This step is for x86 hosts only. If you are working on an aarch64 host, move to the next step.

Host OS	Command
Ubuntu	<pre>sudo apt-get install qemu binfmt-support qemu-user-static sudo docker run --rm --privileged multiarch/qemu-user-static - -reset -p yes</pre>
CentOS/RHEL 7.x	<pre>sudo yum install epel-release sudo yum install qemu-system-arm</pre>
CentOS 8.0/8.2	<pre>sudo yum install epel-release sudo yum install qemu-kvm</pre>
Fedora	<pre>sudo yum install qemu-system-aarch64</pre>

3. If you are using CentOS or Fedora on the host, verify if `qemu-aarch64.conf` Run:

```
cat /etc/binfmt.d/qemu-aarch64.conf
```

If it is missing, run:

```
echo ":qemu-
aarch64:M::\x7fELF\x02\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00
aarch64-static:" > /etc/binfmt.d/qemu-aarch64.conf
```

4. If you are using CentOS or Fedora on the host, restart system binfmt. Run:

```
$ sudo systemctl restart systemd-binfmt
```

5. To load and execute the development container, refer to the "[Setup](#)" section discussing the same docker-based deployment on the BlueField side.

Note

The `doca:devel` container supports multiple architectures. Therefore, Docker by default attempts to pull the one matching that of the current machine (i.e., `amd64` for the host and `arm64` for BlueField). Pulling the `arm64` container from the `x86` host can be done by adding the flag `--platform=linux/arm64`:

```
sudo docker pull --platform=linux/arm64
nvr.io/nvidia/doca/doca:1.5.1-devel
```

Development

Much like the development phase [using a BlueField DPU](#), it is recommended to develop within the container running on top of QEMU.

Testing

While the compilation can be performed on top of the container, testing the compiled software must be done on top of a BlueField Platform. This is because the QEMU environment emulates an `aarch64` architecture, but it does not emulate the hardware devices present on the BlueField Platform. Therefore, the tested program will not be able to access the devices needed for its successful execution, thus mandating that the testing is done on top of a physical BlueField.

Note

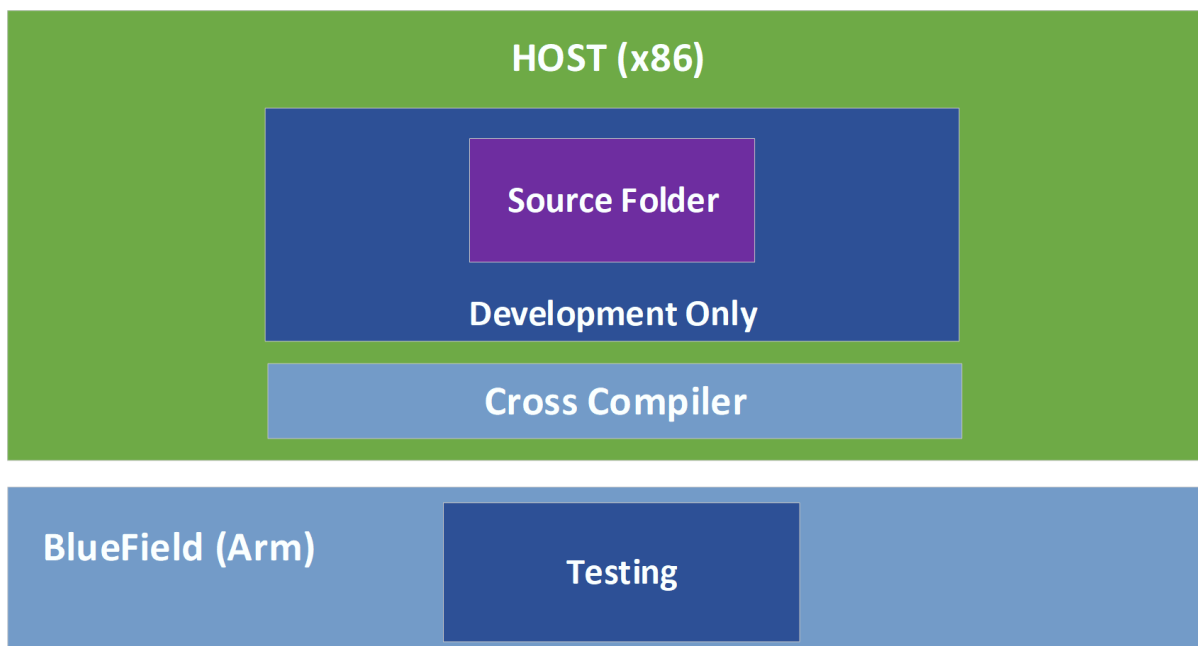
Make sure that the DOCA version used for compilation is the same as the version installed on BlueField used for testing.

Publishing

The publishing process is identical to the publishing process when [using a BlueField DPU](#).

Cross-compilation from Host

In a typical setup, developers prefer to work on a familiar host since compilation is often significantly faster there. Therefore, developers may work on their host while cross-compiling their project to BlueField's Arm architecture.



Setup

1. Install Docker and QEMU your host. See steps 1-4 under section [Setup](#).
2. Download the [doca-cross component](#) as described in the [NVIDIA DOCA Installation Guide for Linux](#) and unpack it under the `/root` directory.
Inside this directory one can find:
 - `arm64_armv8_linux_gcc` – cross file containing specific information about the cross compiler and the host machine
 - `DOCA_cross.sh` – script which handles all the required dependencies and pre-installations steps

- A .txt file used by the script
3. To load the development container, refer to section "Docker Deployment" of the [NVIDIA BlueField Container Deployment Guide](#).

Note

It is important to ensure that the same DOCA version is used in the development container and the DOCA metapackages installed on the host.

4. Start running the container using the container's tag while mapping the `doca-cross` directory to the container's `/doca_devel` directory:

```
sudo docker run -v /root/doca-cross:/doca_devel --privileged -it  
<imagename/ID>
```

Now the shell will be redirected to be within the container.

5. Run the preparation script to copy all the Arm dependencies required for DOCA's cross compilation. The script will be in the mapped directory named `doca_devel`.

```
(container) /# cd doca_devel/  
(container) /doca_devel# ./DOCA_cross.sh
```

6. Exit the container and run the same script from the host side:

```
(host) /root/doca-cross# ./DOCA_cross.sh
```

The `/root/doca-cross` directory is now fully configured and prepared for cross-compilation against DOCA.

7. Update the environment variables to point at the Linaro cross-compiler:

```
export PATH=${PATH}:/opt/gcc-linaro/<linaro_version_dir>/aarch64-linux-gnu/bin:/opt/gcc-linaro/<linaro_version_dir>/bin
```

Everything is set up and the cross-compilation can now be used.

Note

Make sure to update the command according to the Linaro version installed by the script in the previous step.

<linaro_version_dir> can be found under /opt/gcc-linaro/.

Note

Cross-compilation requires Meson version $\geq 0.61.2$ to be installed on the host. This is already provided as part of DOCA's installation.

DOCA and CUDA Setup

1. To cross-compile DOCA and CUDA applications, you must install [CUDA Toolkit 12.1](#):
 1. The first toolkit installation is for x86 architecture. Select `x86_64`.
 2. The second toolkit installation is for Arm. Select `arm64-sbsa` and then `cross`.
 3. Select your host operating system, architecture, OS distribution, and version and select the installation type. It is recommended to use the `deb (local)` type.
2. Execute the following exports:

```
export CPATH=/usr/local/cuda/targets/sbsa-linux/include:$CPATH
export LD_LIBRARY_PATH=/usr/local/cuda/targets/sbsa-
linux/lib:$LD_LIBRARY_PATH
export PATH=/usr/local/cuda/bin:/usr/local/cuda-11.6/bin:$PATH
```

3. Verify the meson version is at least 0.61.2 as provided with DOCA's installation.

Everything is set up and the cross-compilation can now be used.

Development

It is recommended to develop normally while remembering to compile using the cross-compilation configuration file `arm64_armv8_linux_gcc` which can be found under the `doca-cross` directory.

The following is an example procedure for cross-compiling DOCA applications from the host and to the Arm architecture:

1. Enable the meson cross-compilation option in `/opt/mellanox/doca/applications/meson_options.txt` by setting `enable_cross_compilation_to_dpu` to `true`.
2. Cross-compile the DOCA applications:

```
/opt/mellanox/doca/applications # meson cross-build --cross-file /root/doca-
cross/arm64_armv8_linux_gcc
/opt/mellanox/doca/applications # ninja -C cross-build
```

Info

The cross-compiled binaries are created under the `cross-build` directory.

3. Cross-compile the DOCA and CUDA application:

1. Set flag for GPU-enabled cross-compilation, `enable_gpu_support`, in `/opt/mellanox/doca/applications/meson_options.txt` to `true`.
2. Run the compilation command as follows:

```
/opt/mellanox/doca/applications # meson cross-build --cross-file  
/root/doca-cross/arm64_armv8_linux_gcc -Dcuda_ccbindir=aarch64-linux-  
gnu-g++  
/opt/mellanox/doca/applications # ninja -C cross-build
```

Info

The cross-compiled binaries are created under the `cross-build` directory.

Note


Due to the system's use of the `PKG_CONFIG_PATH` environment variable, it is crucial that the cross file include the following:

```
[built-in options]  
pkg_config_path = "
```

This definition, already provided as part of the supplied cross file, guarantees that meson does not accidentally use the build system's environment variable during the cross build.

Testing

While the compilation can be performed on top of the host, testing the compiled software must be done on top of a BlueField Platform. This is because the tested program is not able to access the devices needed for its successful execution, which mandates that the testing is performed on top of a physical BlueField.

 **Note**

Make sure that the DOCA version used for compilation is the same as the version installed on the BlueField Platform used for testing.

Publishing

The publishing process is identical to the publishing process when [using a BlueField DPU](#).

DOCA Programming Guide

The DOCA Programming Guide is intended for developers wishing to utilize DOCA SDK to develop application on top of the NVIDIA® BlueField® DPUs and SuperNICs.

- [DOCA Programming Overview](#) is important to read for new DOCA developers to understand the architecture and main building blocks most applications will rely on.
- [DOCA Development Best Practices](#) outlines common development pitfalls and capabilities to speed up application development, qualification, and productization.
- [DOCA Libraries](#) describes in details how to use each DOCA library, its APIs, and different aspects related to that library. Users may choose to only read the pages concerning DOCA libraries required for their application.
- [DOCA Utils](#) includes modules that may be used by application developers to speed up their development process (e.g., [DOCA Arg Parser](#) which simplifies the creation of a command-line interface for your application).
- [DOCA Drivers](#) describes additional frameworks used within DOCA.

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

DOCA Programming Overview

This section contains the following pages:

- [Hardware Overview](#)

- [DOCA SDK Architecture](#)

DOCA Backward Compatibility Policy

The NVIDIA DOCA™ SDK enables developers to rapidly create applications and services on top of NVIDIA® BlueField® networking platforms.

The DOCA software package is released on a quarterly release cadence to deliver new features, performance improvements, and critical bug fixes. DOCA compatibility allows users to update the latest DOCA software package (including all libraries, drivers, and tools) without requiring updating the application.

DOCA SDK Versioning

DOCA versions follow the [Semantic Versioning](#) scheme. That is, the DOCA version is of the form X.Y.Z, and each part is incremented when the following applies:

- Major version – when incompatible API changes may be introduced
- Minor version – when functionality is added in a backwards compatible manner
- Patch version – when backwards compatible bug fixes are submitted

DOCA SDK API Backwards Compatibility

One of the key attributes of enterprise grade SDK is backward compatibility. Backward compatible APIs allows application developers using the SDK to monetize on their investment, by guaranteeing that their application will continue to operate successfully as they update to a newer SDK version.

DOCA SDK APIs may go through the following lifecycle stages:

1. Experimental – an API marked as DOCA_EXPERIMENTAL is an experimental API and is not guaranteed to be present across upcoming releases
2. Stable – an API marked as DOCA_STABLE is guaranteed to be supported throughout the lifecycle of the current major version

3. **Deprecated** – an API marked as `DOCA_DEPRECATED` will be removed from DOCA SDKs header files in an upcoming release. If the API was previously marked as `DOCA_STABLE`, it will only be removed in an upcoming major release.
4. **Removed** – an API that was present on an older major version and is now no longer supported. If this API was previously marked as `DOCA_STABLE`, the binary representation is preserved to maintain binary backwards compatibility.

The following subsections explain the different backwards compatibility types including how semantic versions are mapped to these different types.

Source Compatibility

Source compatibility guarantees that a program written and compiled using a given DOCA SDK version compiles successfully against a newer DOCA SDK version.

As described in section "[DOCA SDK Versioning](#)", DOCA SDK is source compatible across minor and patch versions. However, across major version, APIs can be changed, deprecated, or removed (see the lifecycle stages under section "[DOCA SDK API Backwards Compatibility](#)"). Therefore, an application that compiles successfully on an older major DOCA SDK version of the toolkit may require changes to compile against a newer major version.

Binary Compatibility

Binary compatibility guarantees that a program dynamically linked against a given DOCA SDK library (*.so) successfully links against a newer DOCA SDK library.

DOCA SDK API has a versioned C-style application binary interface (ABI) which guarantees binary compatibility across both minor and major versions. This means that upgrading the DOCA SDK package installed on a system to a newer version always supports existing applications and their functions.

Behavioral Compatibility

Behavioral compatibility (i.e., semantic compatibility) guarantees that given the same inputs, a function or component will produce the same outputs. Thus, an application developed, compiled, linked, and tested with a given DOCA SDK and relying on the SDK's behavior, can successfully run with newer version of DOCA SDK, as the behavior will be compatible (apart from fixing bugs).

DOCA SDK Protocol Compatibility

Some DOCA SDK components include interaction across remote entities (host-to-BlueField, BlueField-to-BlueField, or host-to-host). That is, communication channel between a process running on the host server and a process running on the BlueField networking platform Arm processors. Since applications using DOCA may be deployed in large clusters and upgraded on a different schedule, DOCA SDK guarantees maintaining different DOCA SDK versions protocol-compatible with each other. This allows the flexibility to perform a rolling upgrade to DOCA SDK applications while maintaining operations throughout the process (nodes with different SDK versions maintain communication).

DOCA SDK Dependencies Compatibility

DOCA is distributed in a meta-package format, either as a *.bfb file for installation on the BlueField networking platform Arm processor, or as a DOCA-for-host package (*.rpm or *.deb) for installation on the server hosting the BlueField networking platform. This package includes different libraries, tools, executables, firmware, and sample applications.

DOCA SDK is developed and tested to work with all components included in the meta-package. There is no guarantee that DOCA SDK would work correctly if any of these components is upgraded independently. Thus, updating DOCA to a newer version requires updating the meta-package with all its components.

DOCA Development Best Practices

The following sub-sections describe some best practices DOCA SDK users/developers should consider when using DOCA SDK.

- [Capability Checking](#)

- [Debuggability](#)

DOCA Libraries

This section describes in details how to use each DOCA library, its APIs, and different aspects related to that library.

Users may choose to only read the pages concerning DOCA libraries required for their application.

This section contains the following pages:

- [DOCA Common](#)
- [DOCA Flow](#)
- [DPA Subsystem](#)
- [DOCA DMA](#)
- [DOCA Comch](#)
- [DOCA UROM](#)
- [DOCA RDMA](#)
- [DOCA Ethernet](#)
- [DOCA GPUNetIO](#)
- [DOCA App Shield](#)
- [DOCA Compress](#)
- [DOCA SHA](#)
- [DOCA Erasure Coding](#)
- [DOCA AES-GCM](#)
- [DOCA Rivermax](#)

- [DOCA Telemetry](#)
- [DOCA Device Emulation](#)

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

DOCA Utils

This section includes modules that may be used by application developers to speed up their development process.

This section contains the following pages:

- [DOCA Arg Parser](#)

DOCA Drivers

This section describes underlying drivers included in DOCA and includes the following pages:

- [DOCA UCX](#)
- [MLX Drivers \(MLNX_OFED\)](#)

DOCA Applications

This page provides an overview of the example DOCA applications implemented on top of NVIDIA® BlueField® DPU.

Introduction

DOCA applications are an educational resource provided as a guide on how to program on the NVIDIA BlueField networking platform using DOCA API.

For instructions regarding the development environment and installation, refer to the [NVIDIA DOCA Developer Guide](#) and the [NVIDIA DOCA Installation Guide for Linux](#) respectively.

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

Installation

DOCA applications are installed under `/opt/mellanox/doca/applications` with each application having its own dedicated folder. Each directory contains the source code and compilation files for the matching application.

Compilation

As applications are shipped alongside their sources, developers may want to modify some of the code during their development process and then recompile the applications. The files required for the compilation are the following:

- /opt/mellanox/doca/applications/meson.build – main compilation file for a project that contains all the applications
- /opt/mellanox/doca/applications/meson_options.txt – configuration file for the compilation process
- /opt/mellanox/doca/applications/<application_name>/meson.build – application-specific compilation definitions

To recompile all the reference applications:

1. Move to the applications directory:

```
cd /opt/mellanox/doca/applications
```

2. Prepare the compilation definitions:

```
meson /tmp/build
```

3. Compile all the applications:

```
ninja -C /tmp/build
```

Info

The generated applications are located under the /tmp/build/ directory, using the following path
/tmp/build/<application_name>/doca_<application_name>.

Note

Compilation against DOCA's SDK relies on environment variables which are automatically defined per user session upon login. For

more information, please refer to section "Meson Complains About Missing Dependencies" in the [NVIDIA DOCA Troubleshooting Guide](#).

Developer Configurations

When recompiling the applications, meson compiles them by default in "debug" mode. Therefore, the binaries would not be optimized for performance as they would include the debug symbol. For comparison, the application binaries shipped as part of DOCA's installation are compiled in "release" mode. To compile the applications in something other than debug, please consult [Meson's configuration guide](#).

The applications also offer developers the ability to use the DOCA log's TRACE level (DOCA_LOG_TRC) on top of the existing DOCA log levels. Enabling the TRACE log level during compilation activates various developer log messages left out of the release compilation. Activating the TRACE log level may be done through `enable_trace_log` in the `meson_options.txt` file, or directly from the command line:

1. Prepare the compilation definitions to use the trace log level:

```
meson /tmp/build -Denable_trace_log=true
```

2. Compile the applications:

```
ninja -C /tmp/build
```

Application Use of DOCA Libs

The following table maps DOCA reference applications to the libraries they make use of.

Application Category	Application	Library Category											
		BareMetal/Virtualized Cloud					Secure Cloud Gateway		Cloud Storage	Monitoring	Streaming	HPC	
		Flow	DPA	DMA	FlexIO SDK	PCC	App Shield	SHA	Compression	Telemetry	GPU NetIO	Cmch	UCX
Network	Ethernet L2 Forwarding												
	GPU Packet Processing												
	NAT												
	Simple Forward VNF												
	Switch												
Security	App Shield Agent												
	East-west Overlay Encryption												
	IPsec Security Gateway												
	PSP Gateway												
	Secure Channel												
	YARA Inspection												

Application Category	Application	Library Category											
Data Path Acceleration	DPA All-to-all												
	DPA L2 Reflector												
	PCC												
Storage	DMA Copy												
	File Compression												
	File Integrity												
HPC	Allreduce												
	UROM RDMO												

Applications

Allreduce

[This application](#) is a collective operation that allows data from many processing units to be collected and merged into a global result before being delivered to all processing units using an operator. The application is implemented using the UCX communication framework, which leverages the DPU's low-latency and high-bandwidth utilization of its network engine.

App Shield Agent

[This application](#) describes how to build secure process monitoring and is based on the DOCA APSH library, which leverages DPU capabilities such as regular expression (RXP) acceleration engine, hardware-based DMA, and more.

DMA Copy

[This application](#) describes how to transfer files between the DPU and the host. The application is based on the direct memory access (DMA) library, which leverages hardware acceleration for data copy for both local and remote memory.

DPA All-to-all

[This application](#) is a collective operation that allows data to be copied between multiple processes. This application is implemented using DOCA DPA, which leverages the data path accelerator (DPA) inside of the BlueField-3 which offloads the copying of the data to the DPA and leaves the CPU free for other computations.

DPA L2 Reflector

[This application](#) uses the data path accelerator (DPA) engine to intercept network traffic and swap the source and destination MAC addresses of each packet. It is based on the FlexIO API which leverages DPU capabilities such as high-speed DPA.

East-West Overlay Encryption

[This application](#) (IPsec) sets up encrypted connections between different devices and works by encrypting IP packets and authenticating the packets' originator. It is based on a strongSwan solution which is an open-source IPsec-based VPN solution.

File Compression

[This application](#) shows how to compress and decompress data using hardware acceleration and to send and receive it. The application is based on the DOCA Compress and DOCA Comm-Channel libraries.

File Integrity

[This application](#) shows how to send and receive files in a secure way using the hardware Crypto engine. It is based on the DOCA SHA and DOCA Comm-Channel libraries.

GPU Packet Processing

[This application](#) shows how to combine DOCA GPUNetIO, DOCA Ethernet, and DOCA Flow to manage ICMP, UDP, TCP and HTTP connections with a GPU-centric approach using CUDA kernels without involving the CPU in the main data path.

IPsec Gateway

[This application](#) demonstrates how to insert rules related to IPsec encryption and decryption based on the DOCA Flow and IPsec libraries, which leverage the DPU's hardware capability for secure network communication.

NAT

[This application](#), network address translation, switches packets with local IP addresses to global ones and vice versa. It is based on the DOCA Flow library which leverages DPU hardware capabilities such as building generic execution pipes in the hardware, executing specific actions on the traffic, and more.

Programmable Congestion Control

[This application](#), programmable congestion control, is based on the DOCA PCC library and allows users to design and implement their own congestion control algorithm, giving them good flexibility to work out an optimal solution to handle congestion in their clusters.

PSP Gateway

[This application](#) demonstrates how to exchange keys between application instances and insert rules controlling PSP encryption and decryption using the DOCA Flow library.

Secure Channel

[This application](#) is used to establish a secure, network-independent communication channel between the host and the DPU based on the DOCA Comm Channel library.

Simple Forward VNF

[This application](#) is a forwarding application that takes VXLAN traffic from a single RX port and transmits it on a single TX port. It is based on the DOCA Flow library which leverages DPU capabilities such as building generic execution pipes in the hardware, and more.

Switch

[This application](#) is used to establish internal switching between representor ports on the DPU. It is based on the DOCA Flow library which leverages DPU capabilities such as building generic execution pipes in the hardware, and more.

UROM RDMO

[This application](#) demonstrates how to execute an Active Message outside the context of the target process. It is based on the DOCA UROM (Unified Resources and Offload Manager) library as a framework to launch UROM workers on the DPU and using the UCX communication framework, which leverages the DPU's low-latency and high-bandwidth utilization of its network engine.

UROM UCC

[This application](#) demonstrates how the UCC all-to-all collective can be offloaded to the BlueField using the DOCA UROM library and UCC UROM plugin.

YARA Inspection

[This application](#) describes how to build YARA rule inspection for processes and is based on the DOCA APSH library, which leverages DPU capabilities such as the regular expression (RXP) acceleration engine, hardware-based DMA, and more.

NVIDIA DOCA Allreduce Application Guide

This guide provides a DOCA Allreduce collective operation implementation on top of NVIDIA® BlueField® DPU using UCX.

Introduction

Allreduce is a collective operation which allows collecting data from different processing units to combine them into a global result by a chosen operator. In turn, the result is distributed back to all processing units.

Allreduce operates in stages. Firstly, each participant scatters its vector. Secondly, each participant gathers the vectors of the other participants. Lastly, each participant performs their chosen operation between all the gathered vectors. Using a sequence of different allreduce operations with different participants, very complex computations can be spread among many computation units.

Allreduce is widely used by parallel applications in high-performance computing (HPC) related to scientific simulations and data analysis, including machine learning calculation

and the training phase of neural networks in deep learning.

Due to the massive growth of deep learning models and the complexity of scientific simulation tasks that utilize a network, effective implementation of allreduce is essential for minimizing communication time.

This document describes how to implement allreduce using the UCX communication framework, which leverages NVIDIA® BlueField® DPU by providing low-latency and high-bandwidth utilization of its network engine.

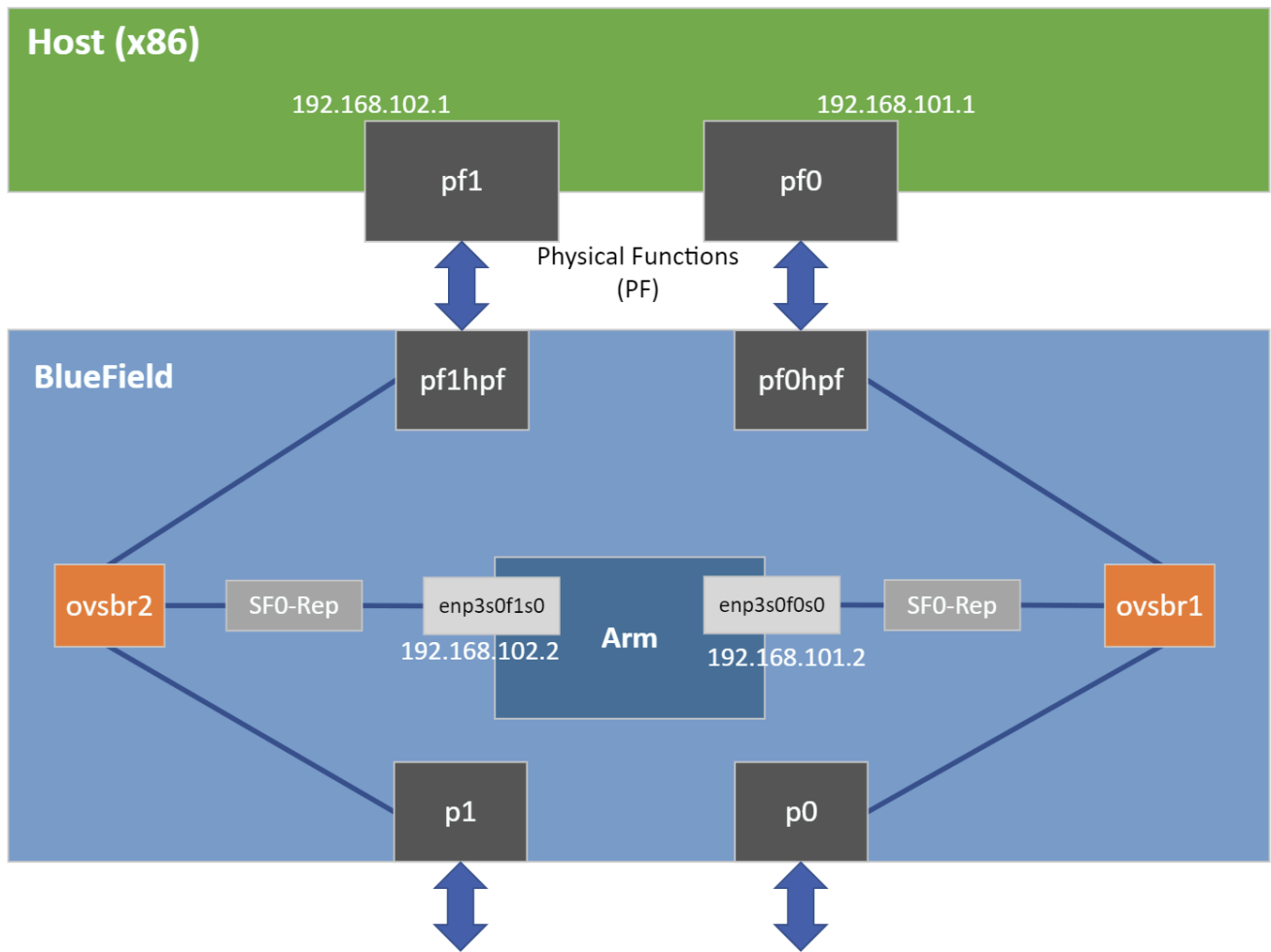
This document describes the following types of allreduce:

- Offloaded client – processes running on the host which only submit allreduce operation requests to a daemon running on the DPU. The daemon runs on the DPU and performs the allreduce algorithm on behalf of its on-host-clients (offloaded-client).
- Non-offloaded client – processes running on the host which execute the allreduce algorithm by themselves

System Design

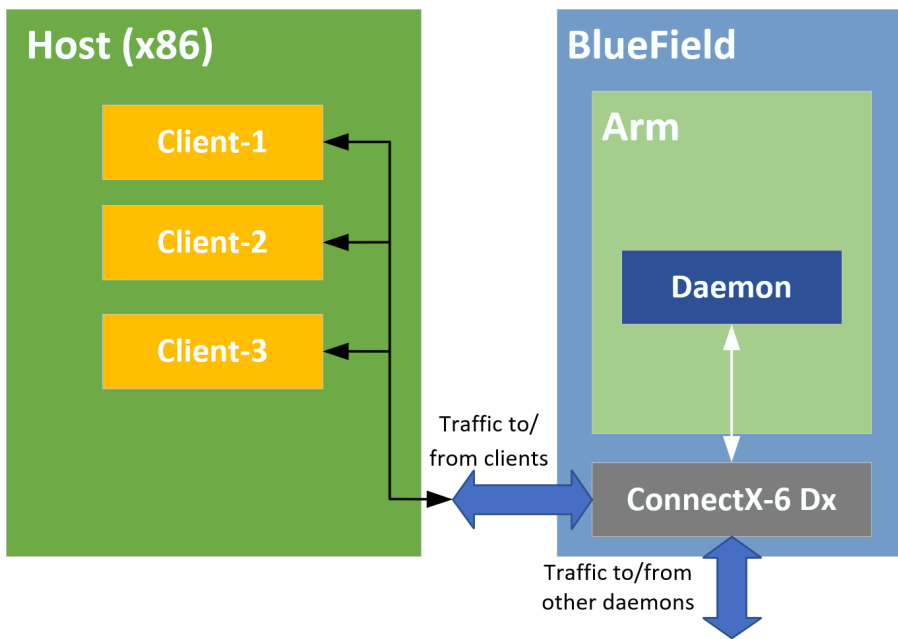
The application is designed to measure three metrics:

- Communication time taken by offloaded and non-offloaded allreduce operations
- Computation time taken by matrix multiplications which are done by clients until the allreduce operation is completed
- The overlap of the two previous metrics. The percentage of the total runtime during which both the allreduce and the matrix multiplications were done in parallel.

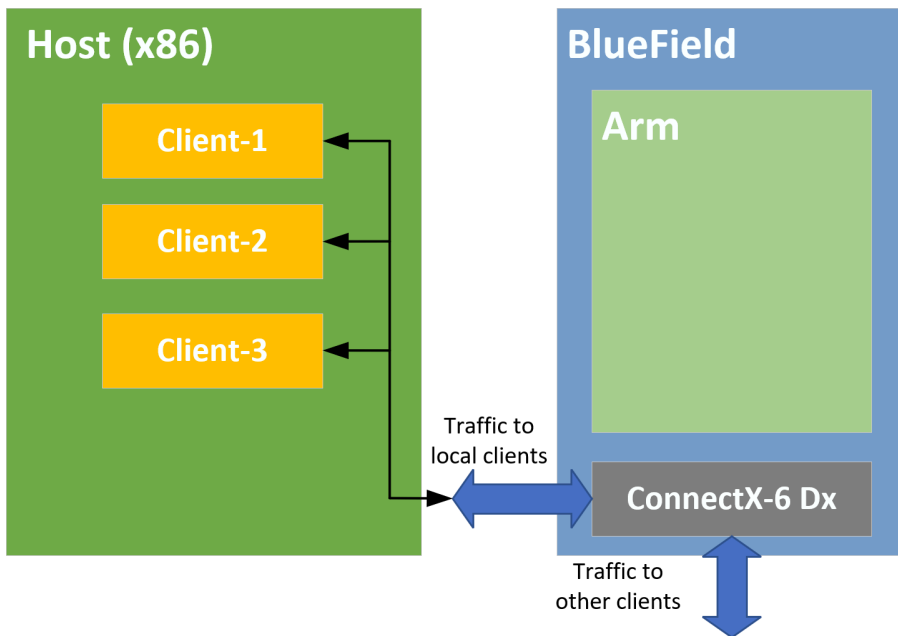


The allreduce implementation is divided into two different types of processes: clients and daemons. Clients are responsible for allocating vectors filled with data and initiating allreduce operations by sending a request with a vector to their daemon. Daemons are responsible for gathering vectors from all connected clients and daemons, applying a chosen operator on all received buffers, and then scattering the reduced result vector back to the clients.

- Offloaded mode



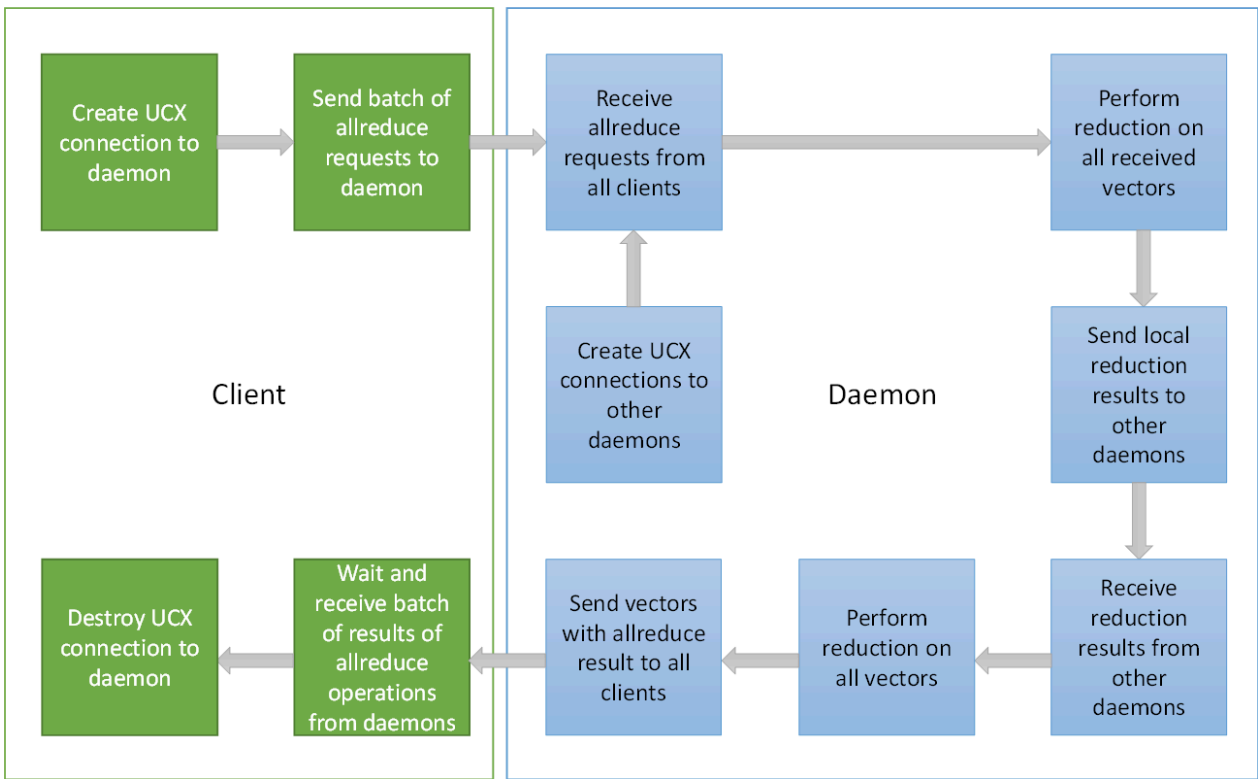
- Non-offloaded mode



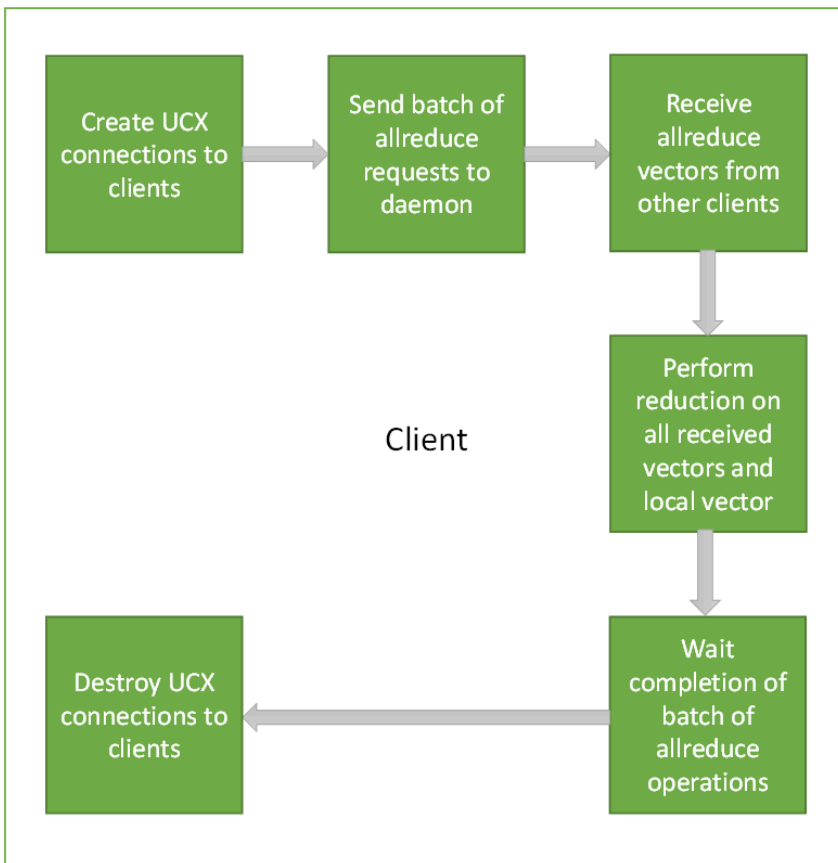
Application Architecture

DOCA's allreduce implementation uses Unified Communication X (UCX) to support data exchange between endpoints. It utilizes UCX's sockaddr-based connection establishment and the UCX Active Messages (AM) API for communications.

- Offloaded mode



- Non-offloaded mode



1. Connections between processes are established by UCX using IP addresses and ports of peers.
2. Allreduce vectors are sent from clients to daemons in offloaded mode, or from clients to clients in non-offloaded mode.
3. Reduce operations on vectors are done using received vectors from other daemons in offloaded mode, or other clients in non-offloaded mode.
4. Vectors with allreduce results are received by clients from daemons in offloaded mode, or are already stored in clients after completing all exchanges in non-offloaded mode.
5. After completing all allreduce operations, connections between clients are destroyed.

DOCA Libraries

This application leverages the UCX framework DOCA driver.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/allreduce/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

`doca_allreduce` is created under `/tmp/build/allreduce/`.

Compiling Only the Current Application

To build the allreduce application only:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_allreduce=true  
ninja -C /tmp/build
```

Info

doca_allreduce is created under `/tmp/build/allreduce/`.

Alternatively, the user can set the desired flags in the `meson_options.txt` file instead of providing them in the compilation command line:

1. Edit the following flags in `/opt/mellanox/doca/applications/meson_options.txt`:

- Set `enable_all_applications` to `false`
- Set `enable_allreduce` to `true`

2. Run the following compilation commands:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_allreduce is created under `/tmp/build/allreduce/`.

Troubleshooting

Please refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Application Execution

The allreduce application is provided in source form, hence a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_allreduce [DOCA Flags] [Program Flags]
```

DOCA Flags:

```
-h, --help Print a help synopsis  
-v, --version Print program version information  
-l, --log-level Set the (numeric) log level for the program <10=DISABLE,  
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>  
--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE,  
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>  
-j, --json <path> Parse all command flags from an input json file
```

Program Flags:

```
-r, --role Run DOCA UCX allreduce process as: "client" or "daemon"  
-m, --mode <allreduce_mode> Set allreduce mode: "offloaded", "non-offloaded"  
(valid for client only)  
-p, --port <port> Set default destination port of daemons/clients, used for IPs  
without a port (see '-a' flag)  
-t, --listen-port <listen_port> Set listening port of daemon or client  
-c, --num-clients <num_clients> Set the number of clients which participate in  
allreduce operations (valid for daemon only)  
-s, --size <size> Set size of vector to do allreduce for  
-d, --datatype <datatype> Set datatype ("byte", "int", "float", "double") of vector  
elements to do allreduce for  
-o, --operation <operation> Set operation ("sum", "prod") to do between allreduce  
vectors  
-b, --batch-size <batch_size> Set the number of allreduce operations submitted  
simultaneously (used for handshakes by daemons)
```

`-i, --num-batches <num_batches>` Set the number of batches of allreduce operations (used [for](#) handshakes by daemons)
`-a, --address <ip_address>` Set comma-separated list of destination IPv4/IPv6 addresses and ports optionally (`<ip_addr>:[<port>]`) of daemons or clients

Info

This usage printout can be printed to the command line using the `-h` (or `--help`) options:

```
./doca_allreduce -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. Configuration steps.

1. All daemons should be deployed before clients. Only after connecting to their peers are daemons able to handle clients.
2. UCX probes the system for any available net/IB devices and, by default, tries to create a multi-device connection. This means that if some network devices are available but provide an unreachable path from the daemon to the peer/client, UCX may still use that path. A common case is that a daemon tries to connect to a different BlueField using `tmfifo_net0` which is connected to the host only. To fix this issue, follow these steps:

1. Use the UCX env variable `UCX_NET_DEVICES` to set usable devices. For example:

```
export UCX_NET_DEVICES=enp3s0f0s0,enp3s0f1s0
```

```
./doca_allreduce -r daemon -t 34001 -c 1 -s 100 -o sum -d float -b 16 -i 16
```

Or:

```
env UCX_NET_DEVICES=enp3s0f0s0,enp3s0f1s0 ./doca_allreduce -r daemon -t 34001 -c 1 -s 100 -o sum -d float -b 16 -i 16
```

2. Get the mlx device name and port of a SF to limit the UCX network interfaces and allow IB. For example:

```
BlueField> show_gids
DEV PORT INDEX GID IPv4 VER DEV
-----
mlx5_2 1 0 fe80:0000:0000:0000:0052:72ff:fe63:1651 v2 enp3s0f0s0
mlx5_3 1 0 fe80:0000:0000:0000:0032:6bff:fe13:f13a v2 enp3s0f1s0

BlueField>
UCX_NET_DEVICES=enp3s0f0s0,enp3s0f1s0,mlx5_2:1,mlx5_3:1
./doca_allreduce -r daemon -t 34001 -c 1 -s 100 -o sum -d float -b 16 -i 16
```

3. CLI example for running the daemon on BlueField:

```
./doca_allreduce -r daemon -t 34001 -c 2 -a 10.21.211.3:35001,10.21.211.4:36001 -s 65535 -o sum -d float -i 16 -b 128
```

Notes:

- The flag `-a` is necessary for communicating with other daemons. In case of an offloaded client, the address must be that of the daemon which performs the allreduce operations for them. In case of a daemon or non-offloaded clients, the flag could be a single or multiple addresses of other daemons/non-offloaded clients which exchange their local allreduce results.

- The flag `-c` must be specified for daemon processes only. It indicates how many clients submit their allreduce operations to the daemon.
- The flags `-s`, `-i`, `-b`, and `-d` must be the same for all clients and daemons participating in the allreduce operation.

Note

The daemon listens to incoming connection requests on all available IPs, but the actual communication after the initial "UCX handshake" does not necessarily use the same device used for the connection establishment.

4. CLI example for running the client on the host:

```
./doca_allreduce -r client -m non-offloaded -t 34001 -a  
10.21.211.3:35001,10.21.211.4:36001 -s 65535 -i 16 -b 128 -o sum -d float  
./doca_allreduce -r client -m offloaded -p 34001 -a 192.168.100.2 -s 65535 -i 16 -b  
128 -o sum -d float
```

5. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_allreduce --json [json_file]
```

For example:

```
./doca_allreduce --json ./allreduce_client_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the desired

PCIe and network addresses required for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description
General flags	h	help	Print a help synopsis
	v	version	Print program version information
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with TRACE log level support)
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70
	j	json	Parse all command flags from an input JSON file
Program flags	r	role	Run DOCA UCX allreduce process as either client or daemon
	m	mode	Set allreduce mode. Available types options: <ul style="list-style-type: none"> • offloaded

Flag Type	Short Flag	Long Flag/JSON Key	Description
			<ul style="list-style-type: none"> non-offloaded (valid for client only)
	p	port	Set default destination port of daemons/clients. Used for IPs without a port (see -a flag).
	c	num-clients	Set the number of clients which participate in allreduce operations Note: Valid for daemon only.
	s	size	Set size of vector to perform allreduce for
	d	datatype	Set datatype of vector elements to do allreduce for <ul style="list-style-type: none"> byte int float double
	o	operation	Set operation to perform between allreduce vectors
	b	batch-size	Set the number of allreduce operations submitted simultaneously. Used for handshakes by daemons.
	i	num-batches	Set the number of batches of allreduce operations. Used for handshakes by daemons.
	t	listen-port	Set listening port of daemon or client
	a	address	Set comma-separated list of destination IPv4/IPv6 address and ports optionally of daemons or clients. Format: <ip_addr>: [<port>].

Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.

Running Application on NVIDIA Converged Accelerator

This section details the steps necessary to run DOCA Allreduce on NVIDIA converged accelerator.

Allreduce running on the converged accelerator has the same logic as described in previous sections except for the reducing of vectors. The reduce of incoming vectors is performed on the GPU side in batches that include the vectors from all peers or all clients. When the GPUDirect module is active, incoming vectors and outgoing vectors are received/sent directly to/from the GPU.

To make use of the GPU's capabilities, make sure to perform the following:

1. Refer to the [NVIDIA DOCA Installation Guide for Linux](#) for instructions on installing NVIDIA driver for CUDA and a CUDA-repo on your setup.
2. Create the sub-functions and configure the OVS according to [NVIDIA BlueField DPU Scalable Function User Guide](#).

Compiling and Running Application

To build and run the application:

1. Setup CUDA paths:

```
export CPATH=/usr/local/cuda/targets/sbsa-linux/include:$CPATH
export
LD_LIBRARY_PATH=/usr/local/nvidia/lib:/usr/local/nvidia/lib64:$LD_LIBRARY_PATH
export PATH=/usr/local/nvidia/bin:/usr/local/cuda/bin:$PATH
```

2. Reinstall UCX with CUDA support. Follow the [UCX installation procedure](#) with an additional flag, `--with-cuda=/usr/local/cuda/`, passed to `configure-release`:

```
dpu# ./contrib/configure-release --with-cuda=/usr/local/cuda/
```

3. To build the application with GPU support:

1. Set the `enable_gpu_support` flag to true in `/opt/mellanox/doca/applications/meson_option.txt`.

2. Compile the application sources. Run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

`doca_allreduce_gpu` is created under `/tmp/build/allreduce/` alongside the regular `doca_allreduce` binary that is compiled without the GPU support.

4. To run the application with GPU support, follow the same steps as described in section "[Running the Application](#)".

Application Code Flow

1. Parse application argument.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register UCX allreduce application parameters.

```
register_allreduce_params();
```

3. Parse all registered parameters.


```
doca_argp_start();
```

2. UCX initialization.

1. Initialize hash table of connections.

```
allreduce_ucx_init();
```

2. Create UCP context.

```
ucp_init();
```

3. Create UCP worker.

```
ucp_worker_create();
```

4. Set AM handler for receiving connection check packets.

```
ucp_worker_set_am_rcv_handler();
```

3. Initialization of the allreduce connectivity.

```
communication_init();
```

1. Initialize hash table of allreduce super requests.

2. Set "receive callback" for handshake messages.

3. If daemon or non-offloaded client:

1. Set AM handler for receiving allreduce requests from clients.

```
allreduce_ucx_am_set_rcv_handler();
```

2. Initialize UCX listening function. This creates a UCP listener.

```
allreduce_ucx_listen();
```

4. Initialize all connections.

```
connections_init();
```

1. Go over all destination addresses and connect to each peer.

2. Repeat until a successful send occurs (to check connectivity).

```
ucp_am_send_nbx();  
allreduce_ucx_request_wait();
```

3. Insert the connection to the hash table of connections.

```
allreduce_outgoing_handshake();
```

5. Scatter handshake message to peers/daemon to make sure they all have the same -s, -i, -b, and -d flags.

4. Daemon: Start UCX progress.

```
daemon_run();
```

1. Set AM handler to receive allreduce requests from clients.

```
allreduce_ucx_am_set_rcv_handler();
```

2. Perform UCP worker progress.

```
while (running)  
allreduce_ucx_progress();
```

3. Callbacks are invoked by incoming/outgoing messages by calling `allreduce_ucx_progress`.

5. Client:

```
client_run();
```

1. Allocate buffers to store allreduce initial data and results.

```
allreduce_vectors_init();
```

2. Set an AM handler for receiving allreduce results.

```
allreduce_ucx_am_set_recv_handler();
```

3. Perform allreduce barrier. Check that all daemons and clients are active.

```
allreduce_barrier();
```

1. Submit a batch of allreduce operations with 0 byte.
2. Wait for completions.

4. Reset metrics and vectors.

```
allreduce_metrics_init();
```

1. Submit some batches and calculate estimated network time.
2. Allocate matrices to multiply.
3. Estimate how many matrix multiplications could have been performed instead of networking (same time window).
4. Calculate the actual computation time of these matrix multiplications.

5. Reset vectors.

6. Submit a batch of allreduce operations to daemon/peer (depends on mode).
7. Perform matrix multiplications during a time period which is approximately equal to doing a single batch of allreduce operations and calculate the actual time cost.
8. Wait for the allreduce operation to complete and calculate time cost.
9. Update metrics.

```
Do num-batches (flag) times:  
allreduce_vectors_reset();  
allreduce_batch_submit();  
cpu_exploit();  
allreduce_batch_wait();  
allreduce_metrics_calculate();
```

10. Print summary of allreduce benchmarking.

```
allreduce_metrics_print();
```

6. Arg parser destroy.

```
doca_argp_destroy();
```

7. Communication destroy.

1. Clean up connections.

```
allreduce_ucx_disconnect();
```

1. Remove the connection from the hash table of the connections.
2. Close inner UCP endpoint.

```
ucp_ep_close_nbx();
```

3. Wait for the completion of the UCP endpoint closure.
 4. Destroy connection.
 5. Free connections array.
2. Destroy the hash table of the allreduce super requests.
8. Destroy UCX context.

1. Destroy the hash table of the connections.

```
g_hash_table_destroy();
```

2. If the UCP listener was created, destroy it.

```
ucp_listener_destroy();
```

3. Destroy UCP worker.

```
ucp_worker_destroy();
```

4. Destroy UCP context.

```
ucp_cleanup();
```

References

- [/opt/mellanox/doca/applications/allreduce/](#)
- [/opt/mellanox/doca/applications/allreduce/allreduce_client_params.json](#)
- [/opt/mellanox/doca/applications/allreduce/allreduce_daemon_params.json](#)

NVIDIA DOCA App Shield Agent Application Guide

This guide provides process introspection system implementation on top of NVIDIA® BlueField® DPU.

Introduction

App Shield Agent monitors a process in the host system using the DOCA App Shield library.

This security capability helps identify corruption of core processes in the system from an independent and trusted DPU. This is a major and innovate intrusion detection system (IDS) ability since it cannot be provided from inside the host.

The [DOCA App Shield Library](#) gives the capability to read, analyze, and authenticate the host (bare metal/VM) memory directly from the DPU.

Using the library, this application hashes the un-writable memory pages (also unloaded pages) of a specific process and its libraries. Then, at regular intervals, the app authenticates the loaded pages.

The app reports pass/fail after every iteration until the first attestation failure. The reports are both printed to the console and exported to the [DOCA Telemetry Service](#) (DTS) using inter-process communication (IPC).

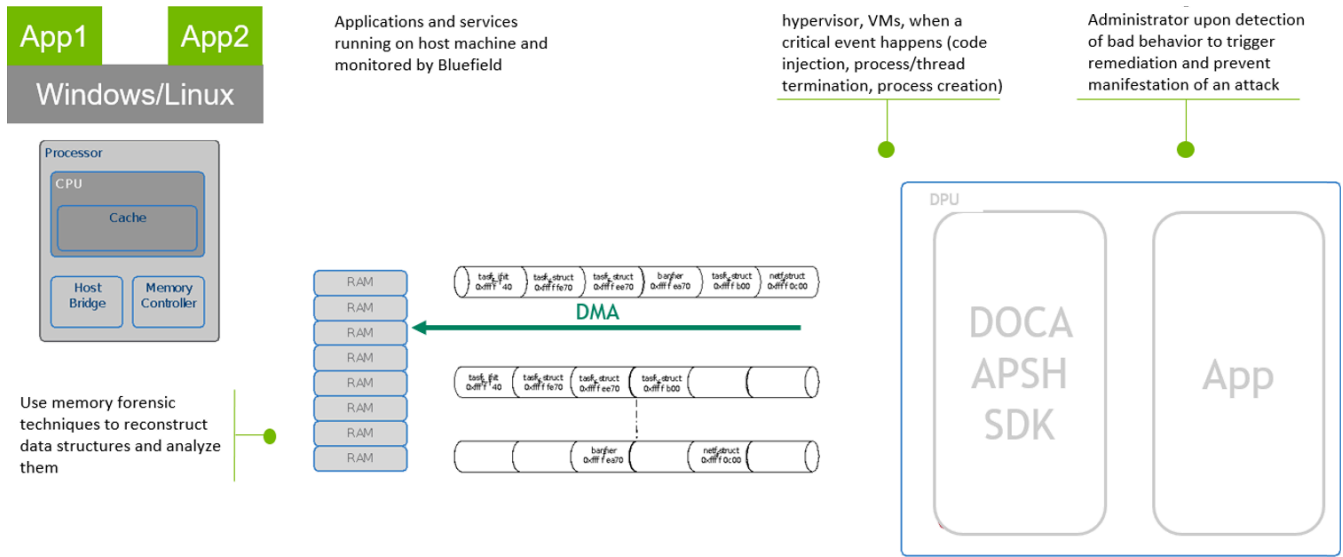
This guide describes how to build secure process monitoring using the DOCA App Shield library, which leverages the DPU's advantages such as hardware-based DMA, integrity, and more.

System Design

The App Shield agent is designed to run independently on the DPU's Arm without hindering the host.

The host's involvement is limited to configuring monitoring of a new process when there is a need to generate the needed ZIP and JSON files to pass to the DPU. This is done at inception ("time 0") which is when the host is still in a "safe" state.

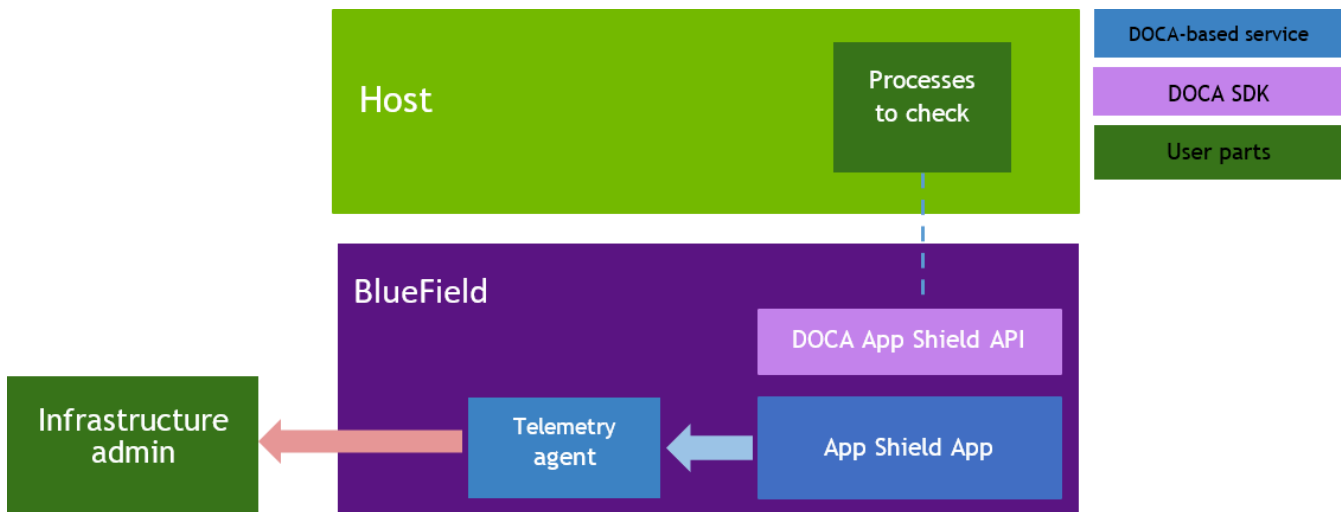
Generating the needed files can be done by running DOCA App Shield's `doca_apsh_config.py` tool on the host. See [DOCA App Shield](#) for more info.



Application Architecture

The user creates three mandatory files using the DOCA tool `doca_apsh_config.py` and copies them to the DPU. The application can report attestation results to the:

- File
- Terminal
- DTS



1. The files are generated by running `doca_apsh_config.py` on the host against the process at time zero.

Note

The actions 2-5 recur at regular time intervals.

2. The App Shield agent requests new attestation from DOCA App Shield library.
3. The DOCA App Shield library creates a new attestation:
 1. Scans and hashes process memory pages (that are currently in use).
 2. Compares the hash to the original hash.
 3. Creates attestation for each lib/exe involved in the process. Each of attestation includes the number of valid pages and the number of pages.
4. The App Shield agent searches each attestation for inconsistency between number of used pages and number of valid pages.
5. The App Shield agent reports results with a timestamp and scan count to:
 1. Local telemetry files – a folder and files representing the data a real DTS would have received. These files are used for the purposes of this example only as normally this data is not exported into user-readable files.
 2. DOCA log (without scan count).
 3. DTS IPC interface (even if no DTS is active).
6. The App Shield agent exits on first attestation failure.

DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA App Shield](#)

- [DOCA Telemetry](#).

Refer to their respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/app_shield_agent/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/
```

```
meson /tmp/build
ninja -C /tmp/build
```

Info

doca_app_shield_agent is created under /tmp/build/app_shield_agent/.

Compiling Only the Current Application

To build only the App Shield Agent application:

```
cd /opt/mellanox/doca/applications/
meson /tmp/build -Denable_all_applications=false -Denable_app_shield_agent=true
ninja -C /tmp/build
```

Info

doca_app_shield_agent is created under /tmp/build/app_shield_agent/.

Alternatively, the user can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
 - o Set enable_all_applications to false
 - o Set enable_app_shield_agent to true
2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_app_shield_agent is created under /tmp/build/app_shield_agent/.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Prerequisites

1. Configure the BlueField's firmware.
 1. On the BlueField system, configure the PF base address register and NVMe emulation. Run:

```
dpu> mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_SIZE=2  
PF_BAR2_ENABLE=1 NVME_EMULATION_ENABLE=1
```

2. Perform a [BlueField system reboot](#) for the mlxconfig settings to take effect.
3. You may verify these configurations using the following command:

```
dpu> mlxconfig -d /dev/mst/mt41686_pciconf0 q | grep -E "NVME|BAR"
```

2. Download target system (host/VM) symbols.

- For Ubuntu:

```
host> sudo tee /etc/apt/sources.list.d/ddebs.list << EOF
deb http://ddebs.ubuntu.com/ $(lsb_release -cs) main restricted universe multiverse
deb http://ddebs.ubuntu.com/ $(lsb_release -cs)-updates main restricted universe
multiverse
deb http://ddebs.ubuntu.com/ $(lsb_release -cs)-proposed main restricted universe
multiverse
EOF
host> sudo apt install ubuntu-dbgsym-keyring
host> sudo apt-get update
host> sudo apt-get install linux-image-$(uname -r)-dbgsym
```

- For CentOS:

```
host> yum install --enablerepo=base-debuginfo kernel-devel-$(uname -r)
kernel-debuginfo-$(uname -r) kernel-debuginfo-common-$(uname -
m)-$(uname -r)
```

- No action is needed for Windows

3. Perform IOMMU passthrough. This stage is only necessary if IOMMU is not enabled by default (e.g., when the host is using an AMD CPU).

Note

Skip this step if you are not sure whether it is needed. Return to it only if DMA fails with a message similar to the following in dmesg:

```
host> dmesg
```

```
[ 3839.822897] mlx5_core 0000:81:00.0: AMD-Vi: Event
logged [IO_PAGE_FAULT domain=0x0047
address=0x2a0aff8 flags=0x0000]
```

1. Locate your OS's grub file (most likely `/boot/grub/grub.conf`, `/boot/grub2/grub.cfg`, or `/etc/default/grub`) and open it for editing. Run:

```
host> vim /etc/default/grub
```

2. Search for the line defining `GRUB_CMDLINE_LINUX_DEFAULT` and add the argument `iommu=pt`. For example:

```
GRUB_CMDLINE_LINUX_DEFAULT="iommu=pt <intel/amd>_iommu=on"
```

3. Run:

Note

Prior to performing a power cycle, make sure to do a graceful shutdown.

- For Ubuntu:

```
host> sudo update-grub
host> ipmitool power cycle
```

- For CentOS:

```
host> grub2-mkconfig -o /boot/grub2/grub.cfg
host> ipmitool power cycle
```

- For Windows targets, turn off Hyper-V capability.

4. Prepare target:

1. Install DOCA on the target system.
2. Create the ZIP and JSON files. Run:

```
target-system> cd /opt/mellanox/doca/tools/  
target-system> python3 doca_apsh_config.py --pid <pid-of-process-to-  
monitor> --os <windows/linux> --path <path to dwarf2json executable or  
pdbparse-to-json.py>  
target-system> cp /opt/mellanox/doca/tools/*.* <shared-folder-with-  
baremetal>  
dpu> scp <shared-folder-with-baremetal>/*.* <path-to-app-shield-binary>
```

If the target system does not have DOCA installed, the script can be copied from the BlueField.

The required dwarf2json and pdbparse-to-json.py are not provided with DOCA.

Note

If the kernel and process .exe have not changed, there no need to redo this step.

Application Execution

1. The App Shield Agent application is provided in source form, hence a compilation is required before the application can be executed.

1. Application usage instructions:


Usage: doca_app_shield_agent [DOCA Flags] [Program Flags]

DOCA Flags:

- h, --help Print a help synopsis
- v, --version Print program version information
- l, --log-level Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
- sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
- j, --json <path> Parse all command flags from an input json file


Program Flags:

- p, --pid Process ID of process to be attested
- e, --ehm <path> Exec hash map path
- m, --memr <path> System memory regions map
- f, --vuid VUID of the System device
- d, --dma DMA device name
- o, --osym <path> System OS symbol map path
- s, --osty <windows|linux> System OS type - windows/linux
- t, --time <seconds> Scan time interval in seconds

 **Info**

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_app_shield_agent -h
```

 **Info**

For additional information, please refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the BlueField:

```
./doca_app_shield_agent -p 13577 -e hash.zip -m mem_regions.json -o symbols.json -f MT2125X03335MLNXS0D0F0VF1 -d mlx5_0 -t 3 -s linux
```

Note

All used identifiers (-f, -p and -d flags) should match the identifier of the desired devices and processes.

Command Line Flags

Flag Type	Short Flag	Long Flag	Description
General flags	h	help	Print a help synopsis
	v	version	Print program version information
	l	log-level	Set the log level for the application: <ul style="list-style-type: none">• DISABLE=10• CRITICAL=20• ERROR=30• WARNING=40• INFO=50• DEBUG=60• TRACE=70 (requires compilation with TRACE log level support)

Flag Type	Short Flag	Long Flag	Description
	N/A	sdk-log-level	<p>Set the log level for the program:</p> <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70
	j	json	Parse all command flags from an input JSON file
Program flags	p	pid	PID of the process to be attested
	e	ehm	Path to the pre-generated hash.zip file transferred from the host
	m	memr	Path to the pre-generated mem_regions.json file transferred from the host
	f	pcif	<p>System PCIe function vendor unique identifier (VUID) of the VF/PF exposed to the target system. Used for DMA operations. To obtain this argument, run:</p> <pre>target-system> lspci -vv grep "[VU\] Vendor specific:"</pre> <p>Example output:</p> <pre>[VU] Vendor specific: MT2125X03335MLNXS0D0F0 [VU] Vendor specific: MT2125X03335MLNXS0D0F1</pre> <p>Two VUIDs are printed for each DPU connected to the target system. The first is of the DPU on pf0 and the second is of the DPU on port pf1.</p> <p>Note Running this command on the DPU outputs VUIDs with an additional "EC" string in the</p>

Flag Type	Short Flag	Long Flag	Description
			<p>middle. You must remove the "EC" to arrive at the correct VUID.</p> <p>The VUID of a VF allocated on PF0/1 is the VUID of the PF with an additional suffix, VF<vf-number>, where vf-number is the VF index +1. For example, for the output in the example above:</p> <ul style="list-style-type: none"> • PF0 VUID = MT2125X03335MLNXS0D0F0 • PF1 VUID = MT2125X03335MLNXS0D0F1 • VUID of VF0 on PF0 = MT2125X03335MLNXS0D0F0VF1 <p>VUIDs are persistent even on reset.</p>
	d	dma	DMA device name to use
	o	osym	Path to the pre-generated symbols.json file transferred from the host
	s	osty	OS type (windows or linux) of the system where the process is running
	t	time	Number of seconds to sleep between scans

i Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.

Application Code Flow

1. Parse application argument.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register application parameters.

```
register_apsh_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

2. Initialize DOCA App Shield lib context.

1. Create lib context.

```
doca_apsh_create();
```

2. Set DMA device for lib.

```
doca_devinfo_list_create();  
doca_dev_open();  
doca_devinfo_list_destroy();  
doca_apsh_dma_dev_set();
```

3. Start the context

```
doca_apsh_start();
```

```
apsh_system_init();
```

3. Initialize DOCA App Shield lib system context handler.

1. Get the representor of the remote PCIe function exposed to the system.

```
doca_devinfo_remote_list_create();  
doca_dev_remote_open();  
doca_devinfo_remote_list_destroy();
```

2. Create and start the system context handler.

```
doca_apsh_system_create();  
doca_apsh_sys_os_symbol_map_set();  
doca_apsh_sys_mem_region_set();  
doca_apsh_sys_dev_set();  
doca_apsh_sys_os_type_set();  
doca_apsh_system_start();
```

4. Find target process by pid.

```
doca_apsh_processes_get();
```

5. Telemetry initialization.

```
telemetry_start();
```

1. Initialize a new telemetry schema.
2. Register attestation type event.
3. Set up output to file (in addition to default IPC).
4. Start the telemetry schema.
5. Initialize and start a new DTS source with the `gethostname()` name as source ID.

6. Get initial attestation of the process.

```
doca_apsh_attestation_get();
```

7. Loop until attestation validation fail.

```
doca_apsh_attst_refresh();  
/* validation logic */  
doca_telemetry_source_report();  
DOCA_LOG_INFO();  
sleep();
```

8. DOCA App Shield Agent destroy.

```
doca_apsh_attestation_free();  
doca_apsh_processes_free();  
doca_apsh_system_destroy();  
doca_apsh_destroy();  
doca_dev_close();  
doca_dev_remote_close();
```

9. Telemetry destroy.

```
telemetry_destroy();
```

10. Arg parser destroy.

```
doca_argp_destroy();
```

References

- /opt/mellanox/doca/applications/app_shield_agent/

NVIDIA DOCA DMA Copy Application Guide

This guide provides an example of a DMA Copy implementation on top of NVIDIA® BlueField® DPU.

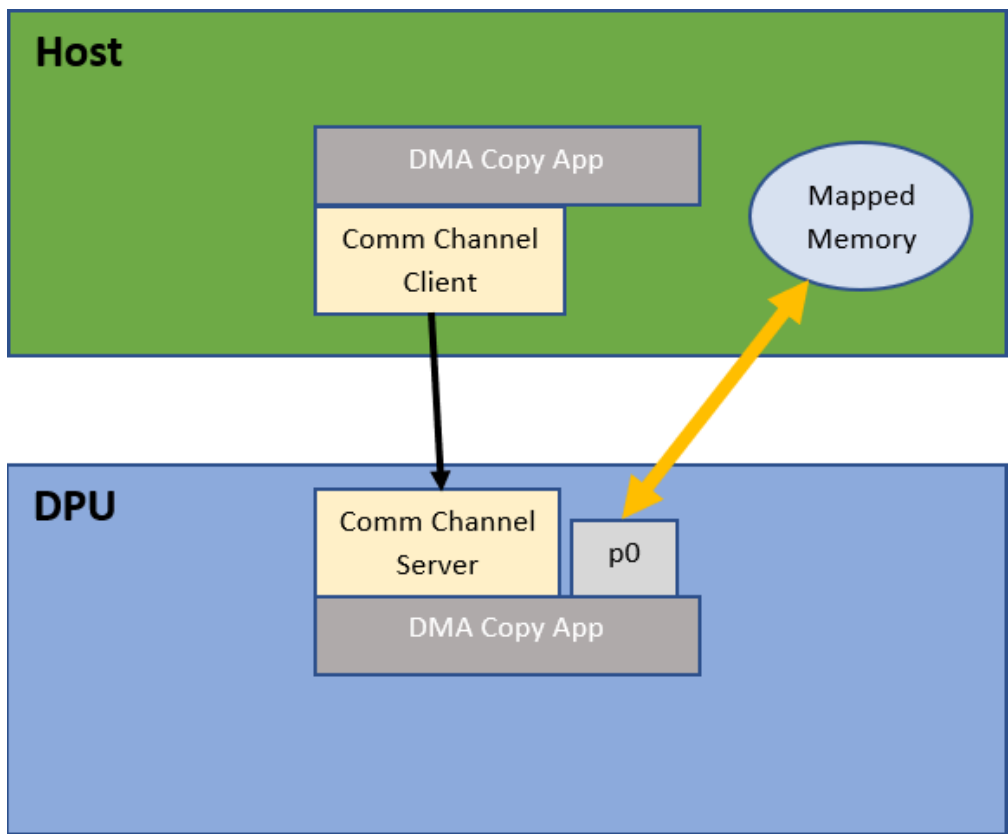
Introduction

DOCA DMA (direct memory access) Copy application transfers files (data path), up to the maximum supported size by the hardware, between the DPU and the x86 host using the [DOCA DMA Library](#) which provides an API to copy data between DOCA buffers using hardware acceleration, supporting both local and remote memory.

DOCA DMA allows complex memory copy operations to be easily executed in an optimized, hardware-accelerated manner.

System Design

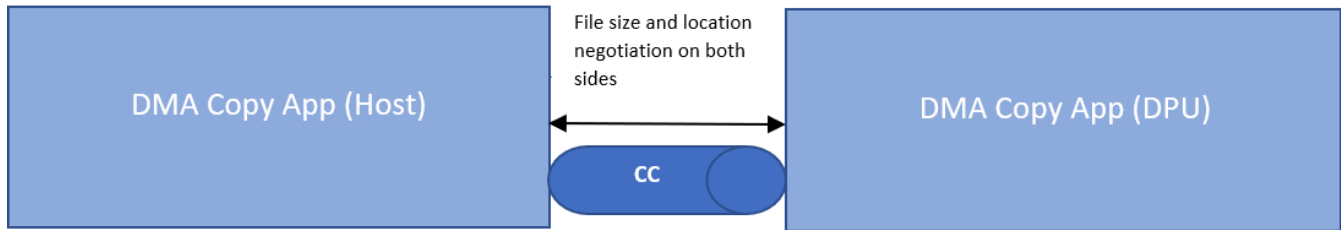
DOCA DMA Copy is designed to run on the instances of the BlueField DPU and x86 host. The DPU application must be the first to spawn as it opens the [DOCA Comch](#) server between the two sides on which all the necessary DOCA DMA library configuration files (control path) are transferred.



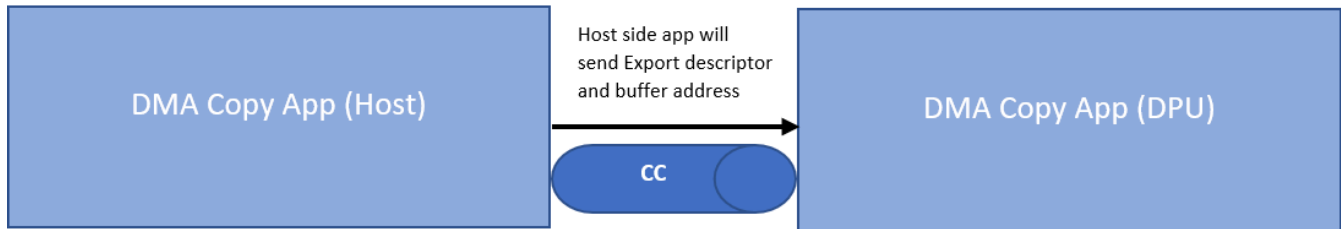
Application Architecture

DOCA DMA Copy runs on top of DOCA DMA to read/write directly from the host's memory without any user/kernel space context switches, allowing for a fast memory copy.

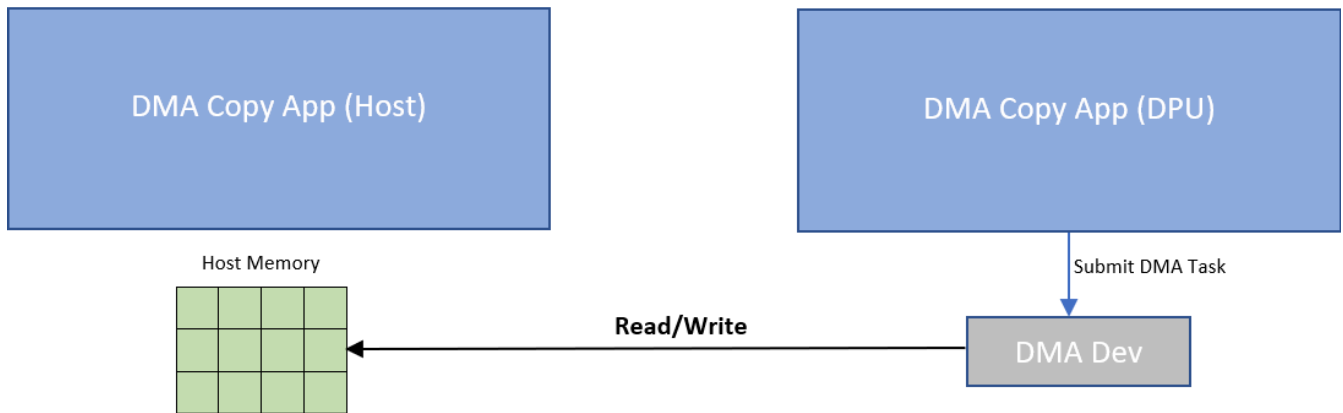
First stage



Second stage



Third stage



Flow:

1. The two sides initiate a short negotiation in which the file size and location are determined.
2. The host side creates the export descriptor with `doca_mmap_export_pci()` and sends it with the local buffer address and length on the Comch to the DPU side application.
3. The DPU side application uses the received export descriptor to create a remote memory map locally with `doca_mmap_create_from_export()` and the host buffer information to create a remote DOCA buffer.

4. From this point on, the DPU side application has all the necessary memory information and the DMA copy can take place.

DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA DMA](#)
- [DOCA Comch](#)

Refer to their respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/dma_copy/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_dma_copy is created under /tmp/build/dma_copy/.

Compiling Only the Current Application

To directly build only the DMA Copy application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_dma_copy=true  
ninja -C /tmp/build
```

Info

doca_dma_copy is created under /tmp/build/dma_copy/.

Alternatively, one can set the desired flags in the `meson_options.txt` file instead of providing them in the compilation command line:

1. Edit the following flags in `/opt/mellanox/doca/applications/meson_options.txt`:

- Set `enable_all_applications` to `false`
- Set `enable_dma_copy` to `true`

2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

`doca_dma_copy` is created under `/tmp/build/dma_copy/`.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Application Execution

The DMA Copy application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

Usage: `doca_dma_copy` [DOCA Flags] [Program Flags]

DOCA Flags:

`-h, --help` Print a help synopsis

`-v, --version` Print program version information

`-l, --log-level` Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

`--sdk-log-level` Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

`-j, --json <path>` Parse all command flags from an input json file

Program Flags:

`-f, --file` Full path to file to be copied/created after a successful DMA copy

`-p, --pci-addr` DOCA Comm Channel device PCI address

`-r, --rep-pci` DOCA Comm Channel device representor PCI address (needed only on DPU)

Info

This usage printout can be printed to the command line using the `-h` (or `--help`) options:

```
./doca_dma_copy -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the BlueField:

```
./doca_dma_copy -p 03:00.0 -r 3b:00.0 -f received.txt
```

Note

Both the DOCA Comch device PCIe address (03:00.0) and the DOCA Comch device representor PCIe address (3b:00.0) should match the addresses of the desired PCIe devices.

3. CLI example for running the application on the host:

```
./doca_dma_copy -p 3b:00.0 -f send.txt
```

Note

The DOCA Comch device PCIe address, 3b:00.0, should match the address of the desired PCIe device.

4. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_dma_copy --json [json_file]
```

For example:

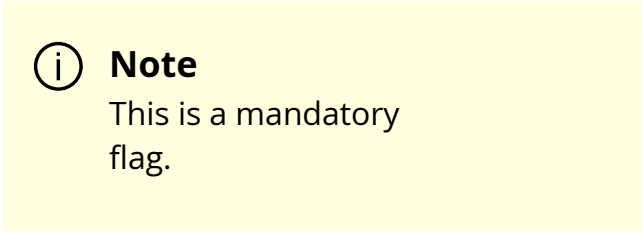

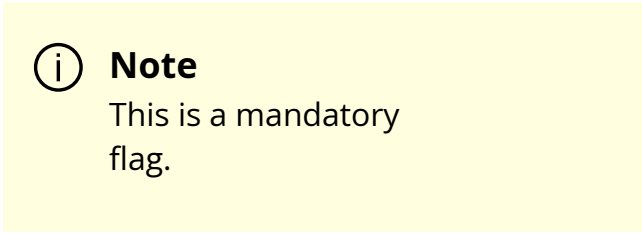
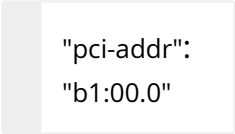
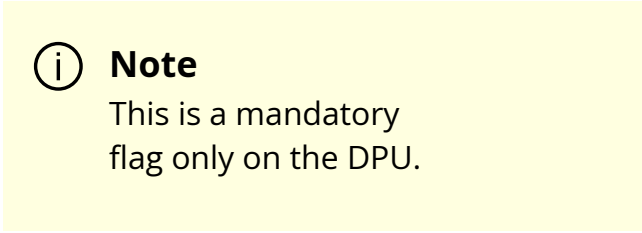
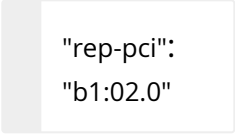
```
./doca_dma_copy --json ./dma_copy_params.json
```


Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Print a help synopsis	N/A
	v	version	Print program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none">• DISABLE=10• CRITICAL=20• ERROR=30• WARNING=40• INFO=50• DEBUG=60• TRACE=70 (requires compilation with TRACE log level support)	<pre>"log-level": 60</pre>
	N/A	sdk-log-level	Set the log level for the program: <ul style="list-style-type: none">• DISABLE=10• CRITICAL=20• ERROR=30• WARNING=40• INFO=50• DEBUG=60• TRACE=70	<pre>"sdk-log- level": 40</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	j	json	Parse all command flags from an input JSON file	N/A
Program flags	f	file	Full path to file to be copied/created after a successful copy 	
	p	pci-addr	DOCA Comch device PCIe address. 	
	r	rep-pci	DOCA Comch device representor PCIe address. 	

 **Info**

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.

Application Code Flow

1. Parse application argument.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register DMA Copy application parameters.

```
register_dma_copy_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

1. Initialize Comch endpoint.

```
init_cc();
```

1. Create Comch endpoint.

2. Parse user PCIe address for Comch device.
3. Open Comch DOCA device.
4. Parse user PCIe address for Comch device representor (on DPU side).
5. Open Comch DOCA device representor (on DPU side).
6. Set Comch endpoint properties.

2. Open the DOCA hardware device from which the copy would be made.

```
open_dma_device();
```

1. Parse the PCIe address provided by the user.
2. Create a list of all available DOCA devices.
3. Find the appropriate DOCA device according to specific properties.
4. Open the device.

3. Create all required DOCA core objects.

```
create_core_objects();
```

4. Initiate DOCA core objects.

```
init_core_objects();
```

5. Start host/DPU DMA Copy.

1. Host side application:

```
host_start_dma_copy();
```

1. Start negotiation with the DPU side application for the location and size of the file.

2. Allocate memory for the DMA buffer.
3. Export the memory map and send the output (export descriptor) to the DPU side application.
4. Send the host local buffer memory address and length on the Comch to the DPU side application.
5. Wait for the DPU to notify that DMA Copy ended.
6. Close all memory objects.
7. Clean resources.

2. DPU side application:

```
dpu_start_dma_copy();
```

1. Start negotiation with the host side application for file location and size.
2. Allocate memory for the DMA buffer.
3. Receive the export descriptor on the Comch.
4. Create the DOCA memory map for the remote buffer on the host.
5. Receive the host buffer information on the Comch.
6. Create two DOCA buffers, one for the remote (host) buffer and one for the local buffer.
7. Submit the DMA copy task.
8. Send a host message to notify that DMA copy ended.
9. Clean resources.

6. Destroy Comch.

```
destroy_cc();
```

7. Destroy DOCA core objects.

```
destroy_core_objects();
```

8. Arg parser destroy.

```
doca_argp_destroy();
```

References

- /opt/mellanox/doca/applications/dma_copy/
- /opt/mellanox/doca/applications/dma_copy/dma_copy_params.json

NVIDIA DOCA DPA All-to-all Application Guide

This guide explains all-to-all collective operation example when accelerated using the DPA in NVIDIA® BlueField®-3 DPU.

Introduction

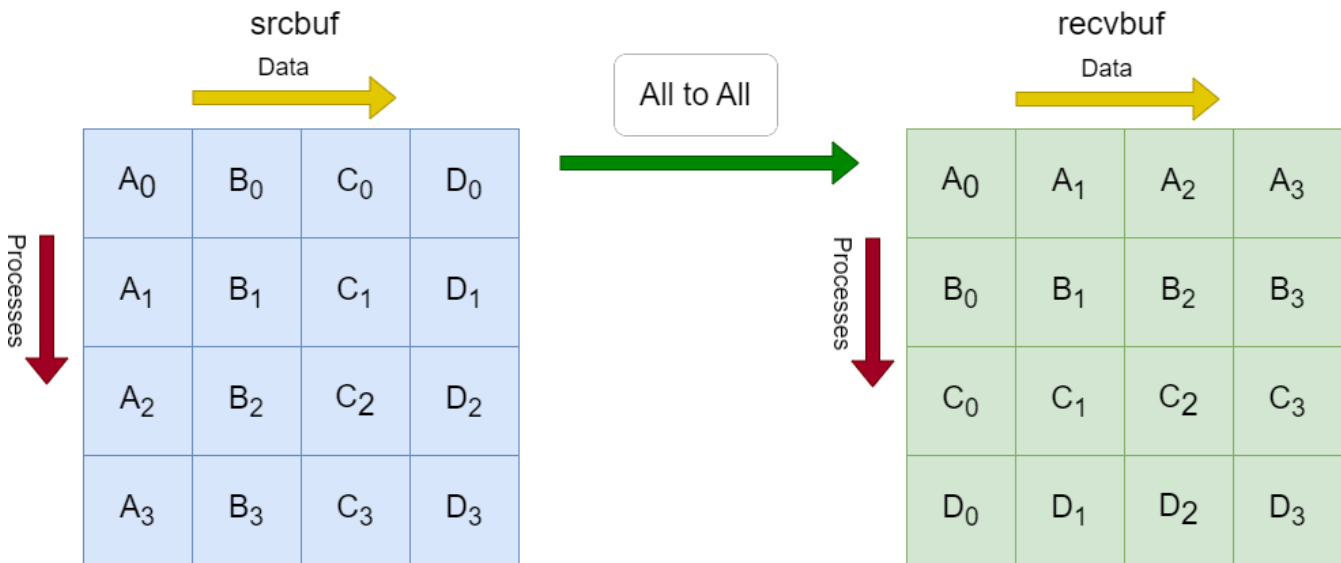
This reference application shows how the message passing interface (MPI) all-to-all collective can be accelerated on the Data Path Accelerator (DPA). In an MPI collective, all processes in the same job call the collective routine.

Given a communicator of n ranks, the application performs a collective operation in which all processes send and receive the same amount of data from all processes (hence all-to-all).

This document describes how to run the all-to-all example using the [DOCA DPA API](#).

System Design

All-to-all is an MPI method. MPI is a standardized and portable message passing standard designed to function on parallel computing architectures. An MPI program is one where several processes run in parallel.

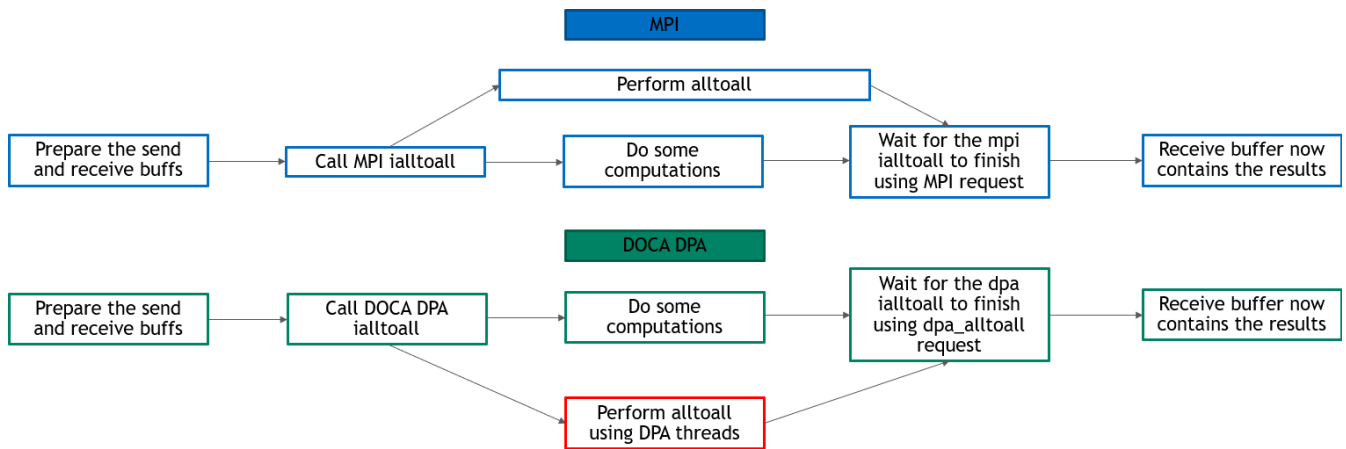


Each process in the diagram divides its local sendbuf into n blocks (4 in this example), each containing sendcount elements (4 in this example). Process i sends the k -th block of its local sendbuf to process k which places the data in the i -th block of its local recvbuf.

Implementing all-to-all method using DOCA DPA offloads the copying of the elements from the srcbuf to the recvbufs to the DPA, and leaves the CPU free to perform other computations.

Application Architecture

The following diagram describes the differences between host-based all-to-all and DPA all-to-all.



- In DPA all-to-all, DPA threads perform all-to-all and the CPU is free to do other computations
- In host-based all-to-all, CPU must still perform all-to-all at some point and is not completely free for other computations

DOCA Libraries

This application leverages the following DOCA library:

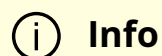
- [DOCA DPA](#)

Refer to its programming guide for more information.

Dependencies

- NVIDIA BlueField-3 platform is required
- The application can be run on target BlueField or on host.
- Open MPI version 4.1.5rc2 or greater (included in DOCA's installation).

Compiling the Application



Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/dpa_all_to_all/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

`doca_dpa_all_to_all` is created under `/tmp/build/dpa_all_to_all/`.

Compiling DPA All-to-all Application Only

To directly build only all-to-all application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_dpa_all_to_all=true  
ninja -C /tmp/build
```

Info

doca_dpa_all_to_all is created under /tmp/build/dpa_all_to_all/.

Alternatively, one can set the desired flags in meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:

- Set enable_all_applications to false
- Set enable_dpa_all_to_all to true

2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_dpa_all_to_all is created under /tmp/build/dpa_all_to_all/.

Troubleshooting

Please refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Prerequisites

MPI is used for compilation and running of this application. Make sure that MPI is installed on your setup (openmpi is provided as part of the installation of doca-tools).

Note

The installation also requires updating the LD_LIBRARY_PATH and PATH environment variable to include MPI. For example, if openmpi is installed under /usr/mpi/gcc/openmpi-4.1.7a1 then updating the environment variables should be like this:

```
export PATH=/usr/mpi/gcc/openmpi-4.1.7a1/bin:${PATH}
export LD_LIBRARY_PATH=/usr/mpi/gcc/openmpi-
4.1.7a1/lib:${LD_LIBRARY_PATH}
```

Application Execution

DPA all-to-all application is provided in source form. Therefore, a compilation is required before application can be executed.

1. Application usage instructions:

Usage: `doca_dpa_all_to_all` [DOCA Flags] [Program Flags]

DOCA Flags:

`-h, --help` Print a help synopsis

`-v, --version` Print program version information

`-l, --log-level` Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

`--sdk-log-level` Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

`-j, --json <path>` Parse all command flags from an input json file

Program Flags:

`-m, --msgsize <Message size>` The message size - the size of the sendbuf and recvbuf (in bytes). Must be in multiplies of integer size. Default is size of one integer times the number of processes.

`-d, --devices <IB device names>` IB devices names that supports DPA, separated by comma without spaces (max of two devices). If not provided then a random IB device will be chosen.

Info

This usage printout can be printed to the command line using the `-h` (or `--help`) option:

```
./doca_dpa_all_to_all -h
```

Info

For additional information, please refer to section "[Command Line Flags](#)".

2. CLI example for running the application on host:

Note

This is an MPI program, so use `mpirun` to run the application (with the `-np` flag to specify the number of processes to run).

- The following runs the DPA all-to-all application with 8 processes using the default message size (the number of processes, which is 8, times the size of 1 integer) with a random InfiniBand device:

```
mpirun -np 8 ./doca_dpa_all_to_all
```

- The following runs DPA all-to-all application with 8 processes, with 128 bytes as message size, and with `mlx5_0` and `mlx5_1` as the InfiniBand devices:

```
mpirun -np 8 ./doca_dpa_all_to_all -m 128 -d "mlx5_0,mlx5_1"
```

Note

The application supports running with a maximum of 16 processes. If you try to run with more processes, an error is printed and the application exits.

3. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_dpa_all_to_all --json [json_file]
```

For example:

```
./doca_dpa_all_to_all --json ./dpa_all_to_all_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, especially the InfiniBand device identifiers.

Command Line Flags

Flag Type	Short Flag	Long Flag / JSON Key	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none">• DISABLE=10• CRITICAL=20• ERROR=30• WARNING=40• INFO=50• DEBUG=60	<pre>"log-level": 60</pre>

Flag Type	Short Flag	Long Flag /JSON Key	Description	JSON Content
			<ul style="list-style-type: none"> TRACE=70 (requires compilation with TRACE log level support) 	
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> DISABLE=10 CRITICAL=20 ERROR=30 WARNING=40 INFO=50 DEBUG=60 TRACE=70 	<pre>"sdk-log-level": 40</pre>
	j	json	Parse all command flags from an input json file	N/A
Program flags	m	msgsize	The message size. The size of the sendbuf and recvbuf (in bytes). Must be in multiples of an integer. The default is size of 1 integer times the number of processes.	<pre>"msgsize": -1</pre> <div style="background-color: #ffffcc; padding: 10px; margin-top: 10px;"> <p>Note The value -1 is a placeholder to use the default size, which is only known at run time (because it depends on the number of processes).</p> </div>

Flag Type	Short Flag	Long Flag /JSON Key	Description	JSON Content
	d	devices	InfiniBand devices names that support DPA, separated by comma without spaces (max of two devices). If NOT_SET then a random InfiniBand device is chosen.	<pre>"devices": "NOT_SET"</pre>

Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications .

Application Code Flow

1. Initialize MPI.

```
MPI_Init(&argc, &argv);
```

2. Parse application arguments.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register the application's parameters.

```
register_all_to_all_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

1. The `msgsize` parameter is the size of the `sendbuf` and `recvbuf` (in bytes). It must be in multiples of an integer and at least the number of processes times an integer size.
2. The `devices_param` parameter is the names of the InfiniBand devices to use (must support DPA). It can include up to two devices names.

4. Only let the first process (of rank 0) parse the parameters to then broadcast them to the rest of the processes.

3. Check and prepare the needed resources for the `all_to_all` call:

1. Check the number of processes (maximum is 16).
2. Check the `msgsize`. It must be in multiples of integer size and at least the number of processes times integer size.
3. Allocate the `sendbuf` and `recvbuf` according to `msgsize`.

4. Prepare the resources required to perform all-to-all method using DOCA DPA:

1. Initialize DOCA DPA context:

1. Open DOCA DPA device (DOCA device that supports DPA).

```
open_dpa_device(&doca_device);
```

2. Initialize DOCA DPA context using the opened device.

```
extern struct doca_dpa_app *dpa_all2all_app;

doca_dpa_create(doca_device, &doca_dpa);

doca_dpa_set_app(doca_dpa, dpa_all2all_app);

doca_dpa_start(doca_dpa);
```

2. Initialize the required [DOCA Sync Events](#) for the all-to-all:

1. One completion event for the kernel launch where the subscriber is CPU and the publisher is DPA.
2. Kernel events, published by remote peer and subscribed to by DPA, as the number of processes.

```
create_dpa_a2a_events() {
    // initialize completion event
    doca_sync_event_create(&comp_event);

    doca_sync_event_add_publisher_location_dpa(comp_event);

    doca_sync_event_add_subscriber_location_cpu(comp_event);

    doca_sync_event_start(comp_event);
    // initialize kernels events
    for (i = 0; i < resources->num_ranks; i++) {
        doca_sync_event_create(&(kernel_events[i]));

        doca_sync_event_add_publisher_location_remote_net(kernel_events[i]);

        doca_sync_event_add_subscriber_location_dpa(kernel_events[i]);

        doca_sync_event_start(kernel_events[i]);
    }
}
```

```
}  
}
```

3. Prepare DOCA RDMA and set them to work on DPA:

1. Create DOCA RDMA as the number of processes/ranks.

```
for (i = 0; i < resources->num_ranks; i++) {  
    doca_rdma_create(&rdma);  
  
    rdma_as_doca_ctx = doca_rdma_as_ctx(rdma);  
  
    doca_rdma_set_permissions(rdma);  
  
    doca_rdma_set_grh_enabled(rdma);  
  
    doca_ctx_set_datapath_on_dpa(rdma_as_doca_ctx, doca_dpa);  
  
    doca_ctx_start(rdma_as_doca_ctx);  
}
```

2. Connect local DOCA RDMA to the remote DOCA RDMA.

```
connect_dpa_a2a_rdmads();
```

3. Get DPA handles for local DOCA RDMA (so they can be used by DPA kernel) and copy them to DPA heap memory.

```
for (int i = 0; i < resources->num_ranks; i++) {  
    doca_rdma_get_dpa_handle(rdmads[i], &(rdma_handles[i]));  
}  
  
doca_dpa_mem_alloc(&dev_ptr_rdma_handles);  
  
doca_dpa_h2d_memcpy(dev_ptr_rdma_handles, rdma_handles);
```


4. Prepare the memory required to perform all-to-all method using DOCA Mmap. This includes creating DPA memory handles for sendbuf and recvbuf, getting other processes recvbufs handles, and copying these memory handles and their remote keys and events handlers to DPA heap memory.

```
prepare_dpa_a2a_memory();
```

5. Launch `alltoall_kernel` using DOCA DPA kernel launch with all required parameters:

1. Every MPI rank launches a kernel of up to `MAX_NUM_THREADS`. This example defines `MAX_NUM_THREADS` as 16.
2. Launch `alltoall_kernel` using `kernel_launch`.

```
doca_dpa_kernel_launch_update_set();
```

3. Each process should perform `num_ranks` RDMA write operations, with local and remote buffers calculated based on the rank of the process that is performing the RDMA write operation and the rank of the remote process that is being written to. The application iterates over the rank of the remote process.*i*

Each process runs `num_threads` threads on this kernel, therefore the number of RDMA write operations (which is the number of processes) is divided by the number of threads.

Each thread should wait on its local events to make sure that the remote processes have finished RDMA write operations.

Each thread should also synchronize its RDMA DPA handles to make sure that the local RDMA operation calls has finished.

```
for (i = thread_rank; i < num_ranks; i += num_threads) {  
    doca_dpa_dev_rdma_post_write();  
    doca_dpa_dev_rdma_signal_set();  
}  
  
for (i = thread_rank; i < num_ranks; i += num_threads) {  
    doca_dpa_dev_sync_event_wait_gt();  
    doca_dpa_dev_rdma_synchronize();  
}
```

```
}
```

4. Wait until `alltoall_kernel` has finished.

```
doca_sync_event_wait_gt();
```

Note

Add an MPI barrier after waiting for the event to make sure that all of the processes have finished executing `alltoall_kernel`.

```
MPI_Barrier();
```

After `alltoall_kernel` is finished, the `recvbuf` of all processes contains the expected output of all-to-all method.

6. Destroy `a2a_resources`:

1. Free all DOCA DPA memories.

```
doca_dpa_mem_free();
```

2. Destroy all DOCA Mmaps

```
doca_mmap_destroy();
```

3. Destroy all DOCA RDMA.

```
doca_ctx_stop();  
doca_rdma_destroy();
```

4. Destroy all DOCA Sync Events.

```
doca_sync_event_destroy();
```

5. Destroy DOCA DPA context.

```
doca_dpa_destroy();
```

6. Close DOCA device.

```
doca_dev_close();
```

References

- /opt/mellanox/doca/applications/dpa_all_to_all/
- /opt/mellanox/doca/applications/dpa_all_to_all/dpa_all_to_all_params.json

NVIDIA DOCA DPA L2 Reflector Application Guide

This document provides a DPA L2 reflector implementation on top of the NVIDIA® BlueField®-3 DPU.

Introduction

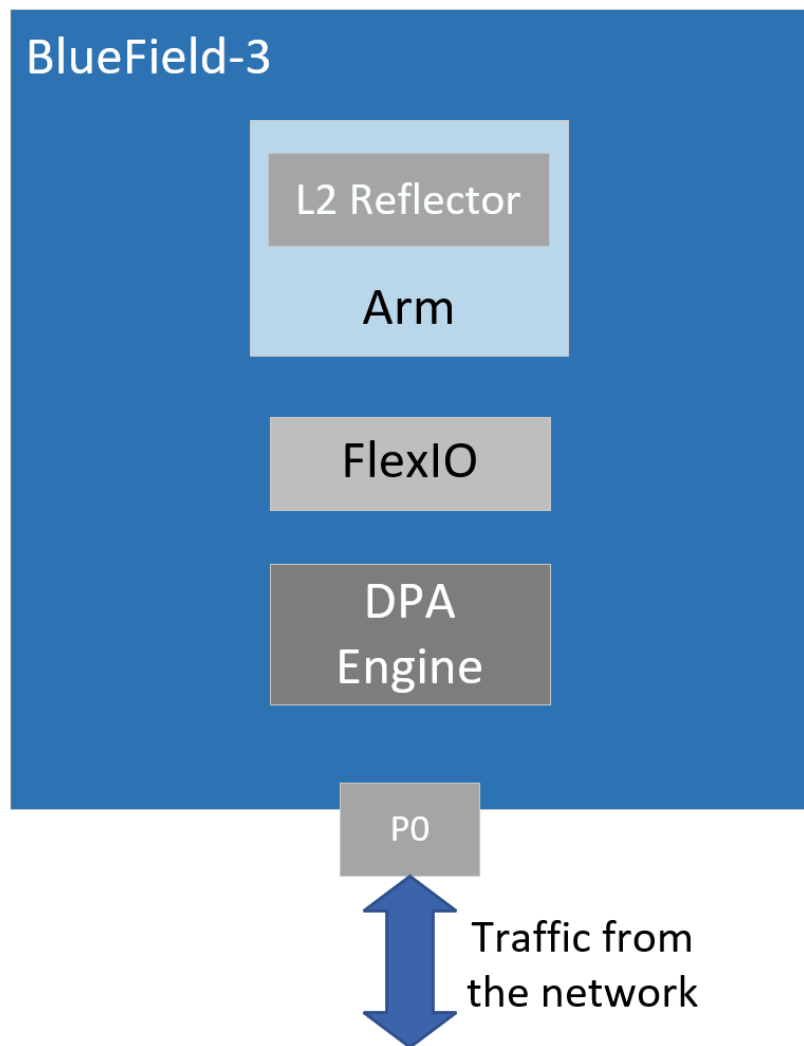
The BlueField-3 DPU supports high-speed Data Path Accelerator (DPA) . Data path accelerator allows for accelerated packet processing and manipulation.

DOCA Layer-2 reflector uses the DPA engine to intercept network traffic and swap the source and destination MAC addresses of each packet.

System Design

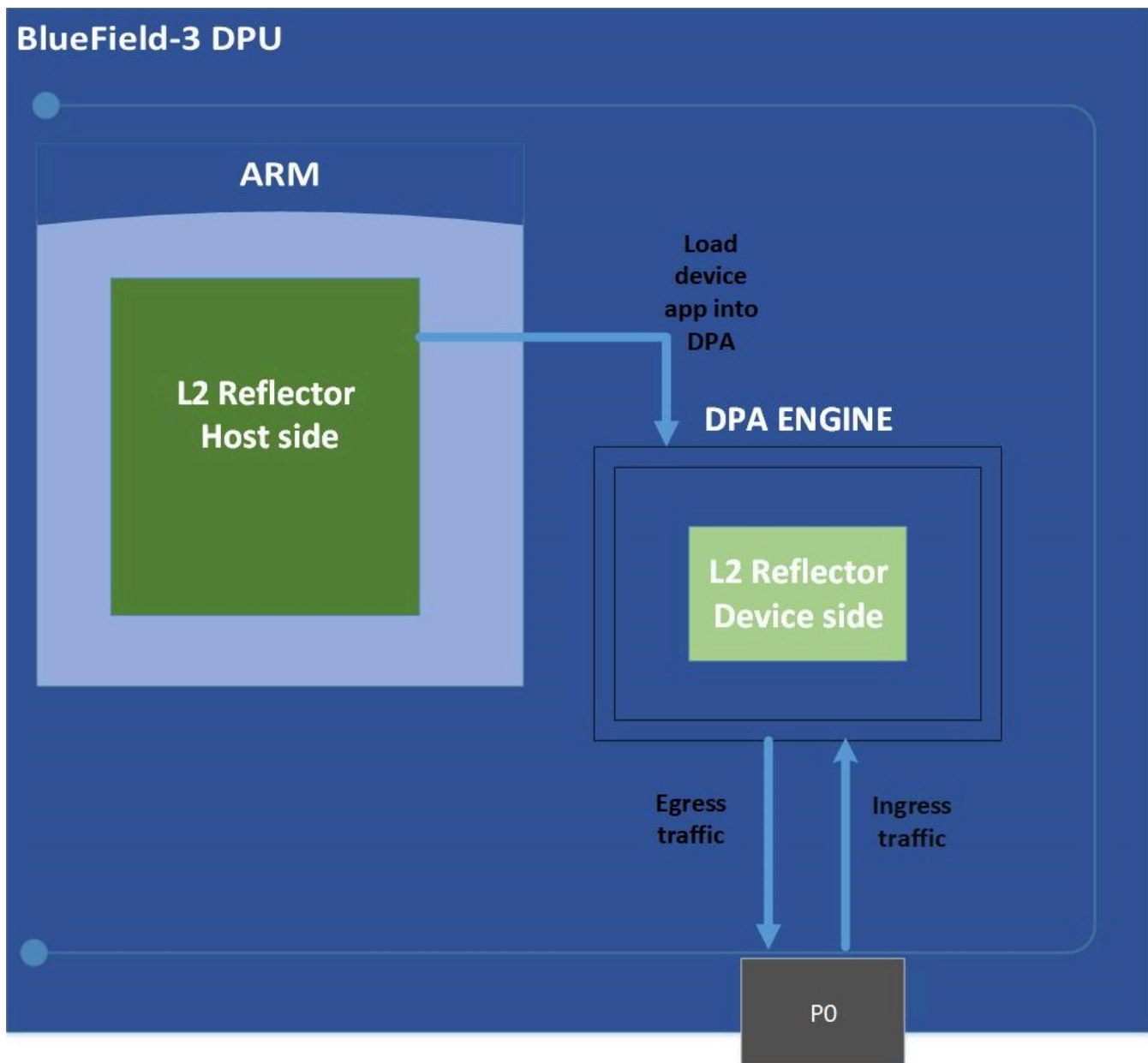
The application accepts traffic from a specific port given as an argument and leverages DPA capabilities for accelerated processing.

The following figure provides a high-level view of the components of the application:



Application Architecture

DOCA L2 reflector runs on top of FlexIO SDK to configure the DPA engine.



The FlexIO application consist of two parts:

- Host side - responsible for allocating resources and loading them to the DPA
- Device side - core processing logic of the application which swaps the MACs on the DPA

For more information, refer to "[Programming FlexIO SDK](#)".

DOCA Libraries and Drivers

This application leverages the following DOCA driver:

- [FlexIO SDK](#)

Refer to its programming guide for more information.

Dependencies

NVIDIA® BlueField®-3 DPU and above is required.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/l2_reflector/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

l2_reflector is created under /tmp/build/l2_reflector/host.

Compiling DPA L2 Reflector Application Only

To directly build only the L2 reflector application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_l2_reflector=true  
ninja -C /tmp/build
```

Info

l2_reflector is created under /tmp/build/l2_reflector/host.

Alternatively, one can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
 - o Set enable_all_applications to false

- o Set `enable_l2_reflector` to true

2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

`l2_reflector` is created under `/tmp/build/l2_reflector/host`.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue you may encounter with the compilation of the DOCA applications.

Running the Application

Application Execution

The L2 reflector application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: l2_reflector [DOCA Flags] [Program Flags]
```

DOCA Flags:

```
-h, --help Print a help synopsis
```


-v, --version Print program version information
-l, --log-level Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
-j, --json <path> Parse all command flags from an input json file

Program Flags:

-d, --device <device name> Device name

Info

This usage printout can be printed to the command line using the -h (or --help) options:

```
./l2_reflector -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on BlueField or host:

```
./l2_reflector -d mlx5_0
```

Note

The used device name (-d flag) must match the identifier of the desired IB device.

(i) Info

To run the application on the second port, verify that it has a partition. Run:

```
dpaeumgmt partition info -d mlx5_1
```

If DPA EU partition creation is required, refer to [NVIDIA DOCA DPA Execution Unit Management Tool](#).

3. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./l2_reflector --json [json_file]
```

For example:

```
./l2_reflector --json ./l2_reflector_params.json
```

(i) Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the desired PCIe addresses required for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with TRACE log level support) 	<pre>"log-level": 60</pre>
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	<pre>"sdk-log- level": 40</pre>
	j	json	Parse all command flags from an input JSON file	N/A
Program flags	d	device	Device name	<pre>"device": mlx5_0</pre>

i Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Note

DPA L2 reflector works with packets with a specific source MAC address. To check the supported MAC address, refer to `/opt/mellanox/doca/applications/l2_reflector/src/host/l2_reflector_core.h`.

Please refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.

Application Code Flow

This section lists the application's configuration flow which includes different FlexIO functions and wrappers.

1. Parse application argument.
 1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register the application's parameters.

```
register_l2_reflector_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

2. Setup the InfiniBand device.

```
l2_reflector_setup_ibv_device();
```

3. Setup the DPA device.

```
l2_reflector_setup_device();
```

4. Allocate the device's resources.

```
l2_reflector_allocate_device_resources();
```

5. Run initialization function on the device.

```
flexio_process_call();
```

6. Create the steering rule.

```
l2_reflector_create_steering_rule();
```

7. Start the event handler on the device.

```
flexio_event_handler_run();
```

8. Main loop.

```
while (!force_quit)  
    sleep(10);
```

9. Cleanup the resources.

```
l2_reflector_destroy();
```

References

- /opt/mellanox/doca/applications/l2_reflector/
- /opt/mellanox/doca/applications/l2_reflector/l2_reflector_params.json

NVIDIA DOCA East-West Overlay Encryption Application

This guide describes IPsec-based strongSwan solution on top of NVIDIA® BlueField® DPU.

Note

If your target application utilizes 100Gb/s or higher bandwidth, where a substantial part of the bandwidth is allocated for IPsec traffic, please refer to the *NVIDIA BlueField-2 DPUs Product Release Notes* to learn about a potential bandwidth limitation. To access the relevant product release notes, please contact your NVIDIA sales representative.

Introduction

IPsec is used to set up encrypted connections between different devices. It helps keep data sent over public networks secure. IPsec is often used to set up VPNs, and it works by encrypting IP packets as well as authenticating the packets' originator.

IPsec contains the following main modules:

- Key exchange – a key is a string of random bytes that can be used for encryption and decryption of messages. IPsec sets up keys with a key exchange between the connected devices, so that each device can decrypt the other device's messages.
- Authentication – IPsec provides authentication for each packet which ensures that they come from a trusted source.
- Encryption – IPsec encrypts the payloads within each packet and possibly, based on the transport mode, the packet's IP header.
- Decryption – at the other end of the communication, packets are decrypted by the IPsec supported node.

IPsec supports two types of headers:

- Authentication header (AH) – AH protocol ensures that packets are from a trusted source. AH does not provide any encryption.
- Encapsulating security protocol (ESP) – ESP encrypts the payload for each packet as well as the IP header depending on the transport mode. ESP adds its own header and a trailer to each data packet.

IPsec support two types of transport mode:

- IPsec tunnel mode – used between two network nodes, each acting as tunnel initiator/terminator on a public network. In this mode, the original IP header and payload are both encrypted. Since the IP header is encrypted, an IP tunnel is added for network forwarding. At each end of the tunnel, the routers decrypt the IP headers to route the packets to their destinations.
- Transport mode – the payload of each packet is encrypted, but the original IP header is not. Intermediary network nodes are therefore able to view the destination of each packet and route the packet, unless a separate tunneling protocol is used.

strongSwan is an open-source IPsec-based VPN solution. For more information, refer to [strongSwan documentation](#).

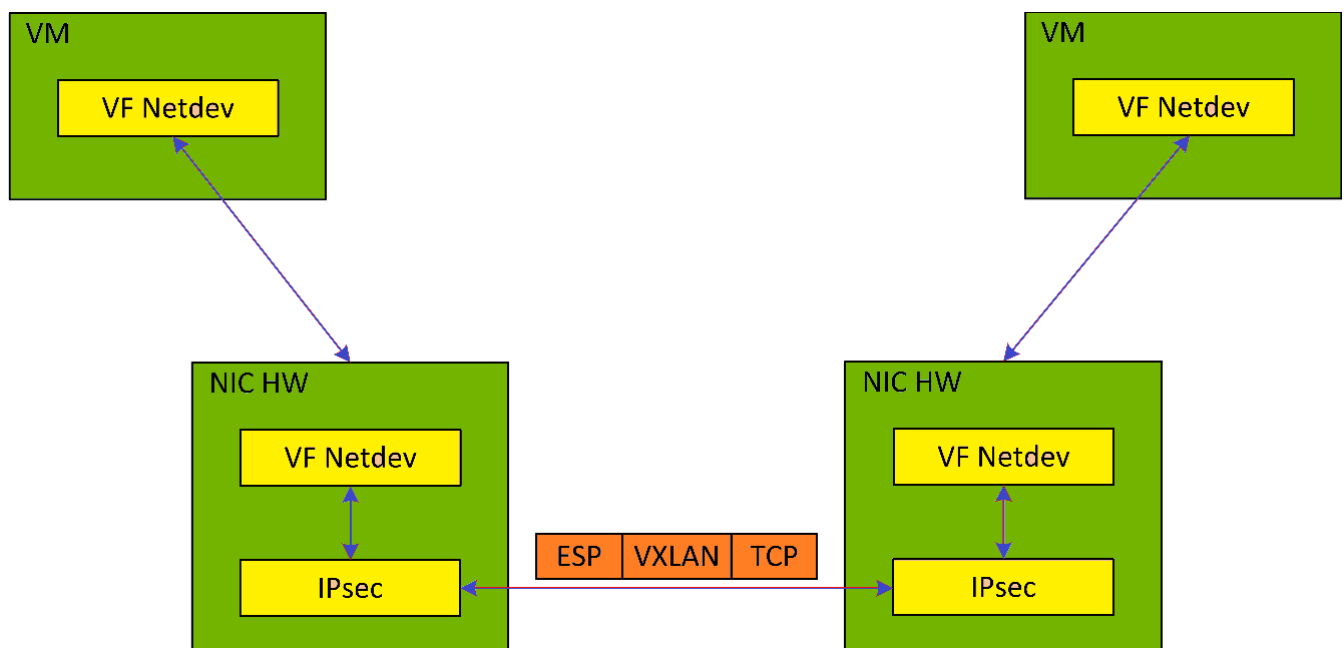
System Design

IPsec packet offload offloads both IPsec crypto (encrypt/decrypt) and IPsec encapsulation to the hardware.

The deployment model allows the IPsec offload to be transparent to the host with the benefits of securing legacy workloads (no dependency on host SW stack) and to zero CPU utilization on host.

IPsec packet offload configuration works with and is transparent to OVS offload. This means all packets from OVS offload are encrypted by IPsec rules.

The following figure illustrates the interaction between IPsec packet offload and OVS VXLAN offload.



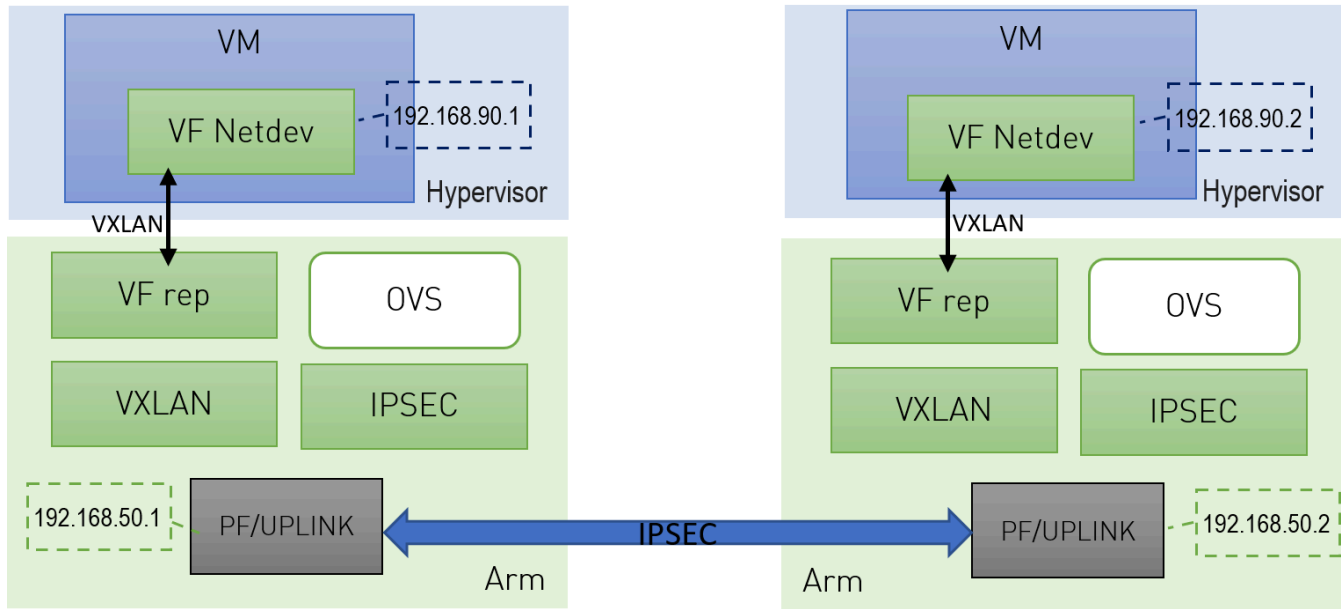
Note

IPsec packet offload is only supported on Ubuntu Bluefield kernel 5.15

Note

OVS offload and IPsec IPv6 do not work together.

Application Architecture



1. Configure strongSwan IPsec offload using `swanctl.conf` configuration file.
2. Traffic is sent from the host through BlueField.
3. Using OVS, the packets are encapsulated on ingress using tunnel protocols (VXLAN for example) to match IPsec configuration by strongSwan.
4. Set by strongSwan configuration file, traffic will be encrypted using the hardware offload.
5. Egress flow is decryption first, decapsulation of the tunnel header and forward to the relevant physical function.

DOCA Libraries

N/A

Configuration Flow

The following section provides information on manually configuring IPsec packet offload in general and on using OVS IPsec with strongSwan specifically.

Note

There is a script, `east_west_overlay_encryption.sh` which performs the steps in this section automatically.

If you are working directly with the `ip xfrm` tool, use `/opt/mellanox/iproute2/sbin/ip` to benefit from IPsec packet offload support.

There are two parts in the configuration flow

1. Enabling IPsec packet offload mode.
2. Configuring the IPsec OVS bridge using one of three modes of authentication.

Note

An alternative for step two is configuring `swanctl.conf` files (configuration files for strongSwan) manually and using strongSwan directly instead of using IPsec OVS (which automatically generates `swanctl.conf` files) as explained in section "[Configuring OVS IPsec Using strongSwan Manually](#)".

Enabling IPsec Packet Offload

This section explicitly enables IPsec packet offload on the Arm cores before setting up offload-aware IPsec tunnels.

Note

If an OVS VXLAN tunnel configuration already exists, stop `openvswitch` service prior to performing the steps below and restart the service afterwards.

Explicitly enable IPsec full offload on the Arm cores.

1. Set `IPSEC_FULL_OFFLOAD="yes"` in `/etc/mellanox/mlnx-bf.conf` .

Note

If `IPSEC_FULL_OFFLOAD` does not appear in `/etc/mellanox/mlnx-bf.conf` then you are probably using an old version of the BlueField image. Check the way of enabling IPsec full offload in a previous DOCA versions in the [NVIDIA DOCA Documentation Archives](#).

2. Restart IB driver (rebooting also works). Run:

```
/etc/init.d/openibd restart
```

Note

If `mlx-regex` is running:

1. Disable `mlx-regex` prior to running restarting the IB driver:

```
systemctl stop mlx-regex
```

2. Restart IB driver according to the command above.
3. Re-enable `mlx-regex` after the restart has finished:

```
systemctl restart mlx-regex
```

(i) Note

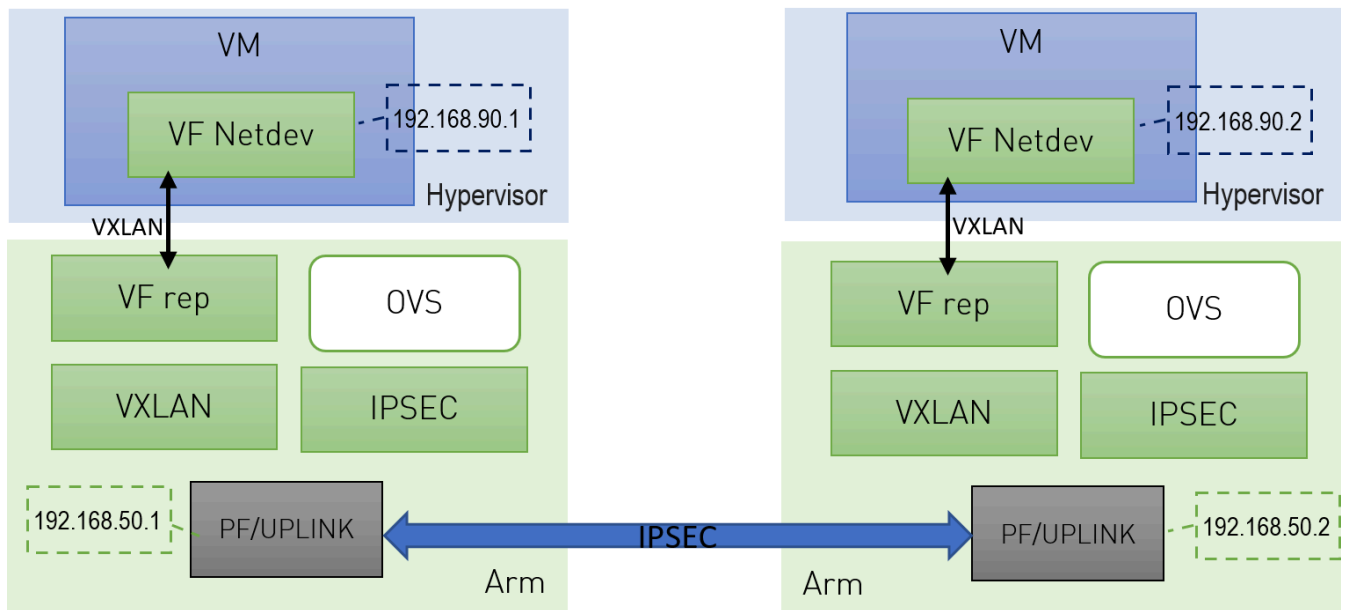
To revert IPsec full offload mode, redo the procedure from step 1, only difference is to set `IPSEC_FULL_OFFLOAD="no"` in `/etc/mellanox/mlnx-bf.conf`.

Configuring OVS IPsec

(i) Note

Before proceeding with this section, make sure to follow the procedure in section "[Enabling IPsec Packet Offload](#)" for both DPUs.

This section configures OVS IPsec VXLAN tunnel which automatically generates the `swanctl.conf` files and runs `strongSwan` (the IPsec daemon). The following figure illustrates an example with two BlueField DPUs, Left and Right, operating with a secured VXLAN channel.



Two BlueField DPUs are required to build an OVS IPsec tunnel between the two hosts, Right and Left.

The OVS IPsec tunnel configures an unaware IPsec connection between the two hosts' InfiniBand devices. For the sake of this example, the host's InfiniBand network device is HOST_PF, and the DPU's host representor is PF_REP and the DPU's physical function PF.

This example sets up the following variables on both Arms:

```
# host_ip1=1.1.1.1
# host_ip2=1.1.1.2
# HOST_PF=ens7np0
# ip1=192.168.50.1
# ip2=192.168.50.2
# PF=p0
# PF_REP=pf0hpf
```

Note

The name of the HOST_PF could be different in your machine. You may verify this by running:

```
host# ibdev2netdev
mlx5_0 port 1 ==> ens7np0 (Down)
mlx5_1 port 1 ==> ens8np1 (Down)
```

This example uses the first InfiniBand's (mlx5_0) network device which is ens7np0.

1. Configure IP addresses for the HOST_PFs of both hosts (x86):

1. On host_1:

```
# ifconfig $HOST_PF $host_ip1/24 up
```

2. On host_2:

```
# ifconfig $HOST_PF $host_ip2/24 up
```

Note

Step 1 is the only command that is performed on the host, the rest of the commands are performed on the Arm (DPU) side.

2. Configure IP addresses for the PFs of both Arms:

1. On Arm_1:

```
# ifconfig $PF $ip1/24 up
```

2. On Arm_2:

```
# ifconfig $PF $ip2/24 up
```

3. Start Open vSwitch. If your operating system is Ubuntu, run the following on both Arm_1 and Arm_2:

```
# service openvswitch-switch start
```

If your operating system is CentOS, run the following on both Arm_1 and Arm_2:

```
# service openvswitch restart
```

4. Start OVS IPsec service. Run on both Arm_1 and Arm_2:

```
# systemctl start openvswitch-ipsec.service
```

5. Set up OVS bridges in both DPUs. Run on both Arm_1 and Arm_2:

```
# ovs-vsctl add-br vxlan-br  
# ovs-vsctl add-port ovs-br $PF_REP  
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

Note

Configuring `other_config:hw-offload=true` sets IPsec Packet offload. Setting it to `false` sets software IPsec.

Note

The MTU of the tunnel interface (PF) should be at least 50 bytes larger than the MTU of the endpoints of the tunnels above (PF_REP) to account for the size of the VXLAN

tunnel header. For example, if the MTU of PF_REP is 1500 then the MTU of PF should be at least 1550.

To configure the MTU of the PF:

```
# ifconfig $PF mtu $PF_MTU up
```

6. Set up IPsec tunnel on the OVS bridge. Three [authentication methods](#) are possible, choose your preferred authentication method and follow the steps relevant to it. Note that the last two authentication methods requires you to create certificates (self-signed certificates or certificate authority certificates).

Note

After the IPsec tunnel is set up using one of the three methods of authentication, strongSwan configuration is done automatically and the `swanctl.conf` files will be generated and strongSwan will run automatically.

Authentication Methods

The following subsections detail the possible authentication methods for setting up the IPsec tunnel on the OVS bridge.

Pre-shared Key

This method configures OVS IPsec using a pre-shared key. You must select a pre-shared key, for example:

```
psk=swordfish
```

1. Set up the VXLAN tunnel:

1. On `Arm_1`, run:


```
# ovs-vsctl add-port vxlan-br tun -- \
set interface tun type=vxlan \
options:local_ip=$ip1 \
options:remote_ip=$ip2 \
options:key=100 \
options:dst_port=4789 \
options:psk=$psk
```

2. On Arm_2, run:

```
# ovs-vsctl add-port vxlan-br tun -- \
set interface tun type=vxlan \
options:local_ip=$ip2 \
options:remote_ip=$ip1 \
options:key=100 \
options:dst_port=4789 \
options:psk=$psk
```

Self-signed Certificate

This method configures OVS IPsec using self-signed certificates. You must generate self-signed certificates and keys. This example demonstrates how to generate self-signed certificates using `ovs-pki` but you may generate them in any other way while skipping step 1.

1. Generate self-signed certificates using `ovs-pki`:

1. On Arm_1, run:

```
# ovs-pki req -u host_1
# ovs-pki self-sign host_1
```

After running this code you should have `host_1-cert.pem` and `host_1-privkey.pem`.

2. On Arm_2, run:

```
# ovs-pki req -u host_2
# ovs-pki self-sign host_2
```

After running this code you should have host_2-cert.pem and host_2-privkey.pem.

2. Configure the certificates and private keys:

1. Copy the certificate of Arm_1 to Arm_2, and the certificate of Arm_2 to Arm_1.
2. On each machine, move both host_1-privkey.pem and host_2-cert.pem to /etc/swanctl/x509/ if on Ubuntu, or /etc/strongswan/swanctl/x509/ if on CentOS.
3. On each machine, move the local private key (host_1-privkey.pem on Arm_1 and host_2-privkey.pem on Arm_2) to /etc/swanctl/private if on Ubuntu, or /etc/strongswan/swanctl/private if on CentOS.

3. Set up OVS other_config on both sides.

1. On Arm_1:

```
# ovs-vsctl set Open_vSwitch .
other_config:certificate=/etc/swanctl/x509/host_1-cert.pem \
other_config:private_key=/etc/swanctl/private/host_1-privkey.pem
```

2. On Arm_2:

```
# ovs-vsctl set Open_vSwitch .
other_config:certificate=/etc/swanctl/x509/host_2-cert.pem \
other_config:private_key=/etc/swanctl/private/host_2-privkey.pem
```

4. Set up the VXLAN tunnel:

1. On Arm_1:

```
# ovs-vsctl add-port vxlan-br vxlanp0 -- set interface vxlanp0 type=vxlan
options:local_ip=$ip1 \
options:remote_ip=$ip2 options:key=100 options:dst_port=4789 \
options:remote_cert=/etc/swanctl/x509/host_2-cert.pem
# service openvswitch-switch restart
```

2. On Arm_2:

```
# ovs-vsctl add-port vxlan-br vxlanp0 -- set interface vxlanp0 type=vxlan
options:local_ip=$ip2 \
options:remote_ip=$ip1 options:key=100 options:dst_port=4789 \
options:remote_cert=/etc/swanctl/x509/host_1-cert.pem
# service openvswitch-switch restart
```

Note

In steps 3 and 4, if you are in CentOS you must change the path of the certificates to `/etc/strongswan/swanctl/x509/` and the path of the private keys to `/etc/strongswan/swanctl/private`.

CA-signed Certificate

This method configures OVS IPsec using certificate authority (CA)-signed certificates. You must generate CA-signed certificates and keys. The example demonstrates how to generate CA-signed certificates using `ovs-pki` but you may generate them in any other way while skipping step 1.

1. Generate CA-signed certificates using `ovs-pki`. For this method, all the certificates and the requests must be in the same directory during the certificate generating and signing. This example refers to this directory as `certsworkspace`.

1. On Arm_1, run:

```
# ovs-pki init --force
# cp /var/lib/openvswitch/pki/controllerca/cacert.pem
<path_to>/certsworkspace
# cd <path_to>/certsworkspace
# ovs-pki req -u host_1
# ovs-pki sign host1 switch
```

After running this code, you should have `host_1-cert.pem`, `host_1-privkey.pem`, and `cacert.pm` in the `certsworkspace` folder.

2. On `Arm_2`, run:

```
# ovs-pki init --force
# cp /var/lib/openvswitch/pki/controllerca/cacert.pem
<path_to>/certsworkspace
# cd <path_to>/certsworkspace
# ovs-pki req -u host_2
# ovs-pki sign host_2 switch
```

After running this code, you should have `host_2-cert.pem`, `host_2-privkey.pem`, and `cacert.pm` in the `certsworkspace` folder.

2. Configure the certificates and private keys:

1. Copy the certificate of `Arm_1` to `Arm_2` and the certificate of `Arm_2` to `Arm_1`.
2. On each machine, move both `host_1-privkey.pem` and `host_2-cert.pem` to `/etc/swanctl/x509/` if on Ubuntu, or `/etc/strongswan/swanctl/x509/` if on CentOS.
3. On each machine, move the local private key (`host_1-privkey.pem` if on `Arm_1` and `host_2-privkey.pem` if on `Arm_2`) to `/etc/swanctl/private` if on Ubuntu, or `/etc/strongswan/swanctl/private` if on CentOS.
4. On each machine, copy `cacert.pem` to the `x509ca` directory under `/etc/swanctl/x509ca/` if on Ubuntu, or `/etc/strongswan/swanctl/x509ca/` if on CentOS.

3. Set up OVS `other_config` on both sides.

1. On Arm_1:

```
# ovs-vsctl set Open_vSwitch . \
other_config:certificate=/etc/strongswan/swanctl/x509/host_1.pem \
other_config:private_key=/etc/strongswan/swanctl/private/host_1-
privkey.pem \
other_config:ca_cert=/etc/strongswan/swanctl/x509ca/cacert.pem
```

2. On Arm_2:

```
# ovs-vsctl set Open_vSwitch . \
other_config:certificate=/etc/strongswan/swanctl/x509/host_2.pem \
other_config:private_key=/etc/strongswan/swanctl/private/host_2-
privkey.pem \
other_config:ca_cert=/etc/strongswan/swanctl/x509ca/cacert.pem
```

4. Set up the tunnel:

1. On Arm_1:

```
# ovs-vsctl add-port vxlan-br vxlanp0 -- set interface vxlanp0 type=vxlan
options:local_ip=$ip1 \
options:remote_ip=$ip2 options:key=100 options:dst_port=4789 \
options:remote_name=host_2
# service openvswitch-switch restart
```

2. On Arm_2:

```
# ovs-vsctl add-port vxlan-br vxlanp0 -- set interface vxlanp0 type=vxlan
options:local_ip=$ip2 \
options:remote_ip=$ip1 options:key=100 options:dst_port=4789 \
options:remote_name=host_1
# service openvswitch-switch restart
```

Note

In steps 3 and 4, if you are in CentOS you must change the path of the certificates to `/etc/strongswan/swanctl/x509/`, the path of the CA certificate to `/etc/strongswan/swanctl/x509ca/`, and the path of the private keys to `/etc/strongswan/swanctl/private/`.

Ensuring IPsec is Configured

Using `/opt/mellanox/iproute2/sbin/ip xfrm state show`, you should be able to see 4 IPsec states for the IPsec connection you configured with the keyword in mode packet meaning which means that you are in IPsec packet HW offload mode.

For example, after configuring IPsec using pre-shared key method, you would get something similar to the following on Arm_1:

```
# /opt/mellanox/iproute2/sbin/ip xfrm state show

src 192.168.50.1 dst 192.168.50.2
proto esp spi 0xcc8bf8ad reqid 1 mode transport
replay-window 0 flag esn
aead rfc4106(gcm(aes))
0x9f45cc4577e70c4e077bcc0c1473a782143e7ad199f58566519639d03b593b8996383f11 128
anti-replay esn context:
seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
replay_window 1, bitmap-length 1
00000000
crypto offload parameters: dev p0 dir out mode packet
sel src 192.168.50.1/32 dst 192.168.50.2/32 proto udp sport 4789
src 192.168.50.2 dst 192.168.50.1
proto esp spi 0xce8bf4b6 reqid 1 mode transport
```

```
replay-window 0 flag esn
aad rfc4106(gcm(aes))
0xf2d0e335d9a64ef6e385a630a32b0e43bb52f581290cd34bbb8f7592d54f11657ed0258e 128
anti-replay esn context:
seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
replay_window 32, bitmap-length 1
00000000
crypto offload parameters: dev p0 dir in mode packet
sel src 192.168.50.2/32 dst 192.168.50.1/32 proto udp dport 4789
src 192.168.50.1 dst 192.168.50.2
proto esp spi 0xcb600a84 reqid 2 mode transport
replay-window 0 flag esn
aad rfc4106(gcm(aes))
0x7fb26035299bcc9b973abea5d581acfbcf87cbf0bd053b745c4d95c62311f934010973f6 128
anti-replay esn context:
seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
replay_window 1, bitmap-length 1
00000000
crypto offload parameters: dev p0 dir out mode packet
sel src 192.168.50.1/32 dst 192.168.50.2/32 proto udp dport 4789
src 192.168.50.2 dst 192.168.50.1
proto esp spi 0xc137d5a0 reqid 2 mode transport
replay-window 0 flag esn
aad rfc4106(gcm(aes))
0x28e3d12ad4e24aa9d9de9459de8ef8bb4379e8e12faac0054c5b629b6aa50fdeda8e4574 128
anti-replay esn context:
seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
replay_window 32, bitmap-length 1
00000000
crypto offload parameters: dev p0 dir in mode packet
sel src 192.168.50.2/32 dst 192.168.50.1/32 proto udp sport 4789
```

After insuring that the IPsec connection is configured, you can send encrypted traffic between host_1 and host_2 using the HOST_PFs IP addresses.

Configuring OVS IPsec Using strongSwan Manually

This section configures an OVS VXLAN tunnel which then uses `swanctl.conf` files and runs strongSwan (the IPsec daemon) manually.

Note

Before proceeding with this section, make sure to follow the procedure in section "[Enabling IPsec Packet Offload](#)" for both DPUs.

1. Build a VXLAN tunnel over OVS and connect the PF representor to the same OVS bridge.

1. On Arm_1:

```
# ovs-vsctl add-br vxlan-br
# ovs-vsctl add-port vxlan-br PF_REP
# ovs-vsctl add-port vxlan-br vxlan11 -- set interface vxlan11 type=vxlan
options:local_ip=$ip1 \
options:remote_ip=$ip2 options:key=100 options:dst_port=4789 \
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

2. On Arm_2:

```
# ovs-vsctl add-br vxlan-br
# ovs-vsctl add-port vxlan-br PF_REP
# ovs-vsctl add-port vxlan-br vxlan11 -- set interface vxlan11 type=vxlan
options:local_ip=$ip2 \
options:remote_ip=$ip1 options:key=100 options:dst_port=4789 \
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

2. If your operating system is Ubuntu, run on both Arm_1 and Arm_2:


```
service openvswitch-switch start
```

If your operating system is CentOS, run:

```
service openvswitch restart
```

3. Enable TC offloading for the PF. Run on both Arm_1 and Arm_2:

```
# ethtool -K $PF hw-tc-offload on
```

4. Disable host PF as the port owner from Arm. Run on both Arm_1 and Arm_2:

```
# mlxprivhost -d /dev/mst/mt${pciconf} --disable_port_owner r
```

Note

To get `${pciconf}`, run the following on the DPU:

```
# ls --color=never /dev/mst/ | grep --color=never '^m.*f0$' |  
cut -c 3-
```

For example:

```
# mlxprivhost -d /dev/mst/mt41686_pciconf0 --  
disable_port_owner r
```

5. Configure the `swanctl.conf` files for each machine. See section [swanctl.conf Files](#).

Note

Each machine should have exactly one `.swanctl.conf` file in `/etc/swanctl/conf.d/`.

6. Load the `swanctl.conf` files and initialize strongSwan. Run:

1. On the Arm_2, run:

```
systemctl restart strongswan.service  
swanctl --load-all
```

2. On the Arm_1, run:

```
systemctl restart strongswan.service  
swanctl --load-all  
swanctl -i --child bf
```

Now the IPsec connection should be established.

swanctl.conf Files

strongSwan configures IPsec packet HW offload using a new value added to its configuration file `swanctl.conf`. The file should be placed under `sysconfdir` which by default can be found at `/etc/swanctl/swanctl.conf`.

The terms Left (BFL) and Right (BFR), in reference to the illustration under "[Application Architecture](#)", are used to identify the two nodes (or machines) that communicate.

Note

Either side (BFL or BFR) can fulfill either role (initiator or receiver).

In this example, 192.168.50.1 is used for the left PF uplink and 192.168.50.2 for the right PF uplink.

```
connections {
  BFL-BFR {
    local_addrs = 192.168.50.1
    remote_addrs = 192.168.50.2

    local {
      auth = psk
      id = host1
    }
    remote {
      auth = psk
      id = host2
    }
    children {
      bf-out {
        local_ts = 192.168.50.1/24 [udp]
        remote_ts = 192.168.50.2/24 [udp/4789]
        esp_proposals = aes128gcm128-x25519-esn
        mode = transport
        policies_fwd_out = yes
        hw_offload = packet
      }
      bf-in {
        local_ts = 192.168.50.1/24 [udp/4789]
        remote_ts = 192.168.50.2/24 [udp]
        esp_proposals = aes128gcm128-x25519-esn
        mode = transport
        policies_fwd_out = yes
        hw_offload = packet
      }
    }
    version = 2
    mobike = no
  }
}
```

```

reauth_time = 0
proposals = aes128-sha256-x25519
}
}

secrets {
ike-BF {
id-host1 = host1
id-host2 = host2
secret = 0sv+NkxY9LLZvwj4qCC2o/gGrWDF2d21jL
}
}
}

```

The BFB installation will place two example `swanctl.conf` files for BFL and BFR (`BFL.swanctl.conf` and `BFR.swanctl.conf` respectively) in the `strongSwan conf.d` directory. Each node should have only one `swanctl.conf` file in its `strongSwan conf.d` directory.

Note that:

- "hw_offload = packet" is responsible for configuring IPsec packet offload
- Packet offload support has been added to the existing `hw_offload` field and preserves backward compatibility.

For your reference:

Value	Description
no	Do not configure HW offload.
crypto	Configure crypto HW offload if supported by the kernel and hardware, fail if not supported.
yes	Same as crypto (considered legacy).
packet	Configure packet HW offload if supported by the kernel and hardware, fail if not supported.
auto	Configure packet HW offload if supported by the kernel and hardware, do not fail (perform fallback to crypto or no as necessary).

- Whenever the value of `hw_offload` is changed, strongSwan configuration must be reloaded.
- Switching to crypto HW offload requires setting up `devlink/ipsec_mode` to `none` beforehand.
- Switching to packet HW offload requires setting up
- `[udp/4789]` is crucial for instructing strongSwan to IPsec only VXLAN communication.
- Packet HW offload can only be done on what is streamed over VXLAN.

Mind the following limitations:

Fields	Limitation
<code>reauth_time</code>	Ignored if set
<code>rekey_time</code>	Do not use. Ignored if set.
<code>rekey_bytes</code>	Do not use. Not supported and will fail if it is set.
<code>rekey_packets</code>	Use for rekeying

References

- - `/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption.sh`
 - `/opt/mellanox/doca/applications/east_west_overlay_encryption/east_west_overlay_encryption_pa`

NVIDIA DOCA Eth L2 Forwarding Application Guide

This document provides an Ethernet L2 Forwarding implementation on top of the NVIDIA® BlueField® DPU.

Introduction

The Ethernet L2 Forwarding application is a [DOCA Ethernet](#) based application that forwards traffic from a single RX port to a single TX port and vice versa, leveraging DOCA's task/event batching feature for enhanced performance.

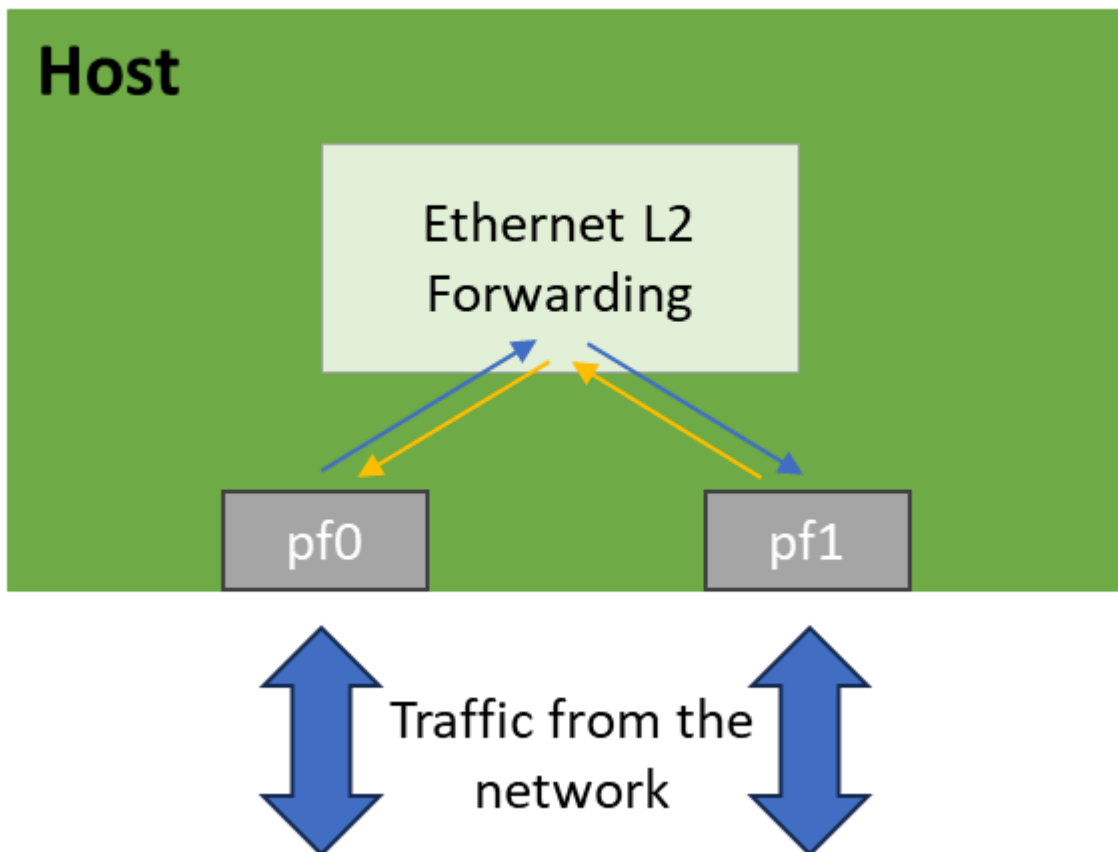
The application can run both on the **Host** and the **BlueField**, and has two main modes:

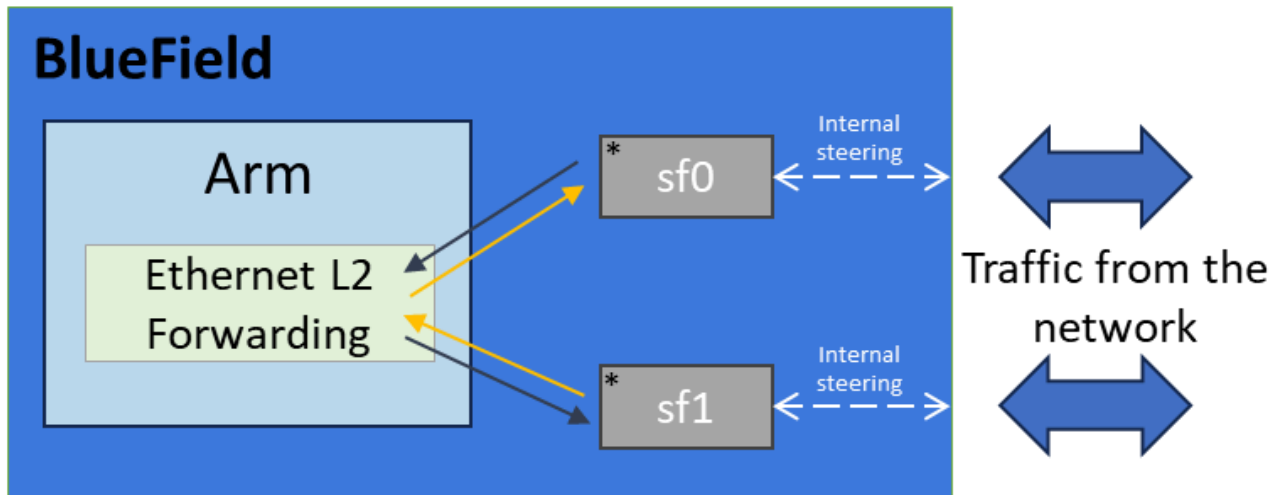
- Two-sided forwarding – device 1 → device 2 **and** device 2 → device 1
- One-sided forwarding – device 1 → device 2 **or** device 2 → device 1

The one-sided mode offers better performance, enlarging the packets forwarding rate.

System Design

The Ethernet L2 Forwarding application runs on the host or the BlueField.

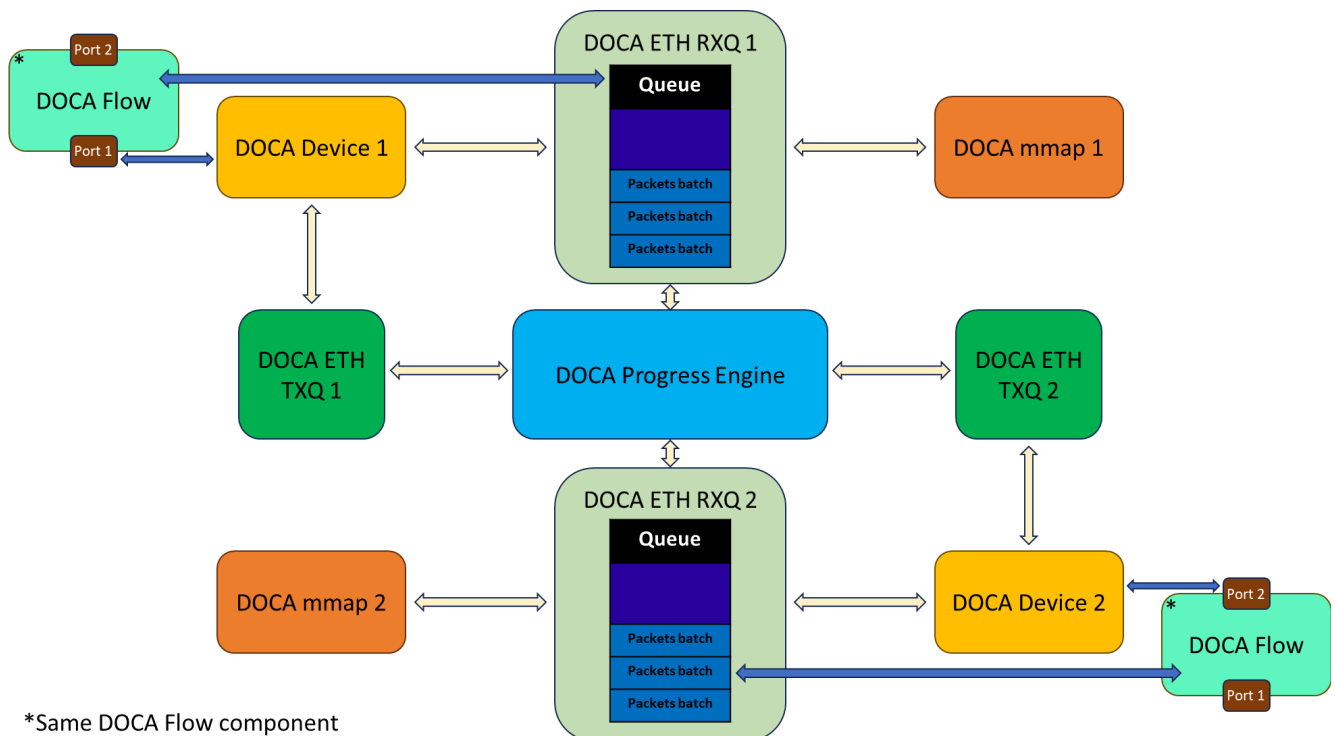




*DOCA Ethernet library only handles SFs

Application Architecture

The Ethernet L2 Forwarding application runs on top of the DOCA Ethernet API to form an (two/one-sided) L2 forwarding between two ports.



1. Two DOCA devices are opened.
2. Two DOCA mmaps are created.

3. Two DOCA Flow ports are configured and started, each with a different opened DOCA device.
4. Two DOCA Ethernet TXQ and RXQ contexts are initialized, each TXQ-RXQ pair with a different opened DOCA device such that traffic is steered from the device to the corresponding RXQ, and from the corresponding TXQ to the device.
5. Forwarding - Packets received by device x are steered to RXQ x, then allocated to TXQ y and sent by device y (and vice versa).

DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA Ethernet - Programming Guide](#)
- [DOCA Flow - Programming Guide](#)

For additional information about the used DOCA libraries, please refer to the respective programming guides.

Compiling the Application

Installation

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

Overview

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for both compilation of the applications "as-is", as well as provides the ability to modify the sources and then compile the new version of the application. For more information about the applications, as well as development and compilation tips, please refer to the [DOCA Applications](#) main guide.

The sources of the application can be found under the application's directory:

```
/opt/mellanox/doca/applications/eth_l2_fwd/.
```


Compiling All Applications

The applications are all defined under a single meson project, meaning that the default compilation will compile **all** the DOCA applications.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Note

`doca_eth_l2_fwd` will be created under `/tmp/build/eth_l2_fwd/`.

Compiling Only the Current Application

1. To directly build only the Ethernet L2 Forwarding application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_eth_l2_fwd=true  
ninja -C /tmp/build
```

Note

`doca_eth_l2_fwd` will be created under `/tmp/build/eth_l2_fwd/`.

2. Alternatively, one can set the desired flags in the `meson_options.txt` file instead of providing them in the compilation command line:

1. Edit the following flags in `/opt/mellanox/doca/applications/meson_options.txt`:

- Set `enable_all_applications` to `false`
- Set `enable_eth_l2_fwd` to `true`

2. The same compilation commands should be used, as were shown in the previous section:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Note

`doca_eth_l2_fwd` will be created under `/tmp/build/eth_l2_fwd/`.

Troubleshooting

Please refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue you may encounter with the compilation of the DOCA applications.

Running the Application

Application Execution

The Ethernet L2 Forwarding application is provided in source form, hence a compilation is required before the application can be executed.

1. Application usage instructions:

Usage: `doca_eth_l2_fwd` [DOCA Flags] [Program Flags]

DOCA Flags:

`-h, --help` Print a help synopsis
`-v, --version` Print program version information
`-l, --log-level` Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
`--sdk-log-level` Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
`-j, --json <path>` Parse all command flags from an input json file

Program Flags:

`-d, --devs-names <name1,name2>` Set two IB devices names separated by a comma, without spaces.
`-r, --rate <rate>` Set packets receive rate (in [MB/s]), default is 12500.
`-ps, --pkt-size <size>` Set max packet size (in [B]), default is 1600.
`-t, --time <time>` Set packet max process time (in [μ s]), default is 1.
`-nt, --num-tasks <num>` Set number of tasks per batch, default is 128.
`-nb, --num-batches <num>` Set number of task batches, default is 32.
`-o, --one-sided-forwarding <num>` Set one-sided forwarding: 0 - two-sided forwarding, 1 - device 1 -> device 2, 2 - device 2 -> device 1. default is 0.
`-f, --max-forwardings <num>` Set max forwardings after which the application run will end, default is 0, meaning no limit.

For additional information, please refer to the [Command Line Flags](#) section below.

Note

The above usage printout can be printed to the command line using the `-h` (or `--help`) options:

```
./doca_eth_l2_fwd -h
```

2. CLI example for running the application either on the **BlueField** or on the **host**:

```
./doca_eth_l2_fwd -d mlx5_0,mlx5_1
```

Note

Both IB devices identifiers (`mlx5_0`, `mlx5_1`) should match the identifiers of the desired IB devices.

3. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_eth_l2_fwd --json [json_file]
```

For example:

```
./doca_eth_l2_fwd --json ./eth_l2_fwd_params.json
```

Note

Before execution, please ensure that the used JSON file contains the correct configuration parameters, and especially the desired IB devices names needed for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (Requires compilation with Trace level support) 	<pre>"log-level": 60</pre>
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	N/A
	j	json	Parse all command flags from an input json file	N/A

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
Program flags	d	devs-names	<p>Two IB devices names, separated by a comma, without spaces.</p> <div style="background-color: #ffffcc; padding: 10px; border: 1px solid #ccc;"> <p>Note This is a mandatory flag.</p> </div>	<pre>"devs-names": "mlx5_0,mlx5_1"</pre>
	r	rate	The rate (in [MB/s]) in which the RX port is expected to receive traffic.	<pre>"rate": 12500</pre>
	ps	pkt-size	The maximum size (in [B]) of a received packet.	<pre>"pkt-size": 1600</pre>
	t	time	The maximum time taking to process a single packet.	<pre>"time": 1</pre>
	nt	num-tasks	The number of tasks to set per a single task batch.	<pre>"num-tasks": 128</pre>
	nb	num-batches	The number of task batches to set for the TX side.	<pre>"num-batches": 32</pre>
	o	one-sided-forwarding	Flag to set one of 3 options: 0 - Two-sided forwarding. 1 - One-sided forwarding from device 1 to device 2. 2 - One-sided forwarding from device 2 to device 1.	<pre>"one-sided-forwarding": 0</pre>
f	max-forwardings	The maximum number of forwarding	<pre>"max-forwardings": 32</pre>	

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Please refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue you may encounter with the installation or execution of the DOCA applications.

Application Code Flow

1. Parse application argument.

1. Initialize Arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register Ethernet L2 Forwarding application parameters.

```
register_eth_l2_forwarding_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

1. Parse DOCA flags.

2. Parse application parameters.

2. Execute Ethernet L2 Forwarding application main logic.

```
eth_l2_fwd_execute();
```

1. Open the two chosen DOCA devices.

2. Initialize necessary DOCA Core objects.
 3. Initialize ETH RXQ/TXQ contexts for the devices.
 4. Forward packets.
3. Clean up application resources.

```
eth_l2_fwd_cleanup();
```

1. Stop all contexts and drain tasks.
 2. Free all application resources.
4. Arg parser destroy.

```
doca_argp_destroy()
```

References

- /opt/mellanox/doca/applications/eth_l2_fwd/
- /opt/mellanox/doca/applications/eth_l2_fwd/eth_l2_fwd_params.json

NVIDIA DOCA File Compression Application Guide

This document provides a file compression implementation on top of the NVIDIA® BlueField® DPU.

Introduction

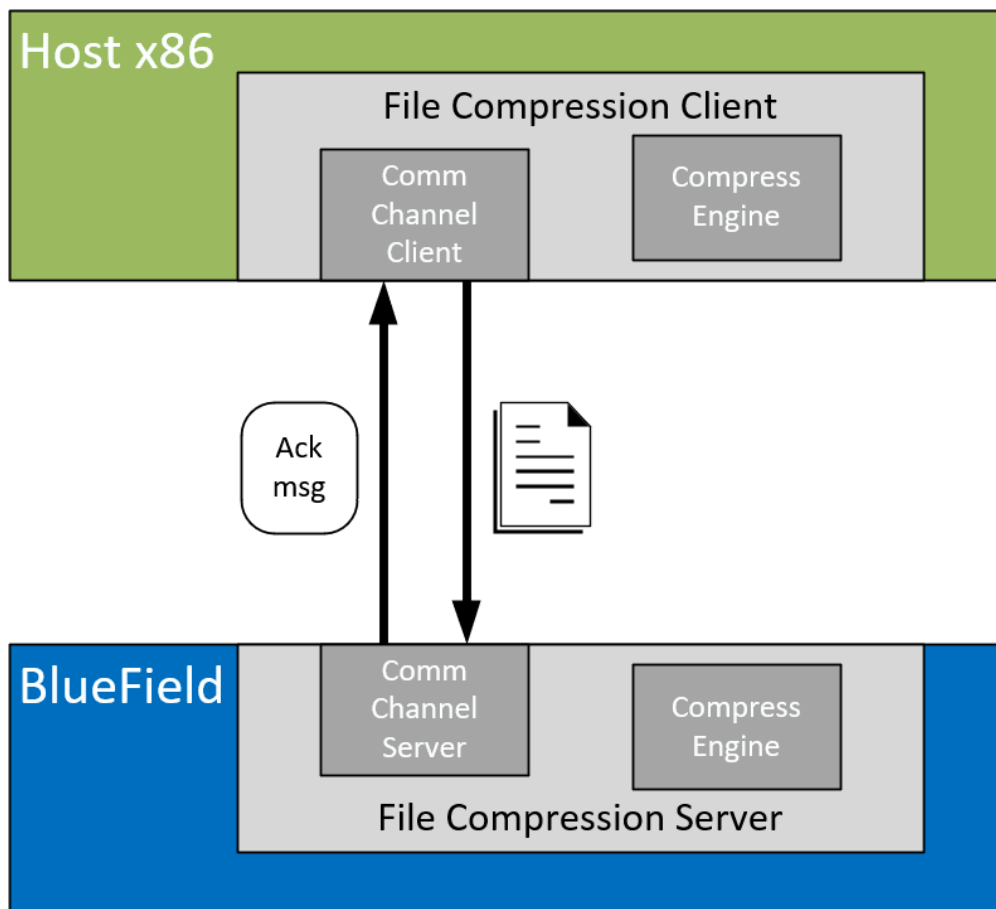
The file compression application exhibits how to use the [DOCA Compress](#) API to compress and decompress data using hardware acceleration as well as sending and receiving it using the [DOCA Comch](#) API.

The application's logic includes both a client and a server:

- Client side – the application opens a file, compresses it, and sends the checksum of the source file with the compressed data to the server
- Server side – the application saves the received file in a buffer, decompresses it, and compares the received checksum with the calculated one

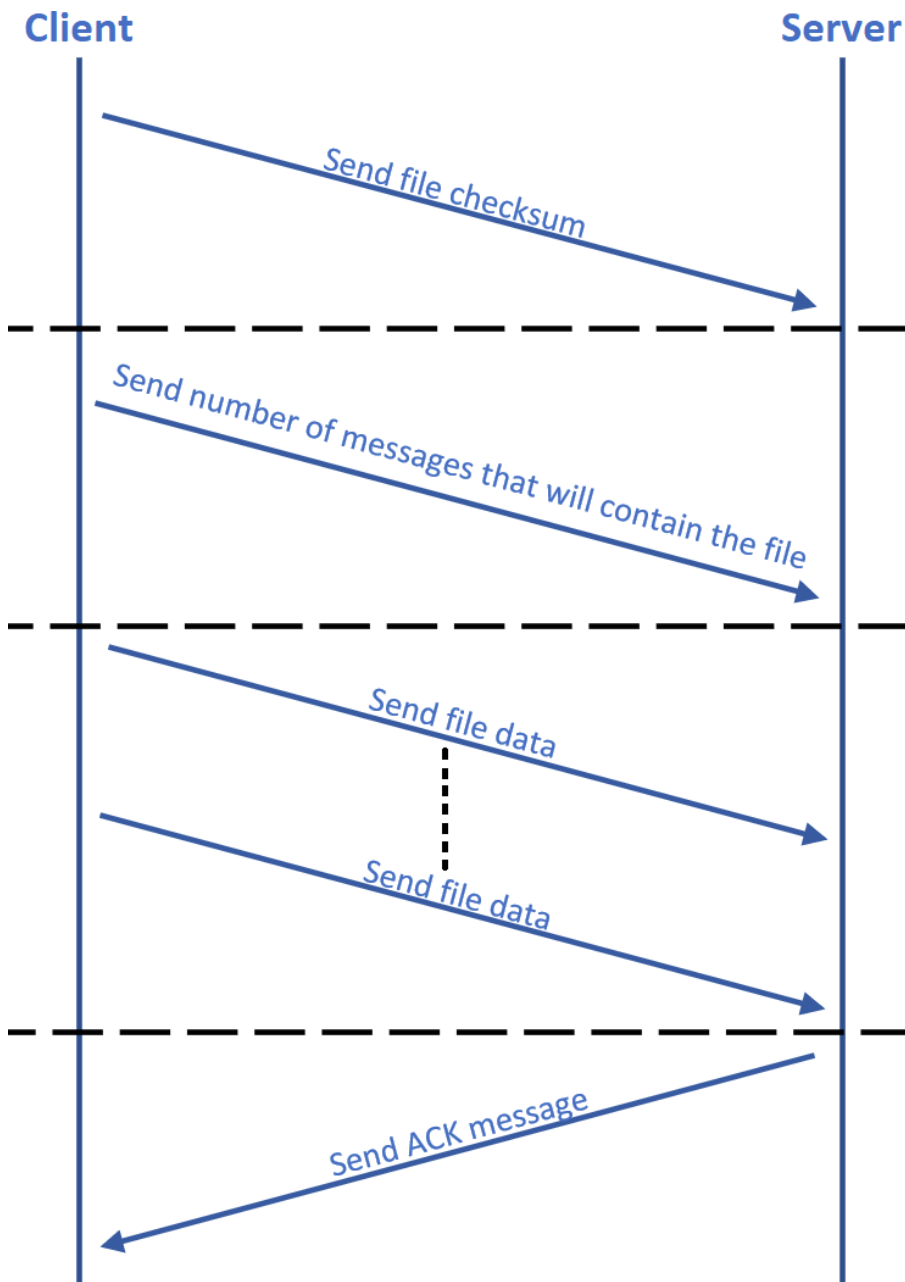
System Design

The file compression application client runs on the host and the server runs on the DPU.



Application Architecture

The file compression application runs on top of the DOCA Comm Channel API to send and receive the file from the host and to the DPU.



1. Connection is established on both sides by DOCA Comm Channel API.
2. Client compresses the data:
 - When compress engine is available – submits compress job with DOCA Compress API and sends the result to the server
 - When compress engine is unavailable – compresses the data in software
3. Client sends the number of messages needed to send the compressed content of the file.

4. Client sends data segments in size of up to 4080 bytes.
5. Server saves the received data in a buffer and submits a decompress job.
6. Server sends an ACK message to the client when all parts of the file are received successfully.
7. Server compares the received checksum to the calculated checksum.
8. Server writes the decompressed data to an output file.

DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA Compress](#)
- [DOCA Comch](#)

Refer to their respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/file_compression/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

`doca_file_compression` is created under `/tmp/build/file_compression/`.

Compiling File Compression Application Only

To directly build only the file compression application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_file_compression=true  
ninja -C /tmp/build
```

Info

doca_file_compression is created under /tmp/build/file_compression/.

Alternatively, the user may set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
 - Set enable_all_applications to false
 - Set enable_file_compression to true
2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_file_compression is created under /tmp/build/file_compression/.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application.

Running the Application

Application Execution

The file compression application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_file_compression [DOCA Flags] [Program Flags]
```

DOCA Flags:

```
-h, --help Print a help synopsis  
-v, --version Print program version information  
-l, --log-level Set the (numeric) log level for the program <10=DISABLE,  
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>  
--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE,  
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>  
-j, --json <path> Parse all command flags from an input json file
```

Program Flags:

```
-p, --pci-addr DOCA Comm Channel device PCI address  
-r, --rep-pci DOCA Comm Channel device representor PCI address  
-f, --file File to send by the client / File to write by the server  
-t, --timeout Application timeout for receiving file content messages, default is 5  
sec
```

Info

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_file_compression -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on BlueField:

```
./doca_file_compression -p 03:00.0 -r 3b:00.0 -f received.txt
```

Note

Both the DOCA Comm Channel device PCIe address (03:00.0) and the DOCA Comm Channel device representor PCIe address (3b:00.0) should match the addresses of the desired PCIe devices.

3. CLI example for running the application on the host:

```
./doca_file_compression -p 3b:00.0 -f send.txt
```

Note

The DOCA Comm Channel device PCIe address (3b:00.0) should match the address of the desired PCIe device.

4. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_file_compression --json [json_file]
```

For example:

```
./doca_file_compression --json ./file_compression_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none">• DISABLE=10• CRITICAL=20• ERROR=30• WARNING=40• INFO=50• DEBUG=60• TRACE=70 (requires compilation with TRACE log level support)	<pre>"log-level": 60</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	<pre>"sdk-log-level": 40</pre>
	j	json	Parse all command flags from an input JSON file	N/A
Program flags	f	file	For client – path to the file to be sent For server – path to write the file into <div style="background-color: #ffffcc; padding: 5px;"> <p>(i) Note This is a mandatory flag.</p> </div>	<pre>"file": "/tmp/data.txt"</pre>
	p	pci-addr	Comm Channel DOCA device PCIe address <div style="background-color: #ffffcc; padding: 5px;"> <p>(i) Note This is a mandatory flag.</p> </div>	<pre>"pci-addr": 03:00.1</pre>
	r	rep-pci	Comm Channel DOCA device representor PCIe address	<pre>"rep-pci": b1:00.1</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
			<p>i Note This flag is mandatory only on the DPU.</p>	

i **Info**

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.

Application Code Flow

1. Parse application argument.
 1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register file compression application parameters.

```
register_file_compression_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

1. Parse app parameters.

2. Set endpoint attributes.

```
set_endpoint_properties();
```

1. Set maximum message size of 4080 bytes.

2. Set maximum number of messages allowed.

3. Create comm channel endpoint.

```
doca_comm_channel_ep_create();
```

1. Create endpoint for client/server.

4. Run client/server main logic.

```
file_compression_client/server();
```

5. Clean up the file compression application.

```
file_compression_cleanup();
```

1. Free all application resources.

6. Arg parser destroy.

```
doca_argp_destroy()
```

References

- /opt/mellanox/doca/applications/file_compression/
- /opt/mellanox/doca/applications/file_compression/file_compression_params.json

NVIDIA DOCA File Integrity Application Guide

This guide provides a file integrity implementation on top of NVIDIA® BlueField® DPU.

Introduction

The file integrity application exhibits how to use the [DOCA Comch](#) and [DOCA SHA](#) libraries to send and receive a file securely.

The application's logic includes both a client and a server:

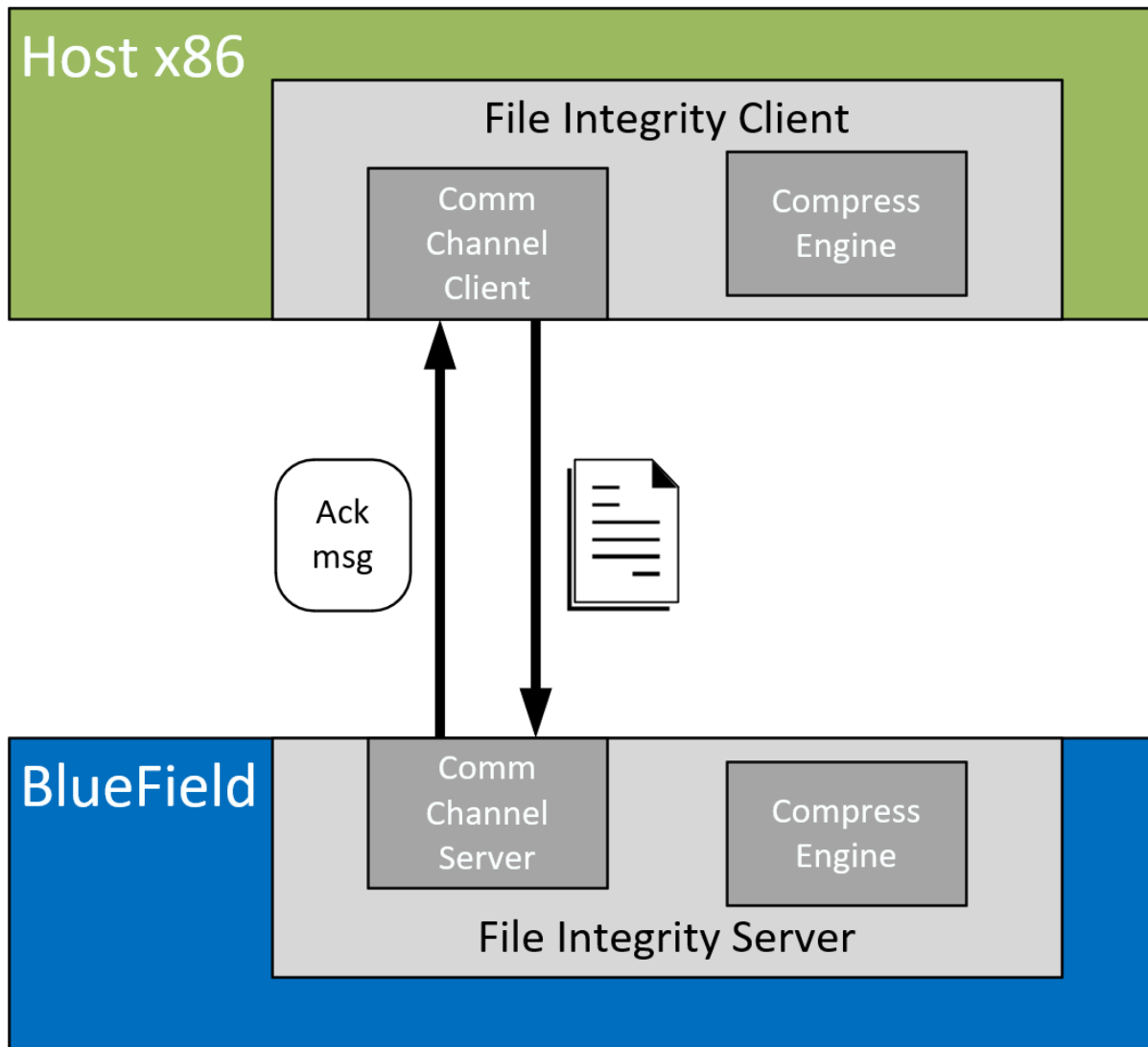
- Client side – the application opens a file, calculates the SHA (secure hash algorithm) digest on it, and sends the digest of the source file alongside the file itself to the server
- Server side – the application calculates the SHA on the received file and compares the received digest to the calculated one to check if the file has been compromised

Note

SHA hardware acceleration is only available on the BlueField-2 DPU.
This application is not supported on BlueField-3.

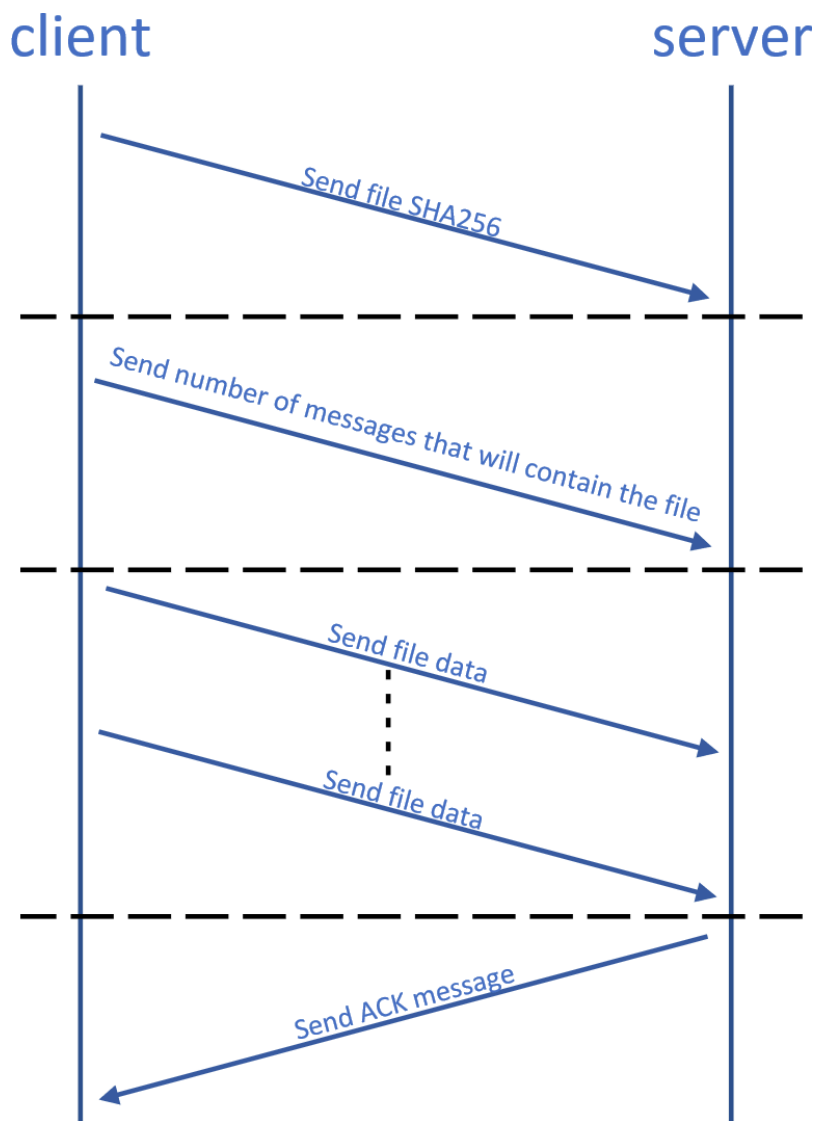
System Design

The file integrity application runs in client mode (host) and server mode (DPU).



Application Architecture

The file integrity application runs on top of the DOCA Comm Channel API to send and receive files from the host and DPU.



1. Connection is established on both sides by the Comm Channel API.
2. Client submits SHA job with the DOCA SHA library and sends the result to the server.
3. Client sends the number of messages required to send the content of the file.
4. Client sends data segments in size of up to 4032 bytes.
5. Server submits a partial SHA job on each received segment.
6. Server sends an ACK message to the client when all parts of the file are received successfully.
7. Server compares the received SHA to the calculated SHA.

DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA SHA](#)
- [DOCA Comch](#)

Refer to their respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory: `/opt/mellanox/doca/applications/file_integrity/` directory.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

`doca_file_integrity` is created under `/tmp/build/file_integrity/`.

Compiling Only the Current Application

To directly build only the file integrity application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_file_integrity=true  
ninja -C /tmp/build
```

Info

`doca_file_integrity` is created under `/tmp/build/file_integrity/`.

Alternatively, one can set the desired flags in the `meson_options.txt` file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:

- Set enable_all_applications to false
- Set enable_file_integrity to true

2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_file_integrity is created under /tmp/build/file_integrity/.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application.

Running the Application

Application Execution

The file integrity application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

Usage: `doca_file_integrity` [DOCA Flags] [Program Flags]

DOCA Flags:

-h, --help Print a help synopsis

-v, --version Print program version information

-l, --log-level Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

-j, --json <path> Parse all command flags from an input json file


Program Flags:

-p, --pci-addr DOCA Comm Channel device PCI address

-r, --rep-pci DOCA Comm Channel device representor PCI address


-f, --file File to send by the client / File to write by the server

-t, --timeout Application timeout for receiving file content messages, default is 5 sec

 **Info**

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_file_integrity -h
```

 **Info**

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on BlueField:

```
./doca_file_integrity -p 03:00.0 -r 3b:00.0 -f received.txt
```

Note

Both the DOCA Comm Channel device PCIe address (03:00.0) and the DOCA Comm Channel device representor PCIe address (3b:00.0) should match the addresses of the desired PCIe devices.

3. CLI example for running the application on the host:

```
./doca_file_integrity -p 3b:00.0 -f send.txt
```

Note

The DOCA Comm Channel device PCIe address (3b:00.0) should match the address of the desired PCIe device.

4. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_file_integrity --json [json_file]
```

For example:

```
./doca_file_integrity --json ./file_integrity_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment .

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none">• DISABLE=10• CRITICAL=20• ERROR=30• WARNING=40• INFO=50• DEBUG=60• TRACE=70 (requires compilation with TRACE log level support)	<pre>"log-level": 60</pre>
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none">• DISABLE=10• CRITICAL=20• ERROR=30• WARNING=40• INFO=50• DEBUG=60• TRACE=70	<pre>"sdk-log- level": 40</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	j	json	Parse all command flags from an input JSON file	N/A
Program flags	f	file	<p>For client – path to the file to be sent For server – path to write the file into</p> <p>Note This is a mandatory flag.</p>	<pre>"file": "/tmp/data .txt"</pre>
	p	pci-addr	<p>Comm Channel DOCA device PCIe address</p> <p>Note This is a mandatory flag.</p>	<pre>"pci-addr": 03:00.1</pre>
	r	rep-pci	<p>Comm Channel DOCA device representor PCIe address</p> <p>Note This flag is mandatory only on the DPU.</p>	<pre>"rep-pci": b1:00.1</pre>

i Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Please refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications .

Application Code Flow

1. Parse application argument.

1. Initialize the arg parser resources and register DOCA general parameters.

```
doca_arg_init();
```

2. Register file integrity application parameters.

```
register_file_integrity_params();
```

3. Parse application parameters.

```
doca_argp_start();
```

2. Set endpoint attributes.

```
set_endpoint_properties();
```

1. Set maximum message size of 4032 bytes.

2. Set number of maximum messages allowed per connection.

3. Create Comm Channel endpoint.

```
doca_comm_channel_ep_create();
```

1. Create endpoint for client/server.

4. Create SHA context.

```
doca_sha_create();
```

1. Create SHA context for submitting SHA jobs for client/server.

5. Run client/server main logic.

```
file_integrity_client/server();
```

6. Clean up the File Integrity app.

```
file_integrity_cleanup();
```

1. Free all application resources.

References

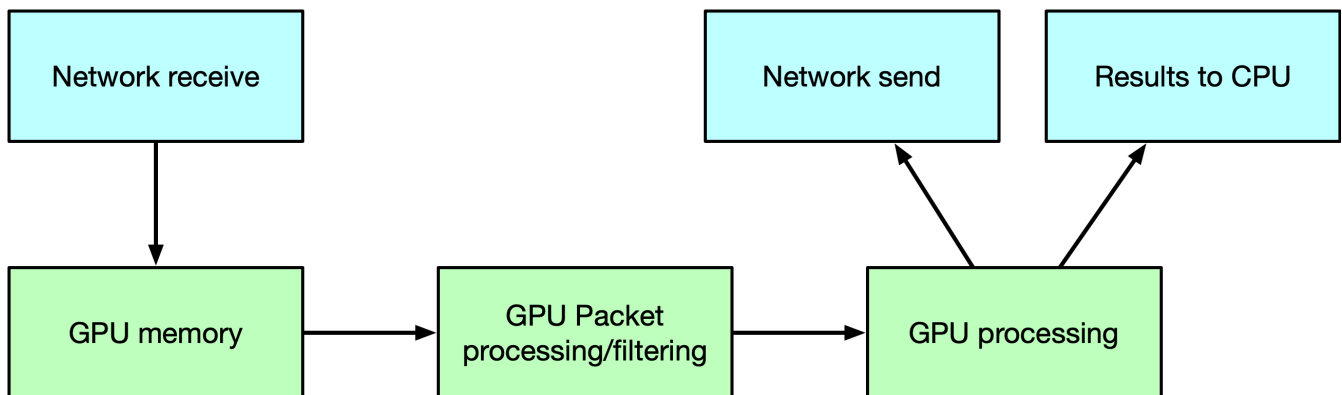
- `/opt/mellanox/doca/applications/file_integrity/`
- `/opt/mellanox/doca/applications/file_integrity/file_integrity_params.json`

NVIDIA DOCA GPU Packet Processing Application Guide

This guide provides a description of the GPU packet processing application to demonstrate the use of [DOCA GPUNetIO](#), [DOCA Ethernet](#), and [DOCA Flow](#) libraries to implement a GPU traffic analyzer.

Introduction

Real-time GPU processing of network packets is a useful technique to several different application domains, including signal processing, network security, information gathering, and input reconstruction. The goal of these applications is to realize an inline packet processing pipeline to receive packets in GPU memory (without staging copies through CPU memory), process them in parallel with one or more CUDA kernels, and then run inference, evaluate, or send the result of the calculation over the network.

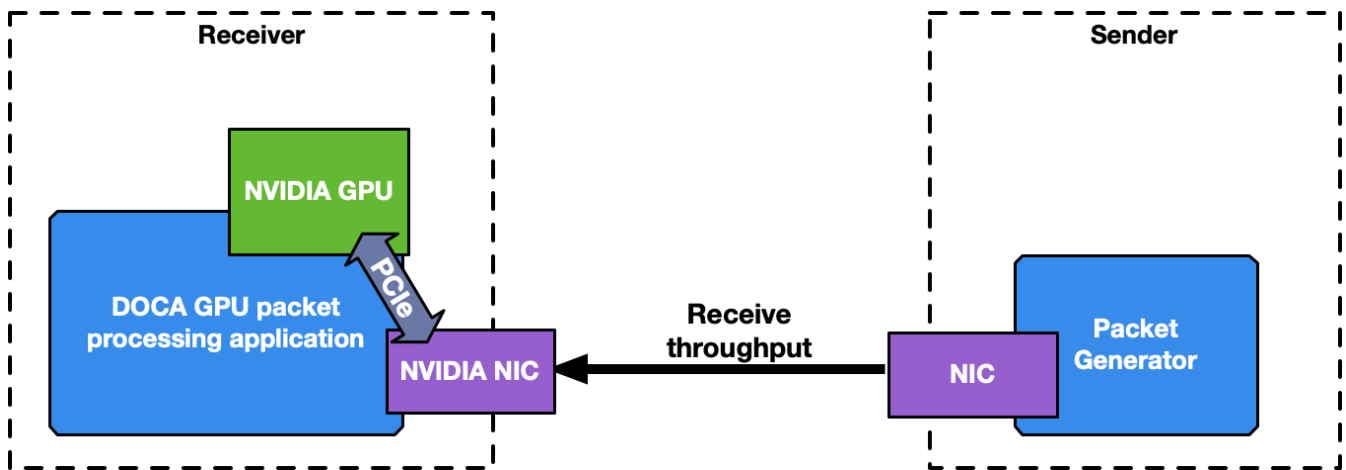


The type of data processing heavily depends on the use case. The goal of this application is to provide a basic layout to reuse in the most common use cases of being able to receive, differentiate and manage the following types of network traffic in multiple queues: UDP, TCP and ICMP.

This application is an enhancement of the use cases presented in [this NVIDIA blog post](#) about DOCA GPUNetIO.

System Design

This is a receive-and-process DOCA application, so a packet generator sending packets is required to test the application.



To launch the application, the PCIe address of the GPU and NIC are required.

Application Architecture

The application manages different types of traffic differently, dedicating up to 4 receive queues to each one using DOCA Flow with RSS mode to assign each packet to the right queue. The more queues the application uses, the higher is the degree of parallelism in how receive data is processed and how long it takes.

Tip

It is highly recommended to use more than one receive queue for 100Gb/s or higher network traffic throughput.

ICMP Network Traffic

If the network interface used for the application has an IP address, it is possible to ping that interface. ICMP packets are received by a dedicated CUDA kernel (file `gpu_kernels/receive_icmp.cu`) which:

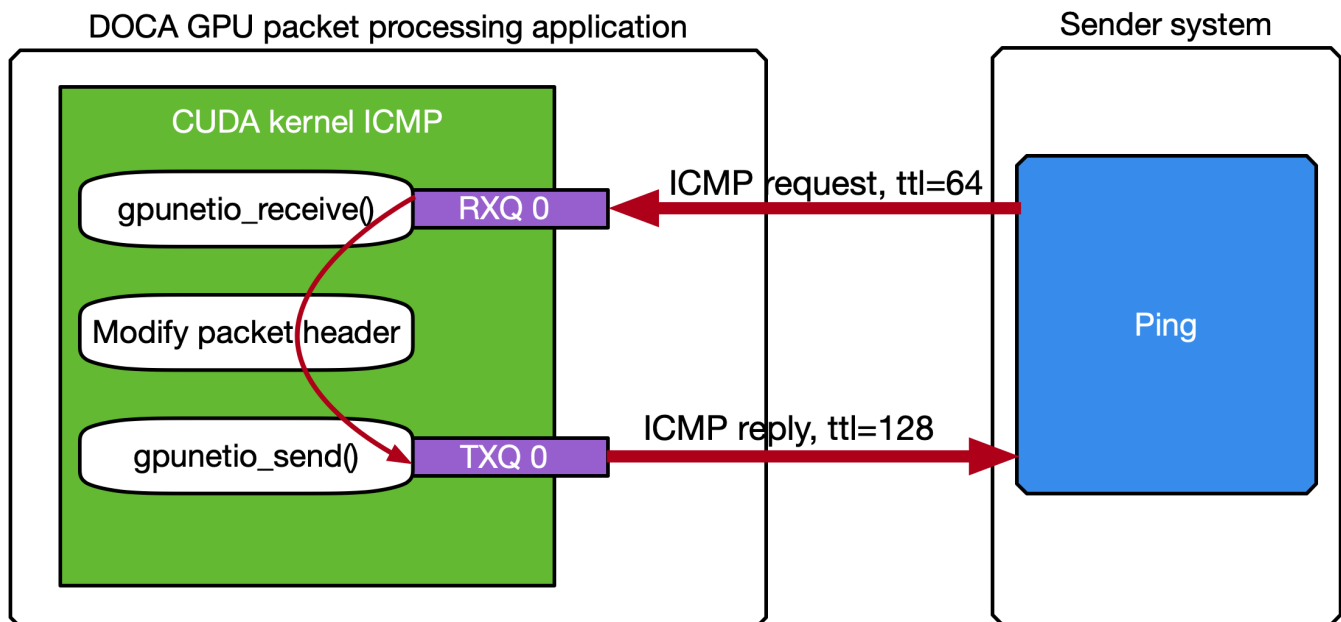
1. Receives packets using the DOCA GPUNetIO CUDA warp-level function `doca_gpu_dev_eth_rxq_receive_warp` .
2. Checks if the packet is an ICMP echo request.

3. Forwards the same packet, modifying some header info (e.g., swapping MAC and IP addresses, changing ICMP packet type).
4. Pushes the modified packet into the send queue using the DOCA GPUNetIO thread-level function `doca_gpu_dev_eth_txq_send_enqueue_strong`.
5. Sends the packet using the DOCA GPUNetIO thread-level functions `doca_gpu_dev_eth_txq_commit_strong` and `doca_gpu_dev_eth_txq_push`.

i Info

This is not a compute intensive use case, so a single CUDA warp with only one receive queue and one send queue is enough to keep up with a decent latency.

By default, the OS CPU ping TTL is set to 64. Therefore, to be sure the GPU is actually replying to ICMP ping requests, TTL is set to 128 in this application.



The following are motivations for this use case:

- Providing an easy tool to check connectivity between packet the generator machine and the DOCA application machine
- Having a sense of network latency between the two machines using a well-known tool like ping
- Showing an easy way to receive and forward modified packets
- Providing a warp-level implementation of a CUDA kernel receiving and forwarding traffic

Assuming the IP address of the network interface to ping is 192.168.1.1, this is the expected output:

```
$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.324 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.332 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.299 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=0.309 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=0.323 ms
64 bytes from 192.168.1.1: icmp_seq=6 ttl=64 time=0.300 ms
64 bytes from 192.168.1.1: icmp_seq=7 ttl=64 time=0.274 ms
64 bytes from 192.168.1.1: icmp_seq=8 ttl=64 time=0.314 ms
64 bytes from 192.168.1.1: icmp_seq=9 ttl=64 time=0.327 ms
64 bytes from 192.168.1.1: icmp_seq=10 ttl=64 time=0.384 ms
# At this point, the DOCA application has been started on the 192.168.1.1 interface
# TTL becomes 128 as it's the GPU replying to ICMP requests now instead of the OS
64 bytes from 192.168.1.1: icmp_seq=11 ttl=128 time=0.346 ms
64 bytes from 192.168.1.1: icmp_seq=12 ttl=128 time=0.274 ms
64 bytes from 192.168.1.1: icmp_seq=13 ttl=128 time=0.294 ms
64 bytes from 192.168.1.1: icmp_seq=14 ttl=128 time=0.240 ms
64 bytes from 192.168.1.1: icmp_seq=15 ttl=128 time=0.273 ms
64 bytes from 192.168.1.1: icmp_seq=16 ttl=128 time=0.238 ms
64 bytes from 192.168.1.1: icmp_seq=17 ttl=128 time=0.252 ms
64 bytes from 192.168.1.1: icmp_seq=18 ttl=128 time=0.232 ms
64 bytes from 192.168.1.1: icmp_seq=19 ttl=128 time=0.278 ms
```

.....

A DOCA Progress Engine is attached to the DOCA Ethernet Txq context used to forward ICMP packets. Those packets are sent from the GPU with the `DOCA_GPU_SEND_FLAG_NOTIFY` flag, which result in creating a notification after every packet is sent by the NIC.

All the notifications are then analyzed by the CPU through the `doca_pe_progress` function. The final effect is the output of the application which returns the distance, in seconds, between two pings. The following is an example with a ping every 0.5 seconds:

```
$ ping -i 0.5 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=128 time=0.202 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=128 time=0.179 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=128 time=0.199 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=128 time=0.180 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=128 time=0.200 ms
64 bytes from 192.168.1.1: icmp_seq=6 ttl=128 time=0.189 ms
.....
```

On the DOCA side, the application should print a log for all the ICMP packets received and retransmitted:

```
Seconds 5
[UDP] QUEUE: 0 DNS: 0 OTHER: 0 TOTAL: 0
[TCP] QUEUE: 0 HTTP: 0 HTTP HEAD: 0 HTTP GET: 0 HTTP POST: 0 TCP [SYN: 0 FIN: 0
ACK: 0] OTHER: 0 TOTAL: 0
[13:54:19:202061][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 3 sent at
1702302859201997120 time from last ICMP is 0.512025 sec
[13:54:19:713960][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 4 sent at
1702302859713896620 time from last ICMP is 0.511899 sec
[13:54:20:225891][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 5 sent at
1702302860225868072 time from last ICMP is 0.511971 sec
```

[13:54:20:737823][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 6 sent at
1702302860737781760 time from last ICMP is 0.511914 sec
[13:54:21:249763][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 7 sent at
1702302861249723044 time from last ICMP is 0.511941 sec
[13:54:21:761614][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 8 sent at
1702302861761588848 time from last ICMP is 0.511866 sec
[13:54:22:273689][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 9 sent at
1702302862273643536 time from last ICMP is 0.512055 sec
[13:54:22:785543][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 10 sent at
1702302862785527576 time from last ICMP is 0.511884 sec
[13:54:23:297545][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 11 sent at
1702302863297501448 time from last ICMP is 0.511974 sec
[13:54:23:809406][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 12 sent at
1702302863809350664 time from last ICMP is 0.511849 sec

Seconds 10

[UDP] QUEUE: 0 DNS: 0 OTHER: 0 TOTAL: 0

[TCP] QUEUE: 0 HTTP: 0 HTTP HEAD: 0 HTTP GET: 0 HTTP POST: 0 TCP [SYN: 0 FIN: 0
ACK: 0] OTHER: 0 TOTAL: 0

[13:54:24:321405][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 13 sent at
1702302864321391148 time from last ICMP is 0.512040 sec
[13:54:24:833338][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 14 sent at
1702302864833270356 time from last ICMP is 0.511879 sec
[13:54:25:345302][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 15 sent at
1702302865345282728 time from last ICMP is 0.512012 sec

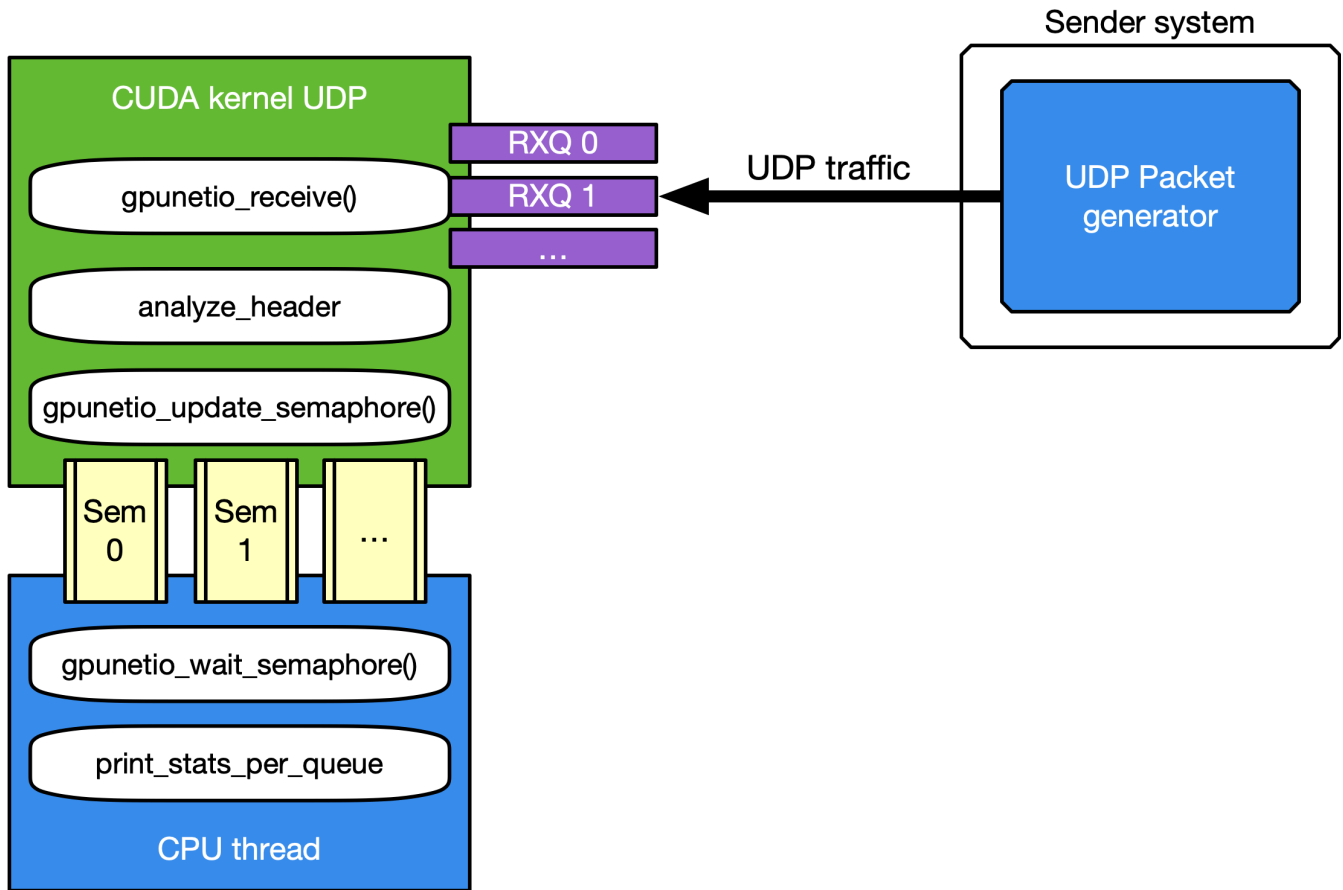
```
[13:54:25:857199][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 16 sent at
1702302865857133664 time from last ICMP is 0.511851 sec
[13:54:26:369131][2688665][DOCA][INF][gpu_packet_processing.c:77]
[debug_send_packet_icmp_cb] ICMP debug event: Queue 0 packet 17 sent at
1702302866369128728 time from last ICMP is 0.511995 sec.....
```

UDP Network Traffic

This is the most generic use case of receive-and-analyze packet headers. Designed to keep up with 100Gb/s of incoming network traffic, the CUDA kernel responsible for the UDP traffic dedicates one CUDA block of 512 CUDA threads (file `gpu_kernels/receive_udp.cu`) to a different Ethernet UDP receive queue.

The data path loop is:

1. Receive packets using the DOCA GPUNetIO CUDA block-level function `doca_gpu_dev_eth_rxq_receive_block`.
2. Each CUDA thread works on a subset of received packets.
3. DOCA buffer containing the packet is retrieved.
4. Packet payload is analyzed to differentiate between DNS packets from other UDP generic packets.
5. Packet payload is wiped-out to ensure that old stale packets are not analyzed again.
6. Each CUDA block reports to the CPU thread statistics about types of received packets through a DOCA GPUNetIO semaphore.
7. CPU thread polls on semaphores to retrieve and print the statistics to the console.



The motivation for this use case is mostly to provide an application template to:

- Receive and analyze packet headers to differentiate across different UDP protocols
- Report statistics to the CPU through the DOCA GPUNetIO semaphore

Several well-known packet generators can be used to test this mode like T-Rex or DPDK testpmd.

TCP Network Traffic and HTTP Echo Server

By default, the TCP flow management is the same as UDP: Receive TCP packets and analyze their headers to report to the CPU statistics about the types of received packets. This is good for passive traffic analyzers or sniffers but sometimes a packet processing application requires receiving packets directly from TCP peers which implies the establishment of a TCP-reliable connection through the 3-way handshake method. Therefore, it is possible to enable TCP "server" mode through the `-s` command-line flag

which enables an "HTTP echo server" mode where the CPU and GPU cooperate to establish a TCP connection and process TCP data packets.

Specifically, in this case there are two different sets of receive queues:

- CPU DPDK receive queues which receive TCP "control" packets (e.g. SYN, FIN or RST)
- DOCA GPUNetIO receive queues to receive TCP "data" packets

This distinction is possible thanks to DOCA Flow capabilities.

The application's flow requires CPU and GPU collaboration as described in the following subsections.

Step 1: TCP Connection Establishment

A CPU thread through DPDK queues receives a TCP SYN packet from a remote TCP peer. The CPU thread establishes a TCP reliable connection (replies with a TCP SYN-ACK packet) with the peer and uses DOCA Flow to create a new steering rule to redirect TCP data packets to one of the DOCA GPUNetIO receive queues. The new steering rule excludes control packets (e.g., SYN, FIN or RST).

Step 2: TCP Data Processing

The CUDA kernel responsible for TCP processing receives TCP data packets and performs TCP packet header analysis. If it receives an HTTP GET request, it stores the relevant packet's info in the next item of a DOCA GPUNetIO semaphore, setting it to `READY`.

Step 3: HTTP Echo Server

A second CUDA kernel responsible for HTTP processing polls the DOCA GPUNetIO semaphore. Once it detects the update of the next item to `READY`, it reads the HTTP GET packet info and crafts an HTTP response packet with an HTML page.

If the request is about `index.html` or `contacts.html`, the CUDA kernel replies with the appropriate HTML page using a 200 OK code. For all other requests, the it returns a "Page not found" and 404 Error code.

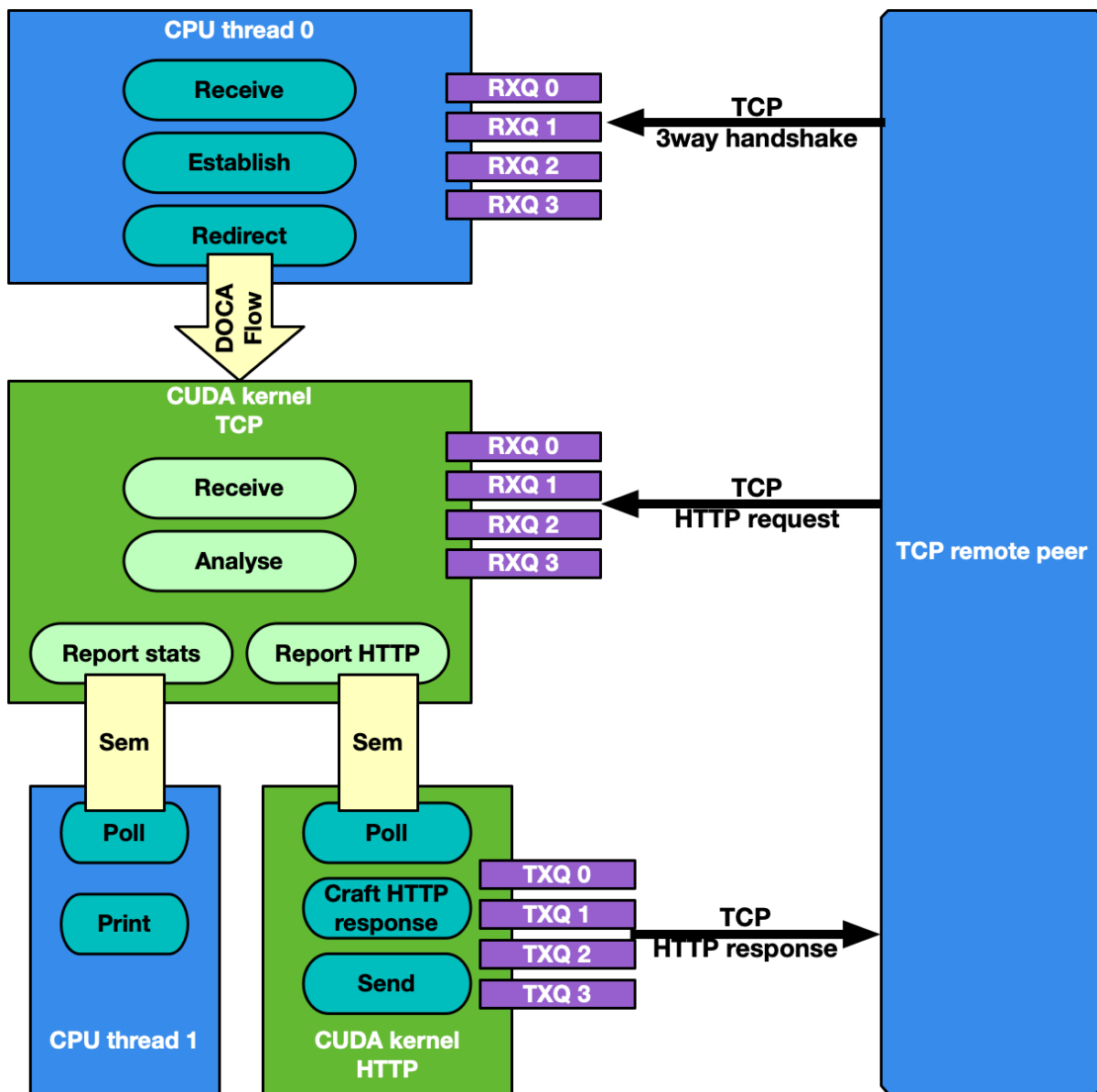
HTTP response packets are sent by this second HTTP CUDA kernel using DOCA GPUNetIO.

Note

Care must be taken to maintain TCP sequence/ack numbers in the packet headers.

Step 4: TCP Connection Closure

If the CPU receives a TCP FIN packet through the DPDK queues, it closes the connection with the remote TCP peer and removes the DOCA Flow rule from the DOCA GPUNetIO queues so the CUDA kernel cannot receive anymore packets from that TCP peer.



Motivations for this use case:

- Receiving and analyzing packet headers to differentiate across different TCP protocols
- Processing TCP packets on GPU in passive mode (sniffing) and active mode (reliable connection)
- Having a DOCA-DPDK application able to establish a TCP reliable connection without using any OS socket and bypassing kernel routines
- Having CUDA-kernel-to-CUDA-kernel communication through a DOCA GPUNetIO semaphore

- Showing how to create and send a packet from scratch with DOCA GPUNetIO

Assuming the network interface used to run the application has the IP address 192.168.1.1 , it is possible to test this HTTP echo server mode using simple tools like curl or wget.

Example with curl:

```
$ curl http://192.168.1.1/index.html -ivvv
* Trying 192.168.1.1:80...
* Connected to 192.168.1.1 (192.168.1.1) port 80 (#0)
> GET /index.html HTTP/1.1
> Host: 192.168.1.1
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< Date: Sun, 30 Apr 2023 20:30:40 GMT
Date: Sun, 30 Apr 2023 20:30:40 GMT
< Content-Type: text/html; charset=UTF-8
Content-Type: text/html; charset=UTF-8
< Content-Length: 158
Content-Length: 158
< Last-Modified: Sun, 30 Apr 2023 22:38:34 GMT
Last-Modified: Sun, 30 Apr 2023 22:38:34 GMT
< Server: GPUNetIO
Server: GPUNetIO
< Accept-Ranges: bytes
Accept-Ranges: bytes
< Connection: keep-alive
Connection: keep-alive
< Keep-Alive: timeout=5
Keep-Alive: timeout=5
```

```
<
<html>
<head>
<title>GPUNetIO index page</title>
</head>
<body>
<p>Hello World, the GPUNetIO server Index page!</p>
</body>
</html>
```

```
* Connection #0 to host 192.168.1.1 left intact
```

DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA GPUNetIO](#)
- [DOCA Ethernet](#)
- [DOCA Flow](#)

Refer to their respective programming guide for more information on system configuration and requirements.

Refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install DOCA package software.

Dependencies

Before running the application you need to be sure you have the following:

- `gdrdrv` kernel module – active and running on the system
- `nvidia-peermem` kernel module – active and running on the system
- Network card interface you want to use is up

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/gpu_packet_processing/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_gpu_packet_processing is created under
/tmp/build/gpu_packet_processing/.

Compiling Only the Current Application

To directly build only the GPU packet processing application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -  
Denable_gpu_packet_processing=true  
ninja -C /tmp/build
```

Info

doca_gpu_packet_processing is created under
/tmp/build/gpu_packet_processing/.

Alternatively, users can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
 - o Set enable_all_applications to false
 - o Set enable_gpu_packet_processing to true
2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_gpu_packet_processing is created under
/tmp/build/gpu_packet_processing/.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

The GPU packet processing application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_gpu_packet_processing [DOCA Flags] [Program Flags]
```

DOCA Flags:

-h, --help Print a help synopsis

-v, --version Print program version information

-l, --log-level Set the (numeric) log level for the program <10=DISABLE,
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE,
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

-j, --json <path> Parse all command flags from an input json file

Program Flags:

-g, --gpu <GPU PCIe address> GPU PCIe address to be used by the app
-n, --nic <NIC PCIe address> DOCA device PCIe address used by the app
-q, --queue <GPU receive queues> DOCA GPUNetIO receive queue per flow
-s, --httpserver <Enable GPU HTTP server> Enable GPU HTTP server mode

Info

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_gpu_packet_processing -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the host:

1. Assuming a GPU PCIe address ca:00.0 and NIC PCIe address 17:00.0 with 2 GPUNetIO receive queues:

```
./doca_gpu_packet_processing -n 17:00.0 -g ca:00.0 -q 2
```

Note

Refer to section "Running DOCA Application on Host" in the [NVIDIA DOCA Virtual Functions User Guide](#).

Command Line Flags

Flag Type	Short Flag	Long Flag	Description
General flags	h	help	Prints a help synopsis
	v	version	Prints program version information
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with TRACE log level support)
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70
	j	json	Parse all command flags from an input JSON file
Program flags	g	gpu	GPU PCIe address in <bus>:<device>.<function> format. This can be obtained using the nvidia-smi or lspci commands.
	n	nic	Network card port PCIe address in <bus>:<device>.<function> format. This can be obtained using the lspci command.

Flag Type	Short Flag	Long Flag	Description
	q	queue	Number of receive queues to use in the example. Default is 1, maximum allowed is 4.
	s	https server	Enable the TCP HTTP server mode. With this flag, TCP packets are not received by GPUNetIO as regular sniffer as it requires a TCP 3-way handshake to establish a reliable connection first.

Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications .

Application Code Flow

The following explains the application's flow, highlighting main code blocks and functions:

1. Parse application argument.

```

doca_argp_init();
register_application_params();
doca_argp_start();

```

2. Initialize network device as DOCA device, initialize DPDK, and get device DPDK port ID.

```
init_doca_device();
```

Calls `rte_eal_init()` with empty flags to initialize EAL resources.

3. Initialize a GPU device, creating a DOCA GPUNetIO handle for it.

```
doca_gpu_create();
```

4. Initialize DOCA Flow, starting the DPDK port.

```
init_doca_flow();
```

Flags to initialize DOCA Flow are VNF, HW steering, and isolated mode (to prevent the default RSS flows from interfering with the GPUNetIO queues).

5. Create RX and TX queue related objects (i.e., Ethernet handlers, GPUNetIO handlers, flow rules, semaphores) to manage UDP, TCP and ICMP flows.

```
create_udp_queues();
create_tcp_queues();
create_icmp_queues();
/* Depending on TCP mode (HTTP server or not) properly connect different DOCA Flow pipes */
create_root_pipe();
```

6. Allocate generic exit flag. All CUDA kernels periodically poll on this flag. If the CPU set it to 1, CUDA kernels exit from their main loop and return.

```
doca_gpu_mem_alloc(gpu_dev, sizeof(uint32_t), alignment,
DOCA_GPU_MEM_GPU_CPU, (void **)&gpu_exit_condition, (void
**)&cpu_exit_condition);
```

7. Launch CUDA kernels, each on a different stream.

```

kernel_receive_udp(rx_udp_stream, gpu_exit_condition, &udp_queues);
kernel_receive_tcp(rx_tcp_stream, gpu_exit_condition, &tcp_queues,
app_cfg.http_server);
kernel_receive_icmp(rx_icmp_stream, gpu_exit_condition, &icmp_queues);
if (app_cfg.http_server)
kernel_http_server(tx_http_server, gpu_exit_condition, &tcp_queues,
&http_queues);

```

8. Launch the CPU thread responsible to poll on DOCA GPUNetIO semaphores and print UDP and TCP stats on the console.

```

rte_eal_remote_launch((void *)stats_core, NULL, current_lcore);

```

9. Launch CPU thread responsible for managing TCP 3-way handshake connections.

```

if (app_cfg.http_server) {
...
rte_eal_remote_launch(tcp_cpu_rss_func, &tcp_queues, current_lcore);
}

```

10. Wait for the user to send a signal to quit the application. When this happens, the signal handler function sets the `force_quit` flag to true which causes the main thread to move forward and set the exit condition to 1.

```

while (DOCA_GPUNETIO_VOLATILE(force_quit) == false);
DOCA_GPUNETIO_VOLATILE(*cpu_exit_condition) = 1;

```

11. Wait for CUDA kernels to exit and finalize all DOCA Flow and GPUNetIO resources.

```

cudaStreamSynchronize(rx_udp_stream);
cudaStreamSynchronize(rx_tcp_stream);
cudaStreamSynchronize(rx_icmp_stream);
if (app_cfg.http_server)
cudaStreamSynchronize(tx_http_server);

```

```
destroy_flow_queue();  
doca_gpu_destroy();
```

References

- /opt/mellanox/doca/applications/gpu_packet_processing/

NVIDIA DOCA IPsec Security Gateway Application Guide

This document provides an IPsec security gateway implementation on top of NVIDIA® BlueField® DPU.

Note

If your target application utilizes 100Gb/s or higher bandwidth, where a substantial part of the bandwidth is allocated for IPsec traffic, please refer to the *NVIDIA BlueField-2 DPUs Product Release Notes* to learn about a potential bandwidth limitation. To access the relevant product release notes, please contact your NVIDIA sales representative.

Introduction

Note

DOCA IPsec Security Gateway is supported at alpha level.

DOCA IPsec Security Gateway leverages the DPU's hardware capability for secure network communication. The application demonstrates how to insert rules related to IPsec encryption and decryption based on the [DOCA Flow](#) library.

The application demonstrates how to insert rules to create an IPsec tunnel.

(i) Note

An example for configuring the Internet Key Exchange (IKE) can be found under section "[Keying Daemon Integration \(StrongSwan\)](#)" but is not considered part of the application.

The application can be configured to receive IPsec rules in one of the following ways:

- Static configuration – (default) receives a fixed list of rules for IPsec encryption and decryption

(i) Note

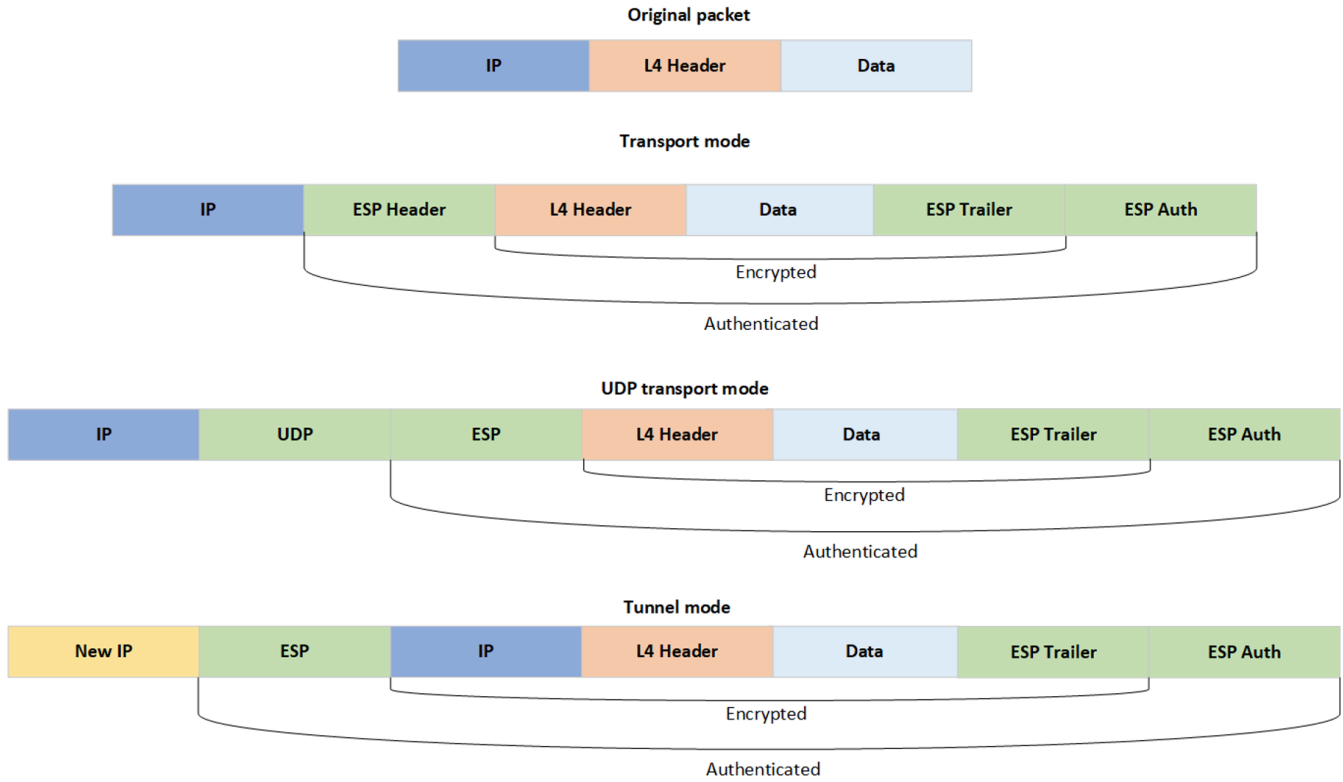
When creating the security association (SA) object, the application gets the key, salt, and other SA attributes from the JSON input file.

- Dynamic configuration – receives IPsec encryption and decryption rules during runtime through a Unix domain socket (UDS) which is enabled when providing a socket path to the application

(i) Note

You may find an example of integrating a rules generator with the application under strongSwan project ([DOCA plugin](#)).

The application supports the following IPsec modes: Tunnel, transport, UDP transport.

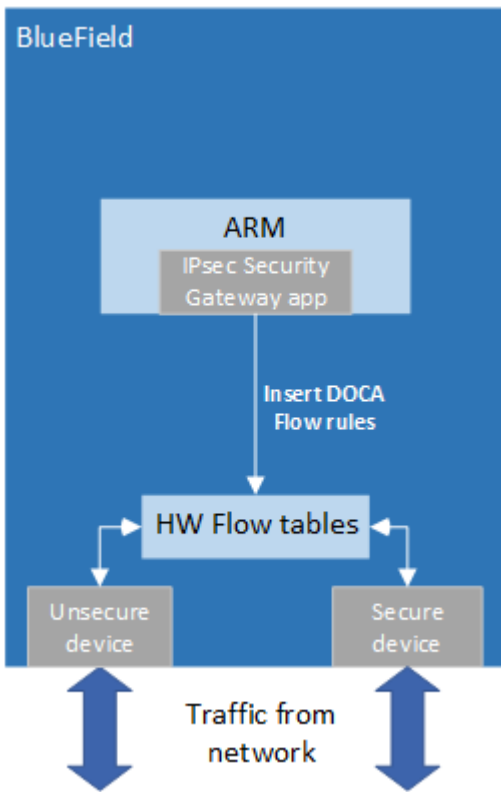


System Design

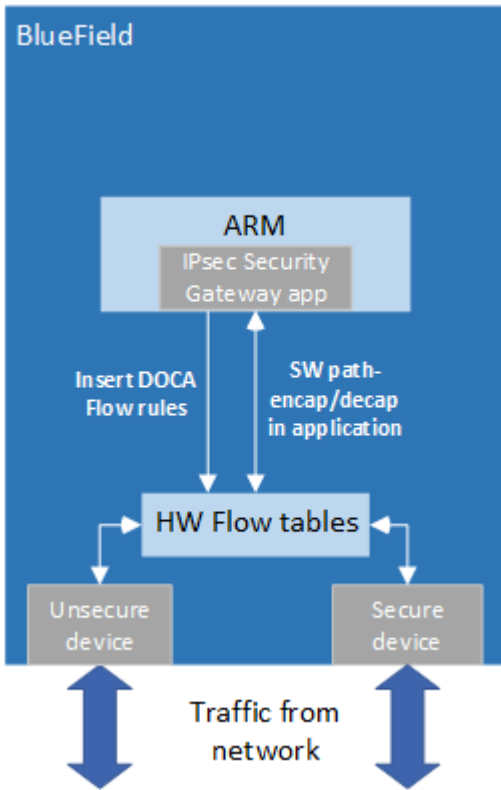
DOCA IPsec Security Gateway is designed to run with 2 ports, secured and unsecured:

- Secured port – BlueField receives IPsec encrypted packets and, after decryption, they are sent through the unsecured port
- Unsecured port – BlueField receives regular (plain text) packets and, after encryption, they are sent through the secured port

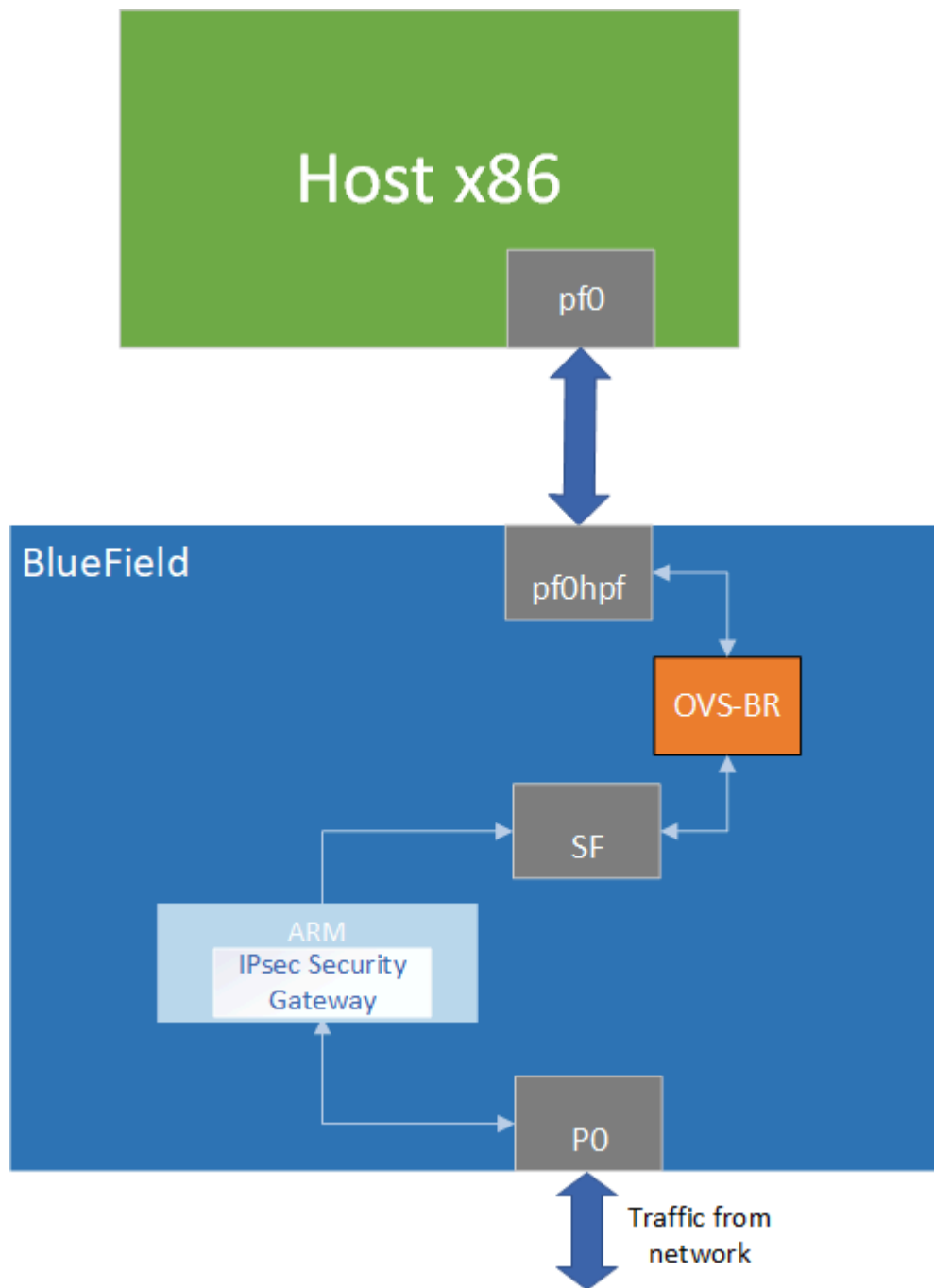
Example packet path for hardware (HW) offloading:



Example packet path for partial software processing (handling encap/decap in software):



Using the application with SF:

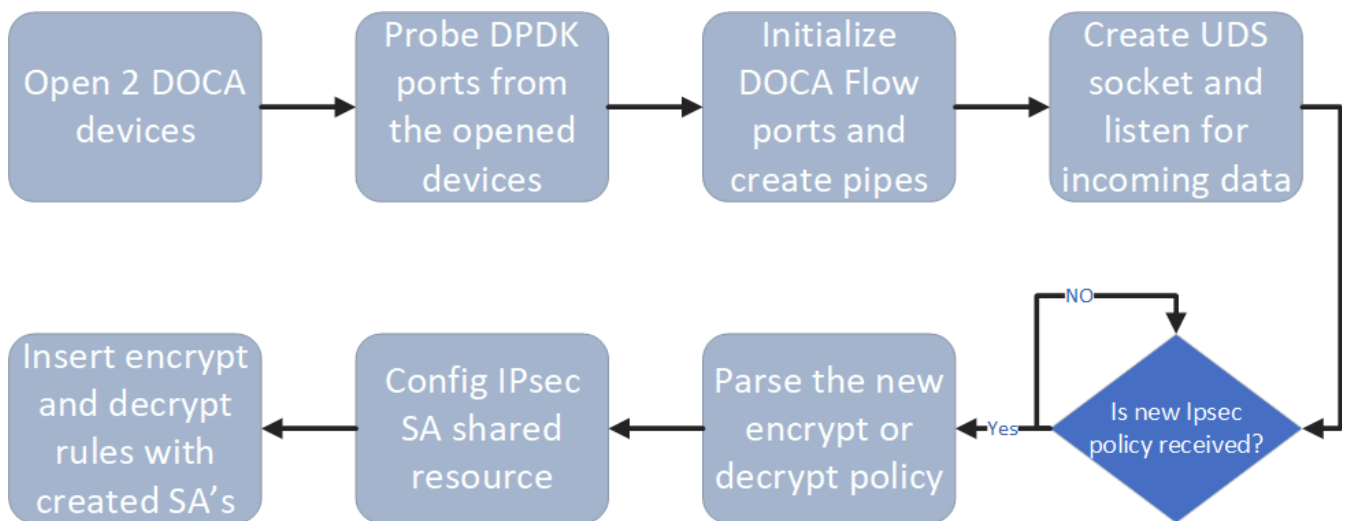


Application Architecture

Static Configuration

1. Open two DOCA devices, one for the secured port and another for the unsecured port.
2. With the open DOCA devices, the application probes DPDK ports and initializes DOCA Flow and DOCA Flow ports accordingly.
3. On the created ports, build DOCA Flow pipes.
4. In a loop according to the JSON rules:
 1. Create IPsec SA shared resource for the new rule.
 2. Insert encrypt or decrypt rule to DOCA Flow pipes.

Dynamic Configuration



1. Open two DOCA devices, one for the secured port and another for the unsecured port.
2. With the open DOCA devices, the application probes DPDK ports and initializes DOCA Flow and DOCA Flow ports accordingly.
3. On the created ports, build DOCA Flow pipes.
4. Create UDS socket and listen for incoming data.

5. While waiting for new IPsec policies to be received in a loop, if a new IPsec policy is received:

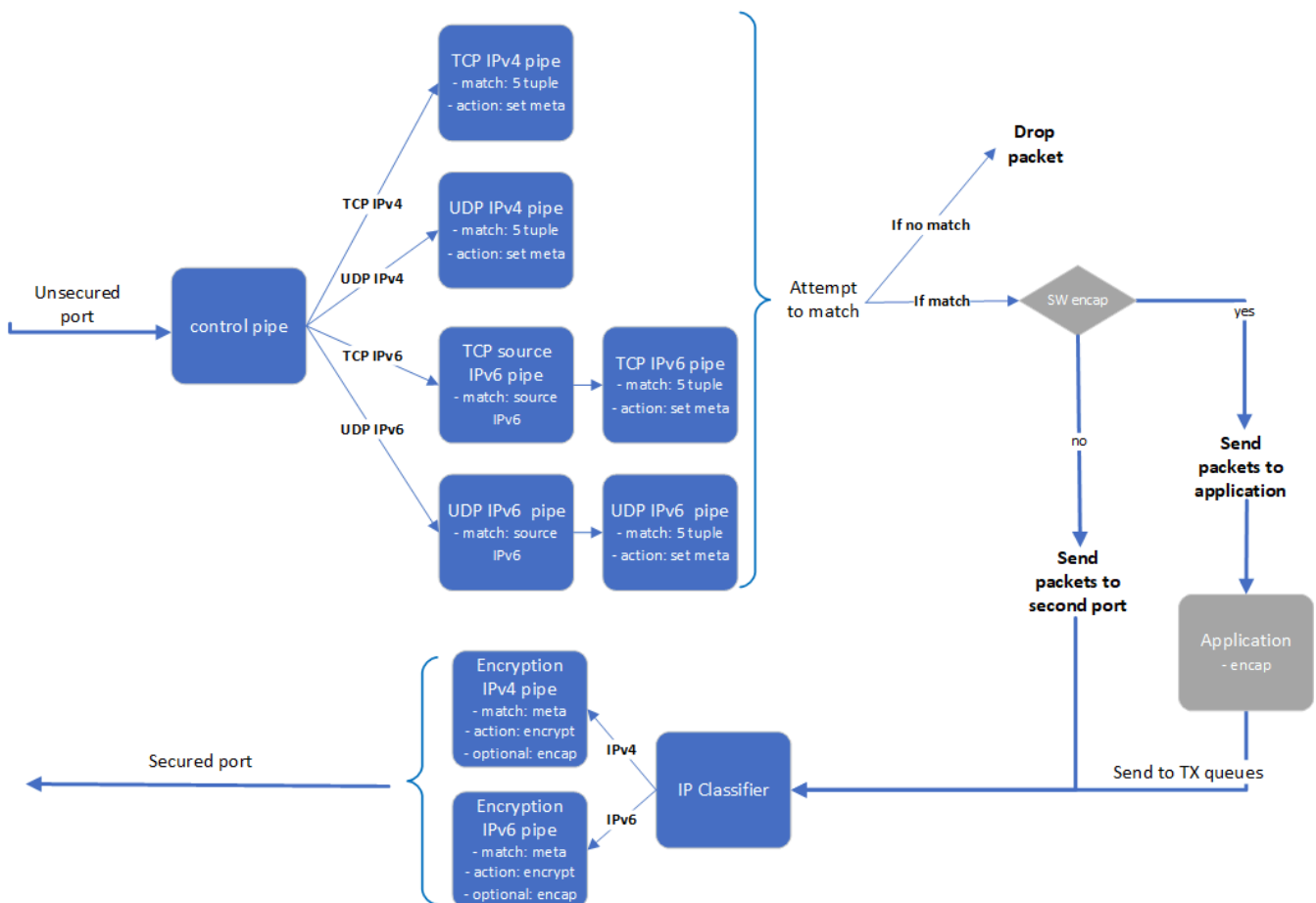
1. Parse the policy whether it is an encryption or decryption rule.
2. Create IPsec SA shared resource for the new rule.
3. Insert encrypt or decrypt rule to DOCA Flow pipes.

DOCA Flow Modes

The application can run in two modes, vnf and switch. For more information about the modes, please refer to "Pipe Mode" in the [DOCA Flow](#).

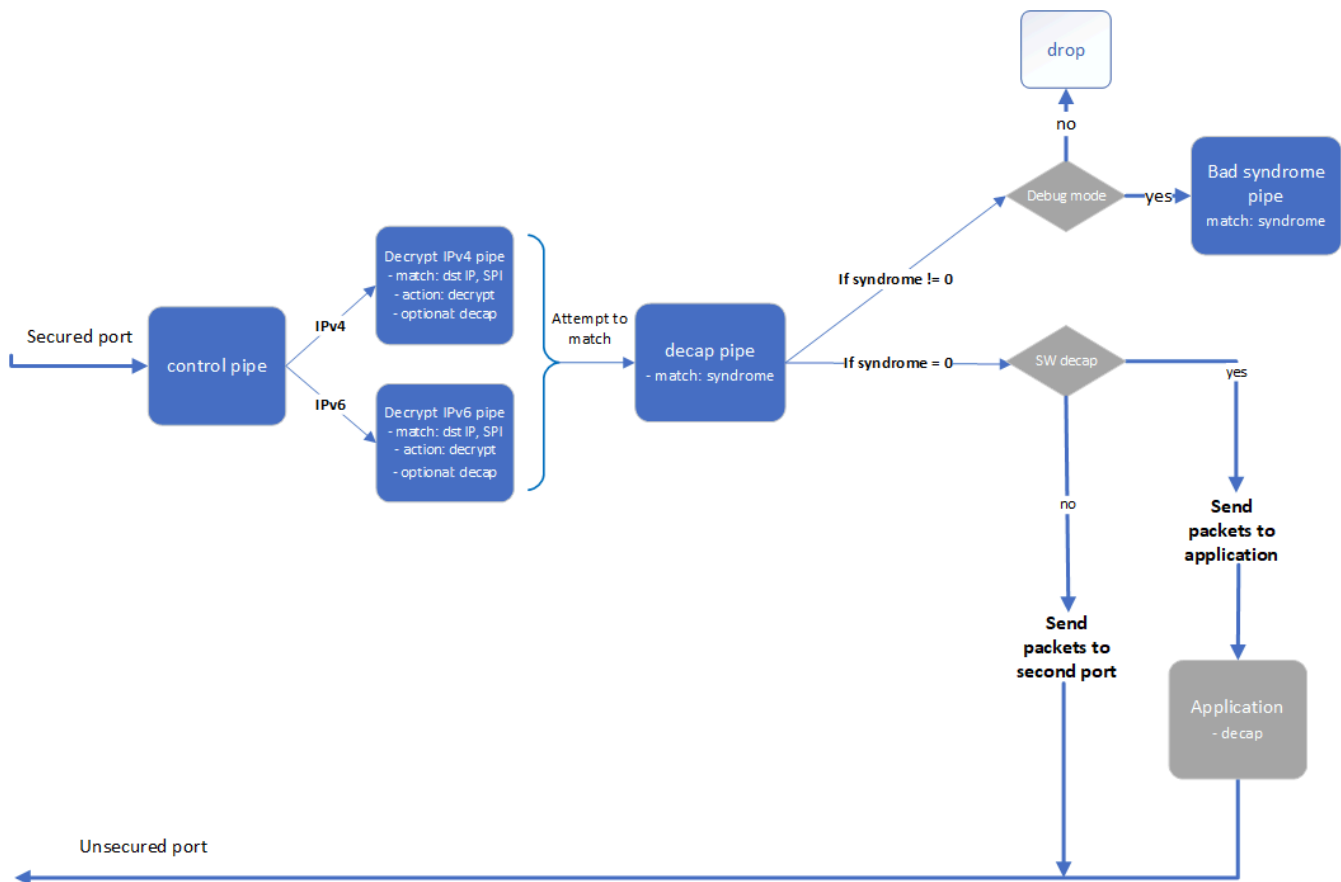
VNF Mode

Encryption



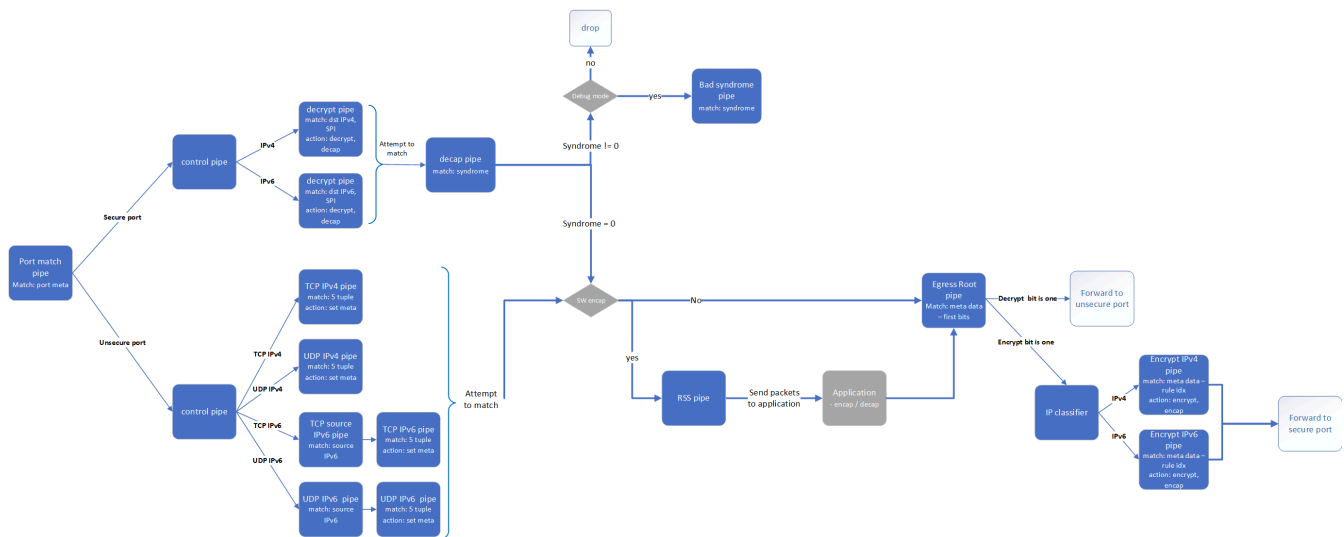
- The application builds pipes for encryption. Control pipe as root with four entries that match L3 and L4 types and forward the traffic to the relevant pipes.
 - IPv6 pipes – match the source IP address and forward the traffic to a pipe that matches 5-tuple excluding the source IP.
 - In the 5-tuple match pipes set action of "set meta data", the metadata would be the rule's index in the JSON file.
 - The matched packet is forwarded to the second port.
- In the secured egress domain, the IP classifier pipe sends the packets to the correct encryption pipe (IPv4 or IPv6) which has a shared IPsec encrypt action. According to the metadata match, the packet is encrypted with the encap destination IP and SPI as defined in the user's rules.

Decryption



1. The application builds pipes for decryption. Control pipe as root with two entries that match L3 type and forward the traffic to the relevant decrypt pipe.
2. The decrypt pipe matches the destination IP and SPI according to the rule files and has a shared IPsec action for decryption.
3. After decryption, the matched packets are forwarded to the decap pipe and, if the syndrome is non-zero, the packets are dropped. Otherwise, the packets decap the ESP header and forward to the second port.
 1. In debug mode, if syndrome is non-zero, then it sends to bad syndrome pipe to match on the syndrome, count and drop/send to application.

Switch Mode



In switch mode, an ingress root pipe matches the source port to decide what the next pipe is:

- Based on the port, the packet passes through almost the same path as VNF mode and the metadata is set. Afterwards, the packet moves to egress root pipe.

In egress root pipe, the match is on encrypt and decrypt bits that were set in the packet meta:

- Decrypt bit is 1 – packet finishes the decrypt path and must be sent to the unsecure port
- Encrypt bit is 1 – packet almost finishes the encrypt path and must be sent to the encrypt pipe on the secure egress domain and to the secure port from there

DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA Flow](#)

Refer to their respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/ipsec_security_gw/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

`doca_ipsec_security_gw` is created under `/tmp/build/ipsec_security_gw/`.

Compiling Only the Current Application

To directly build only the IPsec Security Gateway application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_ipsec_security_gw=true  
ninja -C /tmp/build
```

Info

doca_ipsec_security_gw is created under /tmp/build/ipsec_security_gw/.

Alternatively, users can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
 - Set enable_all_applications to false
 - Set enable_ipsec_security_gw to true
2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_ipsec_security_gw is created under /tmp/build/ipsec_security_gw/.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Prerequisites

1. The IPsec security gateway application is based on DOCA Flow. Therefore, the user is required to allocate huge pages.

```
echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

Note

On some operating systems (RockyLinux, OpenEuler, CentOS 8.2) the default huge page size on the DPU (and Arm hosts) is larger than 2MB, and is often 512MB instead. One can find out the size of the huge pages using the following command:

```
$ grep -i huge /proc/meminfo
```

```
AnonHugePages: 0 kB  
ShmemHugePages: 0 kB  
FileHugePages: 0 kB  
HugePages_Total: 4  
HugePages_Free: 4  
HugePages_Rsvd: 0  
HugePages_Surp: 0  
Hugepagesize: 524288 kB  
Hugetlb: 6291456 kB
```

Given that the guiding principal is to allocate 4GB of RAM, in such cases instead of allocating 2048 pages, one should allocate the matching amount (8 pages):

```
echo '8' | sudo tee -a
/sys/kernel/mm/hugepages/hugepages-
524288kB/nr_hugepages
```

2. VNF mode – the IPsec security gateway application requires disabling some of the hardware tables:

```
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode
legacy
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode
legacy

echo none > /sys/class/net/p0/compat/devlink/encap
echo none > /sys/class/net/p1/compat/devlink/encap

/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode
switchdev
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode
switchdev
```

To restore the old configuration:

```
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode
legacy
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode
legacy


echo basic > /sys/class/net/p0/compat/devlink/encap
echo basic > /sys/class/net/p1/compat/devlink/encap

/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode
switchdev
```

```
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode  
switchdev
```

3. Switch mode – the IPsec security gateway application requires configuring the ports to run in switch mode:

```
sudo mlxconfig -d /dev/mst/mt41686(mt41692)_pciconf0 s  
LAG_RESOURCE_ALLOCATION=1  
# power cycle the host to apply this setting  
  
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode  
legacy  
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode  
legacy  
  
sudo devlink dev param set pci/0000:03:00.0 name esw_pet_insert value false  
cmode runtime  
sudo devlink dev param set pci/0000:03:00.1 name esw_pet_insert value false  
cmode runtime  
  
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode  
switchdev  
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode  
switchdev  
  
sudo devlink dev param set pci/0000:03:00.0 name esw_multiport value true  
cmode runtime  
sudo devlink dev param set pci/0000:03:00.1 name esw_multiport value true  
cmode runtime
```

 **Note**

Make sure to perform graceful shutdown prior to power cycling the host.

To restore the old configuration:

```
sudo devlink dev param set pci/0000:03:00.0 name esw_multiport value false
cmode runtime
sudo devlink dev param set pci/0000:03:00.1 name esw_multiport value false
cmode runtime
```

Application Execution

The IPsec Security Gateway application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_ipsec_security_gw [DOCA Flags] [Program Flags]
```

DOCA Flags:

```
-h, --help Print a help synopsis
-v, --version Print program version information
-l, --log-level Set the (numeric) log level for the program <10=DISABLE,
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE,
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
-j, --json <path> Parse all command flags from an input json file
```

Program Flags:

```
-s, --secured secured port pci-address
-u, --unsecured unsecured port pci-address
-c, --config Path to the JSON file with application configuration
```

```
-m, --mode ipsec mode - {tunnel/transport/udp_transport}  
-i, --ipc IPC socket file path  
-sn, --secured-name secured port interface name  
-un, --unsecured-name unsecured port interface name  
-n, --nb-cores number of cores  
--debug Enable debug counters
```

Info

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_ipsec_security_gw -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the BlueField or host:

- o Static Configuration:

```
./doca_ipsec_security_gw -s 03:00.0 -u 03:00.1 -c  
./ipsec_security_gw_config.json -m transport
```

Note

Both the PCIe address identifiers (`-s` and `-u` flags) should match the addresses of the desired PCIe devices.

- o Dynamic Configuration:

```
./doca_ipsec_security_gw -s 03:00.0 -u 03:00.1 -c  
./ipsec_security_gw_config.json -m transport -i /tmp/rules_socket
```

Note

Both the PCIe address identifiers (`-s` and `-u` flags) should match the addresses of the desired PCIe devices.

3. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_ipsec_security_gw --json [json_file]
```

For example

```
./doca_ipsec_security_gw --json ipsec_security_gw_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with TRACE log level support) 	<code>"log-level": 60</code>
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	<code>"sdk-log-level": 40</code>
	j	json	Parse all command flags from an input json file	N/A
Program flags	c	config	Path to JSON file with configurations	<code>"config": "security_gateway_config.json"</code>
	u	unsecured	PCIe address for the unsecured port	<code>"unsecured":</code>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
				"03:00.1"
	s	secured	PCIe address for the secured port	"secured": "03:00.0"
	m	mode	IPsec mode. Possible values: tunnel, transport, udp_transport	"mode": "tunnel"
	un	unsecured-name	Interface name of the unsecured port	"unsecured-name": "p1"
	sn	secured-name	Interface name of the secured port	"secured-name": "p0"
	i	ipc	IPC socket file path for receiving IPsec rules during runtime	"ipc": "/tmp/rules_socket"
	n	nb-cores	Number of cores	"nb-cores": 8
	N/A	debug	Add counters to all the entries	"debug": true

Info


Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Static Configuration IPsec Rules

IPsec rules and other configuration can be added with a JSON config file which is passed using the --config parameter.

Section	Field	Type	Description	Example
config	switch	boolean	Configures whether DOCA Flow runs in VNF (false) or switch (true) mode	<pre>"switch": true</pre>
	esp-header-offload	string	Decap and encap offloading: both, encap, decap, or none. Default is both (offloading both encap and decap).	<pre>"esp-header-offload": "none"</pre>
	sw-sn-inc-enable	boolean	<p>Increments sequence number of ESP in software if set to true. Default is false.</p> <p>Note Available only if esp_header_offload is decap OR none.</p>	<pre>"sw-sn-inc-enable": true</pre>
	sw-antireplay-enable	boolean	<p>Enables anti-replay mechanism in software if set to true. Default is false.</p> <p>Note Available only if esp_header_offload is encap OR none.</p> <p>Note Window size is 64. Not ESN. Supports non-zero sn_initial.</p>	<pre>"sw-antireplay-enable": true</pre>

Section	Field	Type	Description	Example
	sn-initial	uint	Initial sequence number for ESP header. Used also when <code>sw_antireplay_enable</code> is true. Default is 0.	<code>"sn-initial": 0</code>
	debug	boolean	Set debug counter for all entries when true. Default is false. This parameter is also used from CLI, will be taken as true if was sent in one of them.	<code>"debug": false</code>
	fwd-bad-syndrome	string	Forward packets that has bad syndrome: drop, RSS. Default is drop. Note Only available in debug mode.	<code>"fwd-bad-syndrome": "drop"</code>
	perf-measurements	string	Possible values: none, insertion-rate, bandwidth, both. Default is none. <ul style="list-style-type: none">insertion-rate – print the total time it took to add the entriesbandwidth – optimize the pipe to improve pps for IPv6	<code>"perf-measurements": "both"</code>
encrypt_rules	ip-version	int	Source and destination IP version. Possible values: 4, 6. Optional; default is 4.	<code>"ip-version": 6</code>
	src-ip	string	Source IP to match	<code>"src-ip": "1.2.3.4"</code>
	dst-ip	string	Destination IP to match	<code>"dst-ip": "101:101:101:101:101:101:101:101"</code>
	protocol	string	L4 protocol: TCP or UDP	<code>"protocol"</code>

Section	Field	Type	Description	Example
		g		
	src-port	int	Source port to match	
	dst-port	int	Destination port to match	<code>"dst-port": 55</code>
	encap-ip-version	int	Encap IP version: 4 or 6. Optional; default is 4.	<code>"ip-version": 4</code>
	encap-dst-ip	string	Encap destination IP <div style="background-color: #ffffcc; padding: 5px; border: 1px solid #ccc;"> <p> Note Mandatory for tunnel mode only.</p> </div>	<code>"encap-dst-ip": "1.1.1.1"</code>
	spi	int	SPI integer to set in the ESP header	<code>"spi": 5</code>
	key	string	Key for creating the SA (in hex format)	<code>"key": "112233445566778899aabbccdd"</code>
	key-type	int	Key size: 128 or 256. Optional; default is 256.	<code>"key-type": 128</code>
	salt	int	Salt value for creating the SA. Default is 6.	<code>"salt": 1212</code>
	icv-length	int	ICV length value: 8, 12, or 16. Default is 16.	<code>"icv-length": 12</code>
decrypt_rules	ip-version	int	Destination IP version: 4 or 6. Optional; default is 4.	<code>"ip-version": 6</code>
	dst-ip	string	Destination IP to match	<code>"dst-ip": "1122:3344:5566:</code>

Section	Field	Type	Description	Example
		g		7788:99aa:bbcc:dee:ff00"
	inner-ip-version	int	Inner IP version: 4 or 6. Optional; default is 4. <div style="background-color: #ffffcc; padding: 5px;"> <p>Note Mandatory for tunnel mode only.</p> </div>	"inner-ip-version": 4
	spi	int	SPI to match in the ESP header	"spi": 5
	key	string	Key for creating the SA (in hex format)	"key": "112233445566778899aabbccdd"
	key-type	int	Key size: 128 or 256. Optional; default is 256.	"key-type": 128
	salt	int	Salt value for creating the SA. Default is 6.	"salt": 1212
	icv-length	int	ICV length value: 8, 12, or 16. Default is 16.	"icv-length": 12

Dynamic Configuration IPsec Rules

The application listens on the UDS socket for receiving a predefined structure for the IPsec policy defined in the `policy.h` file.

The client program or keying daemon should connect to the socket with the same socket file path provided to the application by the `--ipc/-i` flags, and send the policy structure as packed to the application through the same socket.

Note

In the dynamic configuration, the application uses the `config` section from the JSON config file and ignores the `encrypt_rules` and `decrypt_rules` sections.

The IPsec policy structure:

```
struct ipsec_security_gw_ipsec_policy {
    /* Protocols attributes */
    uint16_t src_port; /* Policy inner source port */
    uint16_t dst_port; /* Policy inner destination port */
    uint8_t l3_protocol; /* Policy L3 proto {POLICY_L3_TYPE_IPV4, POLICY_L3_TYPE_IPV6} */
    uint8_t l4_protocol; /* Policy L4 proto {POLICY_L4_TYPE_UDP, POLICY_L4_TYPE_TCP} */
    uint8_t outer_l3_protocol; /* Policy outer L3 type {POLICY_L3_TYPE_IPV4, POLICY_L3_TYPE_IPV6} */

    /* Policy attributes */
    uint8_t policy_direction; /* Policy direction {POLICY_DIR_IN, POLICY_DIR_OUT} */
    uint8_t policy_mode; /* Policy IPSEC mode {POLICY_MODE_TRANSPORT, POLICY_MODE_TUNNEL}
    */

    /* Security Association attributes */
    uint8_t esn; /* Is ESN enabled? */
    uint8_t icv_length; /* ICV length in bytes {8, 12, 16} */
    uint8_t key_type; /* AES key type {POLICY_KEY_TYPE_128, POLICY_KEY_TYPE_256} */
    uint32_t spi; /* Security Parameter Index */
    uint32_t salt; /* Cryptographic salt */
    uint8_t enc_key_data[MAX_KEY_LEN]; /* Encryption key (binary) */

    /* Policy inner and outer addresses */
    char src_ip_addr[MAX_IP_ADDR_LEN + 1]; /* Policy inner IP source address in string format */
    char dst_ip_addr[MAX_IP_ADDR_LEN + 1]; /* Policy inner IP destination address in string format
    */
    char outer_src_ip[MAX_IP_ADDR_LEN + 1]; /* Policy outer IP source address in string format */
};
```

```
char outer_dst_ip[MAX_IP_ADDR_LEN + 1]; /* Policy outer IP destination address in string
format */
};
```

Note

The policy type, whether it is encrypted or decrypted, is classified according to the `policy_direction` attribute:

- `POLICY_DIR_IN` – decryption policy
- `POLICY_DIR_OUT` – encryption policy

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications .

Application Code Flow

1. Parse application argument.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register the application's parameters.

```
register_ipsec_security_gw_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

1. Parse app parameters.

2. DPDK initialization.

```
rte_eal_init();
```

Call `rte_eal_init()` to initialize EAL resources with the provided EAL flags for not probing the ports.

3. Parse config file.

```
ipsec_security_gw_parse_config();
```

4. Initialize devices and ports.

```
ipsec_security_gw_init_devices();
```

1. Open DOCA devices with input PCIe addresses / interface names.

2. Probe DPDK port from each opened device.

5. Initialize and start DPDK ports.

```
dpdk_queues_and_ports_init();
```

1. Initialize DPDK ports, including mempool allocation.

2. Initialize hairpin queues if needed.

3. Binds hairpin queues of each port to its peer port.

6. Initialize DOCA Flow.

```
ipsec_security_gw_init_doca_flow();
```

1. Initialize DOCA Flow library.
2. Find the indices of the DPDK-probed ports and start DOCA Flow ports with them.

7. Insert rules.

1. Insert encryption rules.

```
ipsec_security_gw_insert_encrypt_rules();
```

2. Insert decryption rules.

```
ipsec_security_gw_insert_decrypt_rules();
```

8. Wait for traffic.

```
ipsec_security_gw_wait_for_traffic();
```

1. wait in a loop until the user terminates the program.

9. IPsec security gateway cleanup:

1. DOCA Flow cleanup; destroy initialized ports.

```
doca_flow_cleanup();
```

2. SA destruction.

```
ipsec_security_gw_destroy_sas();
```

3. IPsec objects destruction.


```
ipsec_security_gw_ipsec_ctx_destroy();
```

4. Destroy DPDK ports and queues.

```
dpdk_queues_and_ports_fini();
```

5. DPDK finish.

```
dpdk_fini();
```

Calls `rte_eal_destroy()` to destroy initialized EAL resources.

6. Arg parser destroy.

```
doca_argp_destroy()
```

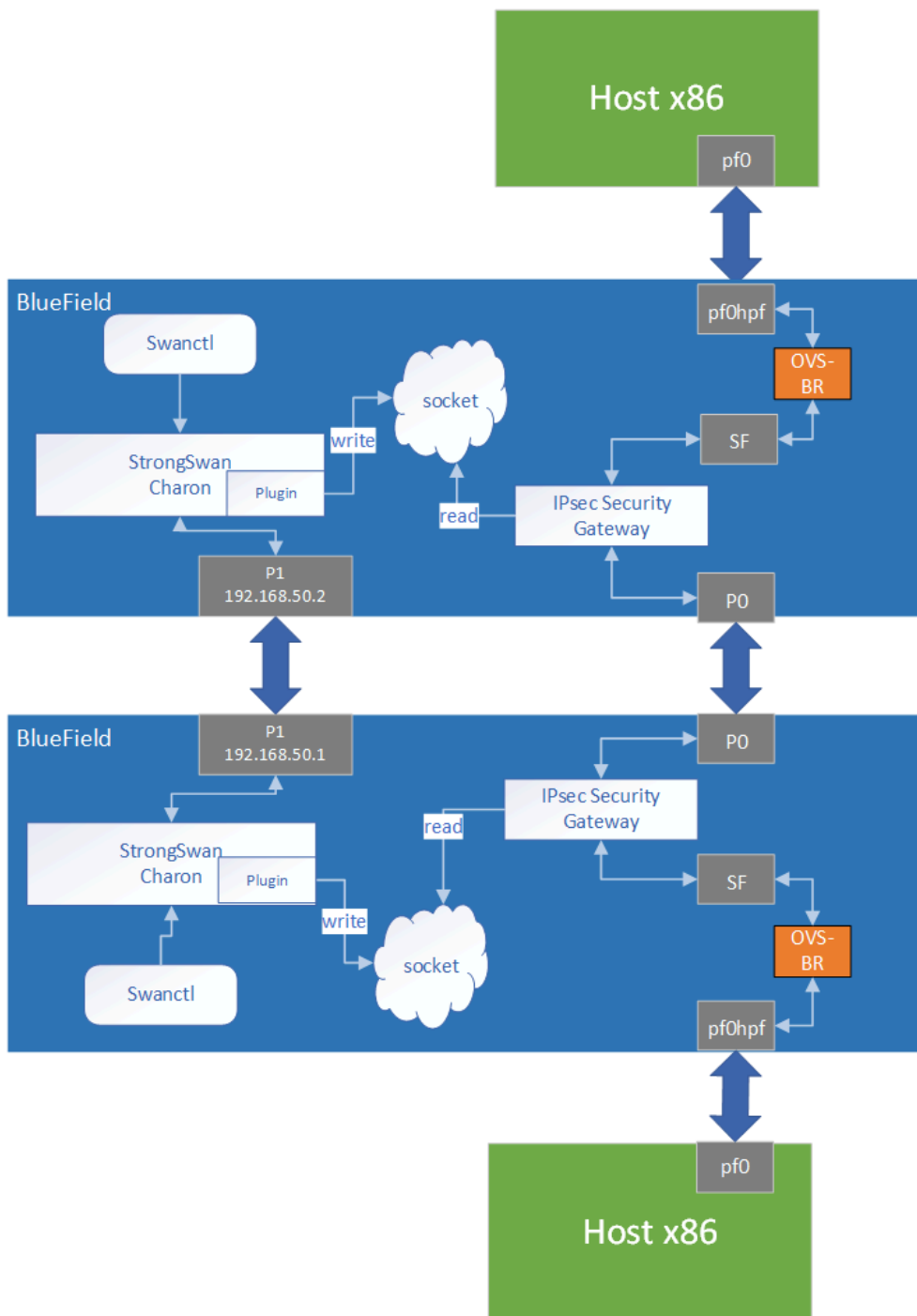
Keying Daemon Integration (StrongSwan)

strongSwan is a keying daemon that uses the Internet Key Exchange Version 2 (IKEv2) protocol to establish SAs between two peers. strongSwan includes a DOCA plugin that is part of the strongSwan package in BFB. The plugin is loaded only if the DOCA IPsec Security Gateway is triggered. The plugin connects to UDS socket and sends IPsec policies to the application after the key exchange completes.

For more information about the key daemon, refer to [strongSwan documentation](#) .

End-to-end Architecture

The following diagram presents an architecture where two BlueField DPUs are connected to each other with DOCA IPsec Security Gateway running on each.



swantcl is a command line tool used for strongSwan IPsec configuration:

1. Run DOCA IPsec Security Gateway on both sides in a dynamic configuration.
2. Start strongSwan service.
3. Configure strongSwan IPsec using the swantcl.conf configuration file on both sides.

4. Start key exchange between the two peers. At the end of the flow, the result arrives to the DOCA plugin, populates the policy-defined structure, and sends it to the socket.
5. DOCA IPsec Security Gateway on both sides reads new policies from the socket, performs the parsing, creates a DOCA SA object, and adds flow decrypt/encrypt entry.

This architecture uses P1 uplink on both BlueField DPUs to run the strongSwan key daemon. To configure the uplink:

1. Configure an IP addresses for the PFs of both DPUs:

1. On BF1:

```
ip addr add 192.168.50.1/24 dev p1
```

2. On BF2:

```
ip addr add 192.168.50.2/24 dev p1
```

Note

It is possible to configure multiple IP addresses to uplinks to run key exchanges with different policy attributes.

2. Verify the connection between two BlueField DPUs.

```
BF1> ping 192.168.50.2
```

Note

Make sure that the uplink is not in OVS bridges.

3. Configure the `swanctl.conf` files for each machine. The file should be located under `/etc/swanctl/conf.d/`.

Adding `swanctl.conf` file examples:

- Transport mode
 - `swanctl.conf` example for BF1:

```
connections {
  BF1-BF2 {
    local_addrs = 192.168.50.1
    remote_addrs = 192.168.50.2
    rekey_time = 0

    local {
      auth = psk
      id = host1
    }
    remote {
      auth = psk
      id = host2
    }

    children {
      bf {
        local_ts = 192.168.50.1/32 [udp/60]
        remote_ts = 192.168.50.2/32 [udp/90]
        esp_proposals = aes128gcm128-x25519-esn
        mode = transport
        policies_fwd_out = yes
        life_time = 0
      }
    }
  }
}
```

```
version = 2
mobike = no
reauth_time = 0
proposals = aes128-sha256-x25519
}
}

secrets {
ike-BF {
id-host1 = host1
id-host2 = host2
secret = 0sv+NkxY9LLZvwj4qCC2o/gGrWDF2d21jL
}
}
```

- swanctl.conf example for BF2:

```
connections {
BF2-BF1 {
local_addrs = 192.168.50.2
remote_addrs = 192.168.50.1
rekey_time = 0

local {
auth = psk
id = host2
}
remote {
auth = psk
id = host1
}

children {
bf {
local_ts = 192.168.50.2/32 [udp/90]
```

```

remote_ts = 192.168.50.1/32 [udp/60]
esp_proposals = aes128gcm128-x25519-esn
mode = transport
life_time = 0
}
}
version = 2
mobike = no
reauth_time = 0
proposals = aes128-sha256-x25519
}
}

secrets {
ike-BF {
id-host1 = host1
id-host2 = host2
secret = 0sv+NkxY9LLZvwj4qCC2o/gGrWDF2d21jL
}
}
}

```

- o Tunnel mode

```

connections {
BF1-BF2 {
local_addrs = 192.168.50.2
remote_addrs = 192.168.50.1
rekey_time = 0

local {
auth = psk
id = host2
}
remote {
auth = psk

```

```
id = host1
}

children {
  bf {
    local_ts = 2001:db8:85a3::8a2e:370:7334/128 [udp/3030]
    remote_ts = 2001:db8:85a3::8a2e:370:7335/128 [udp/55]
    esp_proposals = aes128gcm128-x25519-esn
    life_time = 0
  }
}
version = 2
mobike = no
proposals = aes128-sha256-x25519
}
}

secrets {
  ike-BF {
    id-host1 = host1
    id-host2 = host2
    secret = 0sv+NkxY9LLZvwj4qCC2o/gGrWDF2d21jL
  }
}
```

(i) Note

local_ts and remote_ts must have a netmask of /32 for IPv4 addresses and /128 for IPv6 addresses.

(i) Note

SA rekey is not supported in DOCA plugin.
connection.rekey_time must be set to 0 and
connection.child.life_time must be set to 0.

DOCA IPsec only supports ESP headers, AES-GCM encryption algorithm, and key sizes 128 or 256. Therefore, when setting ESP proposals in the `swanctl.conf`, please adhere to the values provided in the following table:

ESP Proposal	Algorithm Type Including ICV Length	Key Size
aes128gcm8	ENCR_AES_GCM_ICV8	128
aes128gcm64	ENCR_AES_GCM_ICV8	128
aes128gcm12	ENCR_AES_GCM_ICV12	128
aes128gcm96	ENCR_AES_GCM_ICV12	128
aes128gcm16	ENCR_AES_GCM_ICV16	128
aes128gcm128	ENCR_AES_GCM_ICV16	128
aes128gcm	ENCR_AES_GCM_ICV16	128
aes256gcm8	ENCR_AES_GCM_ICV8	256
aes256gcm64	ENCR_AES_GCM_ICV8	256
aes256gcm12	ENCR_AES_GCM_ICV12	256
aes256gcm96	ENCR_AES_GCM_ICV12	256
aes256gcm16	ENCR_AES_GCM_ICV16	256
aes256gcm128	ENCR_AES_GCM_ICV16	256
aes256gcm	ENCR_AES_GCM_ICV16	256

Running the Solution

Run the following commands on both BlueField peers.

1. Run DOCA IPsec Security Gateway in dynamic configuration, assuming the socket location is `/tmp/rules_socket`.

```
doca_ipsec_security_gw -s 03:00.0 -un <sf_net_dev> -c  
./ipsec_security_gw_config.json -m transport -i /tmp/rules_socket
```

Note

DOCA IPsec Security Gateway application should be run first.

2. Edit the `/etc/strongswan.d/charon/doca.conf` file and add the UDS socket path. If the `socket_path` is not set, the plugin uses the default path `/tmp/strongswan_doca_socket`.

```
doca {  
  
# Whether to load the plugin  
load = yes  
  
# Path to DOCA socket  
socket_path = /tmp/rules_socket  
}
```

Note

You must provide the application with this path as well.

3. Restart the strongSwan server:

```
systemctl restart strongswan.service
```

(i) Note

If the application has been run with log level debug, you can see that the connection has been done successfully and the application is waiting for new IPsec policies.

4. Verify that the `swanctl.conf` file exists in `/etc/swanctl/conf.d/` directory.

(i) Note

It is recommended to remove any unused conf files under `/etc/swanctl/conf.d/`.

5. Load IPsec configuration:

```
swanctl --load-all
```

6. Start IKE protocol on either the initiator or the target side:

```
swanctl -i --child <child_name>
```

(i) Info

In the example above, the child's name is `bf`.

Building strongSwan

To perform some changes in the DOCA plugin in [strongSwan](#) zone:

1. Verify that the dependencies listed [here](#) are installed in your environment. `libgmp-dev` is missing from that list so make sure to install that as well.
2. Git clone <https://github.com/Mellanox/strongswan.git>.
3. Git checkout BF-5.9.10 branch.
4. Add your changes in the plugin located under `src/libcharon/plugins/doca`.
5. Run `autogen.sh` within the strongSwan repo.
6. Run the following:

```
./configure --enable-openssl --disable-random --prefix=/usr/local --  
sysconfdir=/etc --enable-systemd --enable-doca  
make  
make install  
systemctl daemon-reload  
systemctl restart strongswan.service
```

References

- `/opt/mellanox/doca/applications/ipsec_security_gw/`
- `/opt/mellanox/doca/applications/ipsec_security_gw/ipsec_security_gw_params.json`

NVIDIA DOCA NAT Application Guide

This document provides a NAT implementation on top of NVIDIA® BlueField® DPU.

Introduction

The Network Address Translation (NAT) reference application leverages the DPU's hardware capability to switch packets with local IP addresses to global ones and vice

versa.

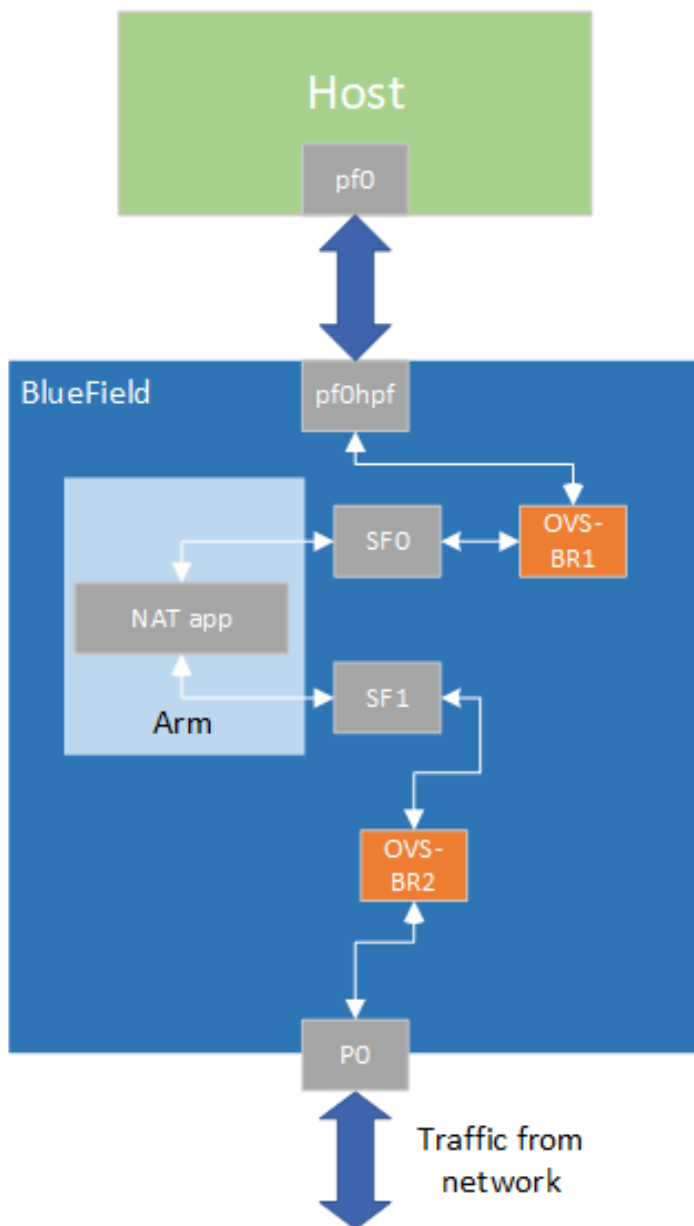
The NAT application is based on the [DOCA Flow](#) API used for the programming of the DPU's hardware.

NAT can operate in three modes:

- Static mode – application gets pairs of local IP address and global IP address from the user using a JSON file
- Dynamic mode – user provides pool of global IP addresses that can be used. The application should pick one address from the pool for new local area network (LAN) IP address and use it. Once the session closes, the addresses are returned to the pool.
- PAT mode (DNS offload) – the user provides one global address to use. In addition, the user provides mapping between the local port address to the global port. For each packet, the local address is replaced with the global one and ports are replaced according to mapping table.

System Design

The NAT application is design to run on the DPU. The DPU intercepts ingress traffic from both wire and host, switches the relevant IP address and port according to data configured by the user, and forwards it to the egress port.



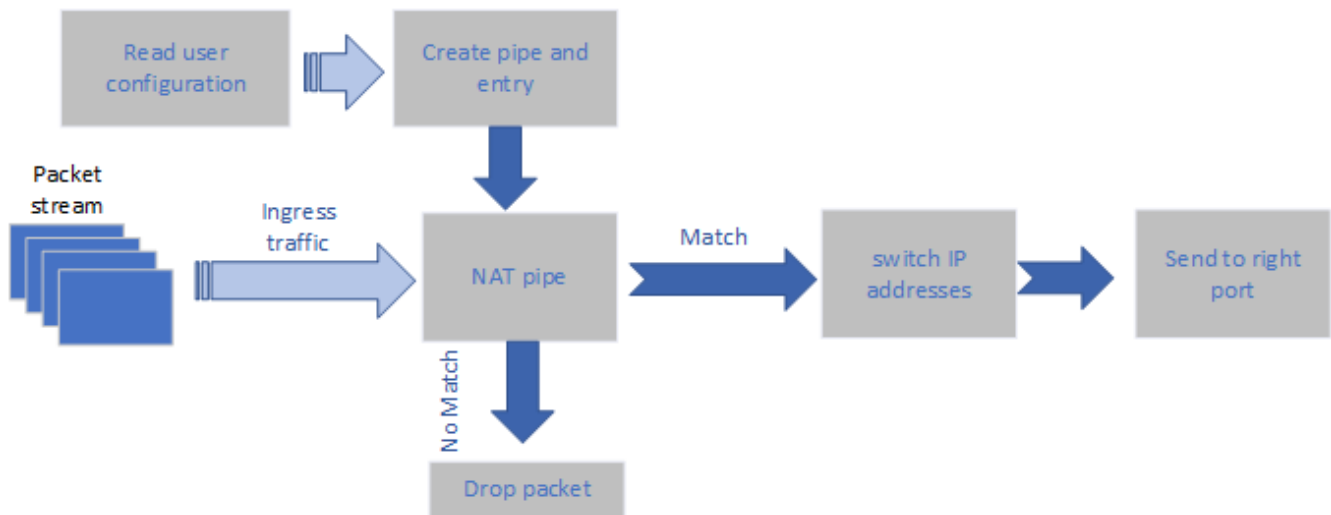
Application Architecture

NAT runs on the DPU to classify packets.

The app should be configured using a JSON file which includes the operation mode.

Static Mode

For static mode, the JSON file should include pairs of local and global IP addresses. No change for ports in this mode.

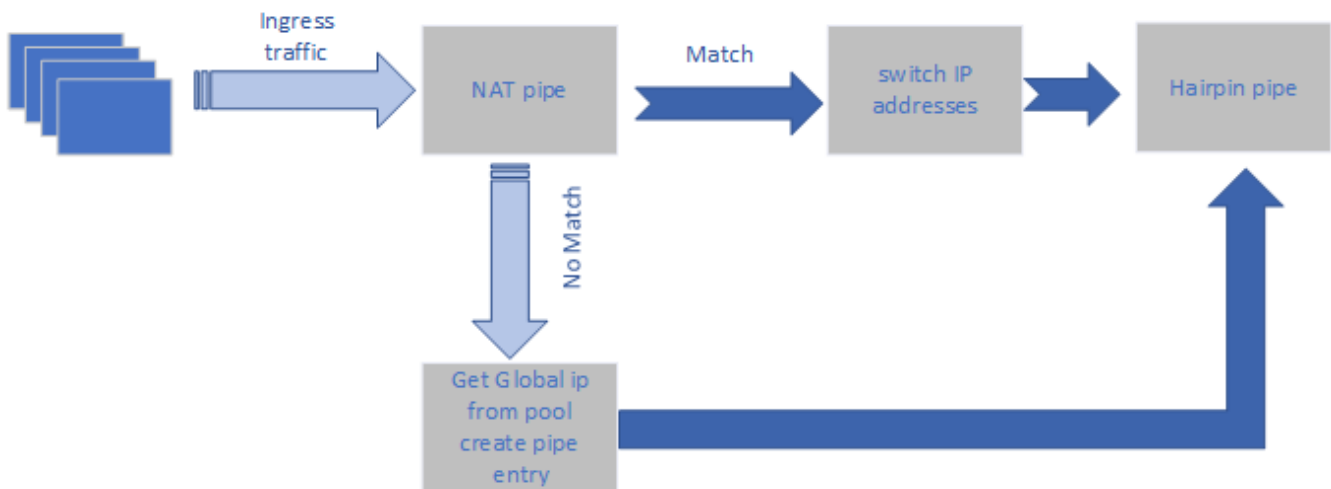


Dynamic Mode

The user must provide a pool of global IP addresses to use. The application allocates a global address to every miss in the pipe (new local address).

If no more global addresses are available in the pool, the user gets an error message and the packet is sent as is.

The application performs a callback to remove the matching of global and local IPs and returns the address to the pool.

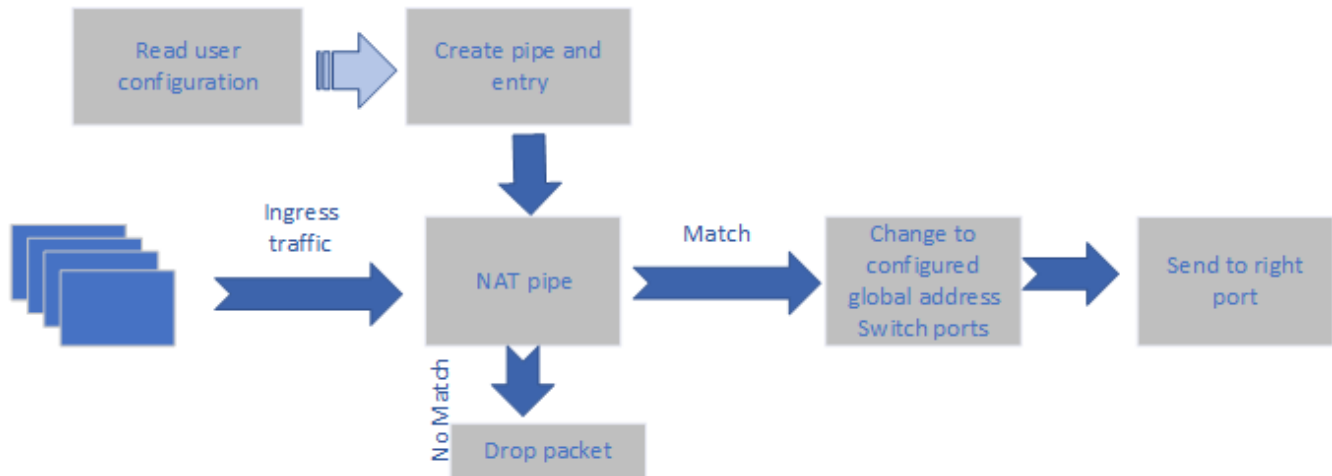


PAT (NAT Offload) Mode

The user provides a global address to replace all local addresses in the user LAN.

The user provides a matching of local IP and port to global port.

The application changes the local IP of every match to the global IP provided by the user and updates the port number according to user configuration.



DOCA Libraries

This application leverages the following DOCA library:

- [DOCA Flow](#)

Refer to its respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/nat/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

`doca_nat` is created under `/tmp/build/nat/`.

Compiling NAT Application Only

To directly build only the NAT application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_nat=true
```



```
ninja -C /tmp/build
```

Info

doca_nat is created under /tmp/build/nat/.

Alternatively, users can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
 - o Set enable_all_applications to false
 - o Set enable_nat to true
2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_nat is created under /tmp/build/nat/

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Prerequisites

The NAT application is based on DOCA Flow. Therefore, the user is required to allocate huge pages.

```
echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

Note

On some operating systems (RockyLinux, OpenEuler, CentOS 8.2) the default huge page size on the DPU (and Arm hosts) is larger than 2MB, and is often 512MB instead. Once can find out the size of the huge pages using the following command:

```
$ grep -i huge /proc/meminfo
```

```
AnonHugePages: 0 kB  
ShmemHugePages: 0 kB  
FileHugePages: 0 kB  
HugePages_Total: 4  
HugePages_Free: 4  
HugePages_Rsvd: 0  
HugePages_Surp: 0  
Hugepagesize: 524288 kB  
Hugetlb: 6291456 kB
```

Given that the guiding principal is to allocate 4GB of RAM, in such cases instead of allocating 2048 pages, one should allocate the matching amount (8 pages):

```
echo '8' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-524288kB/nr_hugepages
```

Application Execution

The NAT application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_nat [DPDK Flags] -- [DOCA Flags] [Program Flags]
```

DOCA Flags:

- h, --help Print a help synopsis
- v, --version Print program version information
- l, --log-level Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
- sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
- j, --json <path> Parse all command flags from an input json file

Program Flags:

- m, --mode <mode> set NAT mode
- r, --nat-rules <path> Path to the JSON file with NAT rules
- lan, --lan-intf <lan intf> name of LAN interface
- wan, --wan-intf <wan intf> name of wan interface

Info

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_nat -- -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the BlueField:

```
./doca_nat -a auxiliary:mlx5_core.sf.4,dv_flow_en=2 -a  
auxiliary:mlx5_core.sf.5,dv_flow_en=2 -- -m static -r nat_static_rules.json -lan sf3  
-wan sf4
```

Note

SFs must be enabled according to [NVIDIA BlueField DPU Scalable Function User Guide](#).

Note

The flag `-a auxiliary:mlx5_core.sf.4,dv_flow_en=2 -a auxiliary:mlx5_core.sf.5,dv_flow_en=2` is mandatory for proper usage of the application. Modifying this flag results in unexpected behavior as only 2 ports are supported. The SF number is arbitrary and configurable.

Note

The SF numbers must match the identifiers of the configured SFs.

3. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_nat --json [json_file]
```

For example:

```
./doca_nat --json ./nat_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
DPDK Flags	a	devices	Add a PCIe device into the list of devices to probe	<pre>"devices": [</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
				<pre> {"device": "sf", "id": "4","hws": true}, {"device": "sf", "id": "5","hws": true},]</pre>
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with TRACE log level support) 	<pre>"log-level": 60</pre>
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	<pre>"sdk-log-level": 40</pre>
	j	json	Parse all command flags from an input json file	N/A

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
Program Flags	m	mode	Set NAT mode	<pre>"mode": "static"</pre>
	r	nat-rules	Path to the JSON file with NAT rules	<pre>"nat-rules": "nat_static_rules.json"</pre>
	lan	lan-intf	Name of LAN interface	<pre>"lan-intf": "sf3"</pre>
	wan	Wan-intf	Name of WAN interface	<pre>"wan-intf": "sf4"</pre>

Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications .

Application Code Flow

1. Parse application argument.
 1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register NAT application.

```
register_nat_params()
```

3. Parse the arguments.

```
doca_argp_start();
```

1. Parse DPDK flags and invoke handler for calling the `rte_eal_init()` function.

2. Parse app parameters.

2. DPDK initialization.

```
dpdk_init();
```

Calls `rte_eal_init()` to initialize EAL resources with the provided EAL flags.

3. DPDK port initialization and start.

```
dpdk_queues_and_ports_init();
```

1. Initialize DPDK ports, including mempool allocation.

2. Initialize hairpin queues if needed.

3. Bind hairpin queues of each port to its peer port.

4. NAT initialization.

```
nat_init();
```

1. DOCA Flow and DOCA Flow port initialization.

5. Init user configuration rules into app structure.

```
parsing_nat_rules();
```

6. Init pipes and entry according to rules.

```
nat_pipes_init();
```

7. Wait for signal to end application.

8. NAT destroy.

```
nat_destroy();
```

9. DPDK ports and queues destruction.

```
dpdk_queues_and_ports_fini();
```

10. DPDK finish.

```
dpdk_fini();
```

1. Calls `rte_eal_destroy()` to destroy initialized EAL resources.

11. Arg parser destroy.

```
doca_argp_destroy();
```

References

- `/opt/mellanox/doca/applications/nat/`
- `/opt/mellanox/doca/applications/nat/nat_params.json`

NVIDIA DOCA PCC Application Guide

This document provides a DOCA PCC implementation on top of NVIDIA® BlueField® networking platform .

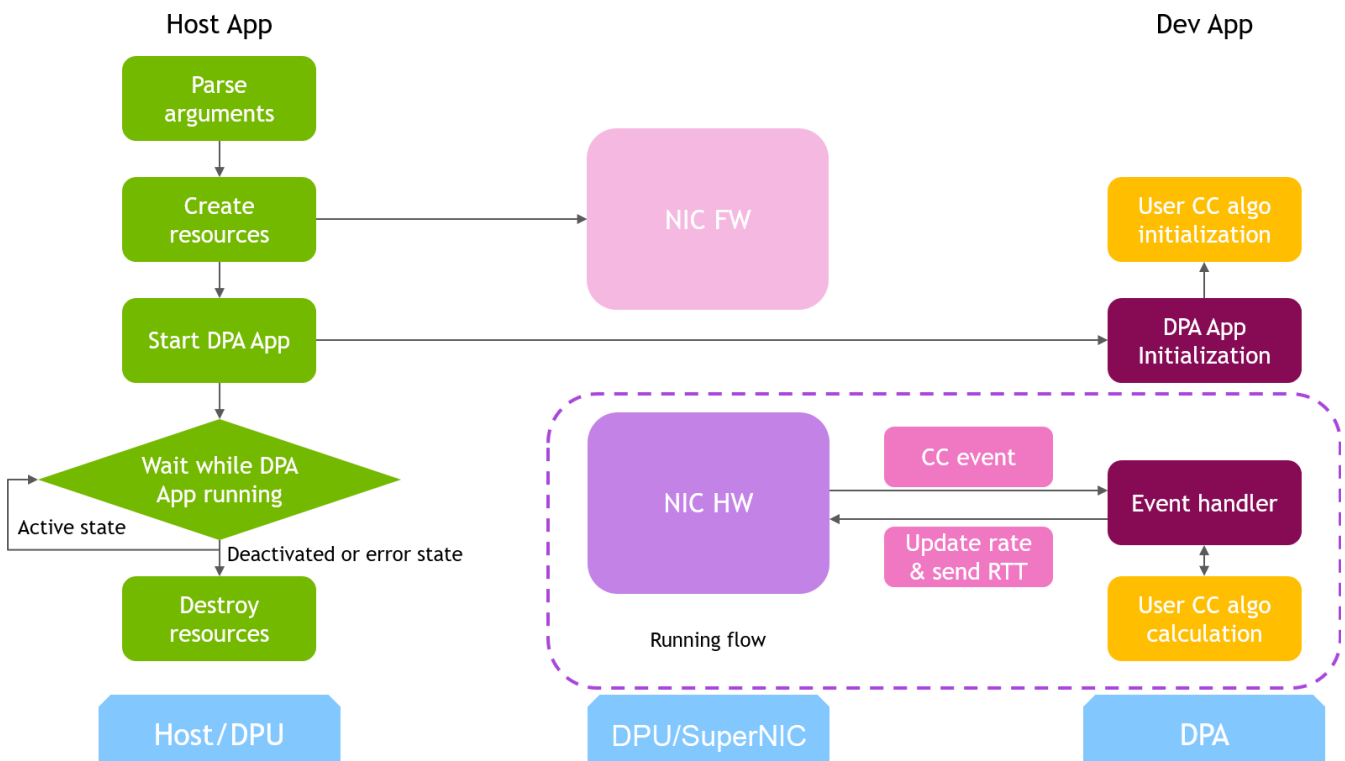
Introduction

Programmable Congestion Control (PCC) allows users to design and implement their own congestion control (CC) algorithm, giving them the flexibility to work out an optimal solution to handle congestion in their clusters. On BlueField-3 networking platform , PCC is provided as a component of DOCA.

The application leverages the [DOCA PCC API](#) to provide users the flexibility to manage allocation of DPA resources according to their requirements.

Typical DOCA application includes App running on host/Arm and App running on DPA. Developers are advised to use the host/Arm application with minimal changes and focus on developing their algorithm and integrating it into the DPA application.

System Design



DOCA PCC application consists of two parts:

- Host/Arm app is the control plane. It is responsible for allocating all resources and handover to the DPA app initially, then destroying everything when the DPA app finishes its operation. The host app must always be alive to stay in control while the device app is working.
- Device/DPA app is the data plane. It is mainly for reaction point CC event handler. When the first thread is activated, DPA App initialization is done in the DOCA PCC library by calling the algorithm initialization function implemented by the user in the app. Moreover, the user algorithm execution function is called when a CC event arrives. The user algorithm takes event data as input and performs a calculation using per-flow context and replies with updated rate value and a flag to sent RTT request.

Info

The device/DPA app is as well capable of functioning as a telemetry program for notification point NIC operations, which users can configure as a runtime option.

The host/Arm application sends a command to the BlueField platform firmware when allocating or destroying resources. CC events are generated by the BlueField platform hardware automatically when sending data or receiving ACK/NACK/CNP/RTT packets, then the device application handles these events by calling the user algorithm. After the DPA application replies to hardware, handling of current event is done and the next event can arrive.

Application Architecture

```
/opt/mellanox/doca/applications/pcc/  
  host  
    pcc.c  
    pcc_core.c  
    pcc_core.h
```

```
device
rp
  algo
  rtt_template.h
  rtt_template_algo_params.h
  rtt_template_ctxt.h
  rtt_template.c
  pcc_rp_dev.c
np_nic_telemetry
  pcc_np_nic_telemetry_dev.c
```

The main content of the reference DOCA PCC application files are the following:

- `host/pcc.c` – entry point to entire application
- `host/pcc_core.c` – host functions to initialize and destroy the PCC application resources, parsers for PCC command line parameters
- `device/rp/pcc_rp_dev.c` – callbacks for user CC algorithm initialization, user CC algorithm calculation, algorithm parameter change notification
- `device/rp/algo/*` – user CC algorithm reference template. Put user algorithm code here.
- `device/np_nic_telemetry/pcc_np_nic_telemetry_dev.c` – callback for user notification point handling, implemented as a NIC telemetry program to observe RX counters

DOCA Libraries

This application leverages the following DOCA library:

- [DOCA PCC](#)

Refer to its respective programming guide for more information.

Dependencies

- NVIDIA BlueField-3 Platform is required
- Firmware 32.38.1000 and higher

- MFT 4.25 and higher

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/pcc/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build
```

```
ninja -C /tmp/build
```

Info

doca_pcc is created under /tmp/build/pcc/.

Compiling Only the Current Application

To directly build only the PCC application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_pcc=true  
ninja -C /tmp/build
```

Info

doca_pcc is created under /tmp/build/pcc/.

Alternatively, one can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
 - Set enable_all_applications to false
 - Set enable_pcc to true
2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_pcc is created under /tmp/build/pcc/.

Compilation Options

The application offers specific compilation flags which one can set for a desired behavior in the device/DPA program.

In the meson_options.txt file, one can find the following options:

- enable_pcc_application_tx_counter_sampling: set to true to use TX counters sampled at runtime in the reaction point CC handling algorithm.
- enable_pcc_application_np_rx_rate: set to true to use RX counters received from notification point in the reaction point CC handling algorithm.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application.

Running the Application

Prerequisites

Enable USER_PROGRAMMABLE_CC in mlxconfig:

```
mlxconfig -y -d /dev/mst/mt41692_pciconf0 set USER_PROGRAMMABLE_CC=1
```

Perform a [BlueField system reboot](#) for the mlxconfig settings to take effect.

Application Execution

The PCC application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

Usage: `doca_pcc [DOCA Flags] [Program Flags]`

DOCA Flags:

`-h, --help` Print a help synopsis

`-v, --version` Print program version information

`-l, --log-level` Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

`--sdk-log-level` Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

`-j, --json <path>` Parse all command flags from an input json file

Program Flags:

`-d, --device <IB device names>` IB device name that supports PCC (mandatory).

`-np-nt, --np-nic-telemetry <PCC Notification Point NIC Telemetry>` Flag to indicate running as a Notification Point NIC Telemetry (optional). By default the flag is set to `false`.

`-t, --threads <pcc-threads-list>` A list of the PCC threads numbers to be chosen for the DOCA PCC context to run on (optional). Must be provided as a string, such that the number are separated by a space.

-w, --wait-time <PCC wait time> The duration of the DOCA PCC wait (optional), can provide negative values which means infinity. If not provided then -1 will be chosen.

-p, --probe-packet-format <PCC probe-packet format> The probe packet format of the DOCA PCC (optional). Available values for each type: CCMAD-0, IFA1-1, IFA2-2. By default format is set to CCMAD.

-r-handler, --remote-sw-handler <CCMAD remote SW handler> CCMAD remote SW handler flag (optional). If not provided then false will be chosen.

-gns, --global-namespace <IFA2 global namespace> The IFA2 probe packet global namespace (optional). If not provided then 0xF will be chosen.

-gns-ignore_mask, --global-namespace-ignore-mask <IFA2 global namespace ignore mask> The IFA2 probe packet global namespace ignore mask (optional). If not provided then 0 will be chosen.

-gns-ignore_val, --global-namespace-ignore-value <IFA2 global namespace ignore value> The IFA2 probe packet global namespace ignore value (optional). If not provided then 0 will be chosen.

-f, --coredump-file <PCC coredump file> A pathname to the file to write coredump data in case of unrecoverable error on the device (optional). Must be provided as a string.

-i, --port-id <Physical port ID> The physical port ID of the device running the application (optional). If not provided then ID 0 will be chosen.

Info

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_pcc -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the BlueField Platform or the host:

```
./doca_pcc -d mlx5_0
```

Note

The IB device identifier (`mlx5_0`) should match the identifier of the desired IB device.

3. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_pcc --json [json_file]
```

For example:

```
./doca_pcc --json ./pcc_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	N/A

i Info
 The application uses a unique logging implementation that makes use

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
			of DOC A's loggi ng levels .	
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	N/A
	j	json	Parse all command flags from an input JSON file	N/A
Program flags	d	device	IB device name that supports PCC	<pre>"device": ""</pre>
	np-nt	np-nic-telemetry	(Optional) Flag to indicate running as a Notification Point NIC Telemetry. The DOCA PCC application can also run as a Notification Point NIC telemetry program, instead of a Reaction point that runs the CC algorithm. If the user uses this flag, the application will load a program to run on the DPA to sample	<pre>"np-nic-telemetry": false</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
			<p>RX NIC counters and send them in response packet.</p>	
	t	threads	<p>(Optional) A list of the PCC EU indexes to be chosen for the DOCA PCC event handler threads to run on. Must be provided as a string, such that the numbers are separated by a space.</p> <p>The placement of the PCC threads per core can be controlled using the EU indexes. Utilizing a large number of EUs, while limiting the number of threads per core, gives the best event handling rate and lowest event latency.</p> <p>The last EU is used for communication with the BlueField Platform while all others are for data path CC event handling.</p>	<pre>"pcc-threads": "176 177 178 179 180 181 182 183 184 185 186 187 192 193 194 195 196 197 198 199 200 201 202 203 208 209 210 211 212 213 214 215 216 217 218 219 224 225 226 227 228 229 230 231 232 233 234 235 240"</pre>
<p>Note If np-nic-telemetry option is chosen by the user, a different set of threads will be</p>				

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
			chosen as default list.	
	w	wait-time	(Optional) In seconds, the duration of the DOCA PCC wait. Negative values mean infinity.	<code>"wait-time": -1</code>
	p	probe-packet-format	(Optional) The probe packet format of the DOCA PCC (optional). Available values for each type: CCMAD-0, IFA1-1, IFA2-2. By default, format is set to CCMAD.	<code>"probe-packet-format": 0</code>
	r-handler	remote-sw-handler	(Optional) CCMAD remote SW handler flag. Relevant for reaction point contexts. This flag indicates whether the expected CCMAD probe packet responses are generated by a remote DOCA notification point process or not. <div style="background-color: #ffffcc; padding: 10px;"> <p>(i) Note If using other probe types than CCMAD, probe packet</p> </div>	<code>"remote-sw-handler": false</code>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
			<p>responses are always expected to be generated from a remote DOCA notification point process.</p>	
	gns	global-namespace	<p>(Optional) The IFA2 probe packet global namespace</p> <p>i Info Relevant for reaction point contexts.</p>	<pre>"global-namespace": 0xF</pre>
	gns-ignore-mask	global-namespace-ignore-mask	<p>(Optional) The IFA2 probe packet global namespace ignore mask</p> <p>i Info Relevant for</p>	<pre>"global-namespace-ignore-mask": 0</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
			notification point contexts.	
	gns-ignore-val	global-namespace-ignore-value	<p>(Optional) The IFA2 probe packet global namespace ignore value</p> <p>i Info Relevant for notification point contexts.</p>	<pre>"global-namespace-ignore-value": 0</pre>
	f	coredump-file	(Optional) A pathname to the file to write core dump data if an unrecoverable error occurs on the device	<pre>"coredump-file": "/tmp/doca_pcc_coredump.txt "</pre>
	i	port-id	(Optional) The physical port ID of the device running the application	<pre>"port-id": 0</pre>

i Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.

Application Code Flow

This section lists the application's configuration flow, explaining the different DOCA function calls and wrappers.

1. Parse application argument.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register PCC application parameters.

```
register_pcc_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

1. Parse DOCA flags.

2. Parse DOCA PCC parameters.

2. PCC initialization.

```
pcc_init();
```

1. Open DOCA device that supports PCC.

2. Create DOCA PCC context.
3. Configure affinity of threads handling CC events.

3. Start DOCA PCC.

```
doca_pcc_start();
```

1. Create PCC process and other resources.
2. Trigger initialization of PCC on device.
3. Register the PCC in the BlueField Platform hardware so CC events can be generated and an event handler can be triggered.

4. Process state monitor loop.

```
doca_pcc_get_process_state();  
doca_pcc_wait();
```

1. Get the state of the process:

State	Description
DOCA_PCC_PS_ACTIVE = 0	The process handles CC events (only one process is active at a given time)
DOCA_PCC_PS_STANDBY = 1	The process is in standby mode (another process is already ACTIVE)
DOCA_PCC_PS_DEACTIVATED = 2	The process has been deactivated by the BlueField Platform firmware and should be destroyed
DOCA_PCC_PS_ERROR = 3	The process is in error state and should be destroyed

2. Wait on process events from the device.

5. PCC destroy.

```
doca_pcc_destroy();
```

1. Destroy PCC resources. The process stops handling PCC events.
 2. Close DOCA device.
6. Arg parser destroy.

```
doca_argp_destroy()
```

Port Programmable Congestion Control Register

The Port Programmable Congestion Control (PPCC) register allows the user to configure and read PCC algorithms and their parameters/counters.

It supports the following functionalities:

- Enabling different algorithms on different ports
- Querying information of both algorithms and tunable parameters/counters
- Changing algorithm parameters without compiling and reburning user image
- Querying or clearing programmable counters

Usage

The PPCC register can be accessed using a string similar to the following:

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=0" --reg_name  
PPCC --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"  
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --set "cmd_type=1" --reg_name PPCC  
--indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
```

Where you must:

- Set the `cmd_type` and the indexes

- Give values for algo_slot, algo_param_index
- Keep local_port=1, pnat=0, lp_msb=0
- Keep doca_pcc application running

cmd_type	Description	Method	Index	Input (in --set)	Output
0x0	Get algorithm info	Get	algo_slot	N/A	<ul style="list-style-type: none"> • Value – 32-bit algo_num or 0 if no algo is available at this index • Text – algorithm description • sl_bitmask_support – indicates whether the device supports sl_bitmask logic
0x1	Enable algorithm	Set		sl_bitmask trace_en counter_en	N/A
0x2	Disable algorithm	Set		N/A	N/A
0x3	Get algorithm enabling status	Get		N/A	<ul style="list-style-type: none"> • Value: <ul style="list-style-type: none"> ◦ 0 – disabled ◦ 1 – enabled • sl_bitmask – this field allows to apply to specific SLs based on the bitmask • sl_bitmask_support – indicates whether the device supports sl_bitmask logic
0x4	Get number of parameters	Get		N/A	<ul style="list-style-type: none"> • Value – num of params of algo
0x5	Get parameter information	Get	algo_slot algo_param_index	N/A	<ul style="list-style-type: none"> • param_value1 – default value of param • param_value2 – min value of param • param_value3 – max value of param • prm –

cmd_type	Description	Method	Index	Input (in --set)	Output
					<ul style="list-style-type: none"> ○ 0: read-only ○ 1: read-write ○ 2: read-only but may be cleared using the "get and clear" command
0x6	Get parameter value	Get		N/A	<ul style="list-style-type: none"> ● Value – param value
0x7	Get and clear parameter	Get		N/A	<ul style="list-style-type: none"> ● Value – param value
0x8	Set parameter value	Set		Parameter value	N/A
0xA	Bulk get parameters	Get		N/A	<ul style="list-style-type: none"> ● text_length – param num x 4 bytes ● text[0]...text[n] – param values
0xB	Bulk set parameters	Set	algo_slot	text_length - param num x 4 text[0]... text[n] - param values	N/A
0xC	Bulk get counters	Get		N/A	<ul style="list-style-type: none"> ● text_length – counter num x 4 bytes ● text[0]...text[n] – counter values
0xD	Bulk get and clear counters	Get		N/A	<ul style="list-style-type: none"> ● text_length – counter num x 4 bytes ● text[0]...text[n] – counter values
0xE	Get number of counters	Get		N/A	<ul style="list-style-type: none"> ● Value – num of counters of algo

cmd_type	Description	Method	Index	Input (in --set)	Output
0xF	Get counter information	Get	algo_slot algo_param_index	N/A	<ul style="list-style-type: none"> param_value3 – max value of parameter prm – <ul style="list-style-type: none"> 0: read-only 1: read-write 2: read-only but may be cleared via "get & clear" command
0x10	Get algorithm info array	Get	N/A	N/A	<ul style="list-style-type: none"> text_length – algo slot initialized x 4 bytes text[0]...text[n] – 32-bit algo_num or 0 if no algorithm is available at this slot index

Internal Default Algorithm

The internal default algorithm is used when enhanced connection establishment (ECE) negotiation fails. It is mainly used for backward compatibility and can be disabled using "force mode". Otherwise, users may change `doca_pcc_dev_user_algo()` in the device app to run a specific algorithm without considering the algorithm negotiation.

The force mode command is per port:

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=2" --reg_name PPCC --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=15,algo_param_index=0"
sudo mlxreg -d /dev/mst/mt41692_pciconf0.1 -y --get --op "cmd_type=2" --reg_name PPCC --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=15,algo_param_index=0"
```

Counters

Counters are shared on the port and are only enabled on one `algo_slot` per port. The following command enables the counters while enabling the algorithm according to the `algo_slot`:

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --set "cmd_type=1,counter_en=1" --  
reg_name PPCC --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
```

After counters are enabled on the `algo_slot`, they can be queried using `cmd_type 0xC` or `0xD`.

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=12" --reg_name  
PPCC --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"  
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=13" --reg_name  
PPCC --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
```

References

- `/opt/mellanox/doca/applications/pcc/`
- `/opt/mellanox/doca/applications/pcc/pcc_params.json`

NVIDIA DOCA PSP Gateway Application Guide

This document describes the usage of the NVIDIA DOCA PSP Gateway sample application on top of an NVIDIA® BlueField® networking platform or NVIDIA® ConnectX® SmartNIC.

Introduction

Note

DOCA PSP Gateway is supported at alpha level.

The DOCA PSP Gateway application leverages the BlueField or ConnectX hardware capability for fully offloaded secure network communication using the [PSP](#) security protocol. The application demonstrates how to exchange keys between application instances and insert rules controlling PSP encryption and decryption using the DOCA Flow library.

Warning

The application exchanges keys using an unencrypted [gRPC](#) channel. If your environment requires the protection of encryption keys, you must modify the application to create the gRPC channel using the applicable certificates.

Info

The PSP Gateway application supports only the PSP tunnel protocol. The PSP transport protocol is not supported by the application in this release, although it is supported by the underlying DOCA Flow library.

Info

The PSP Gateway application supports only IPv4 inner and IPv6 outer headers. Other combinations are not supported by the application in the current release, although they are supported by the underlying DOCA Flow library.

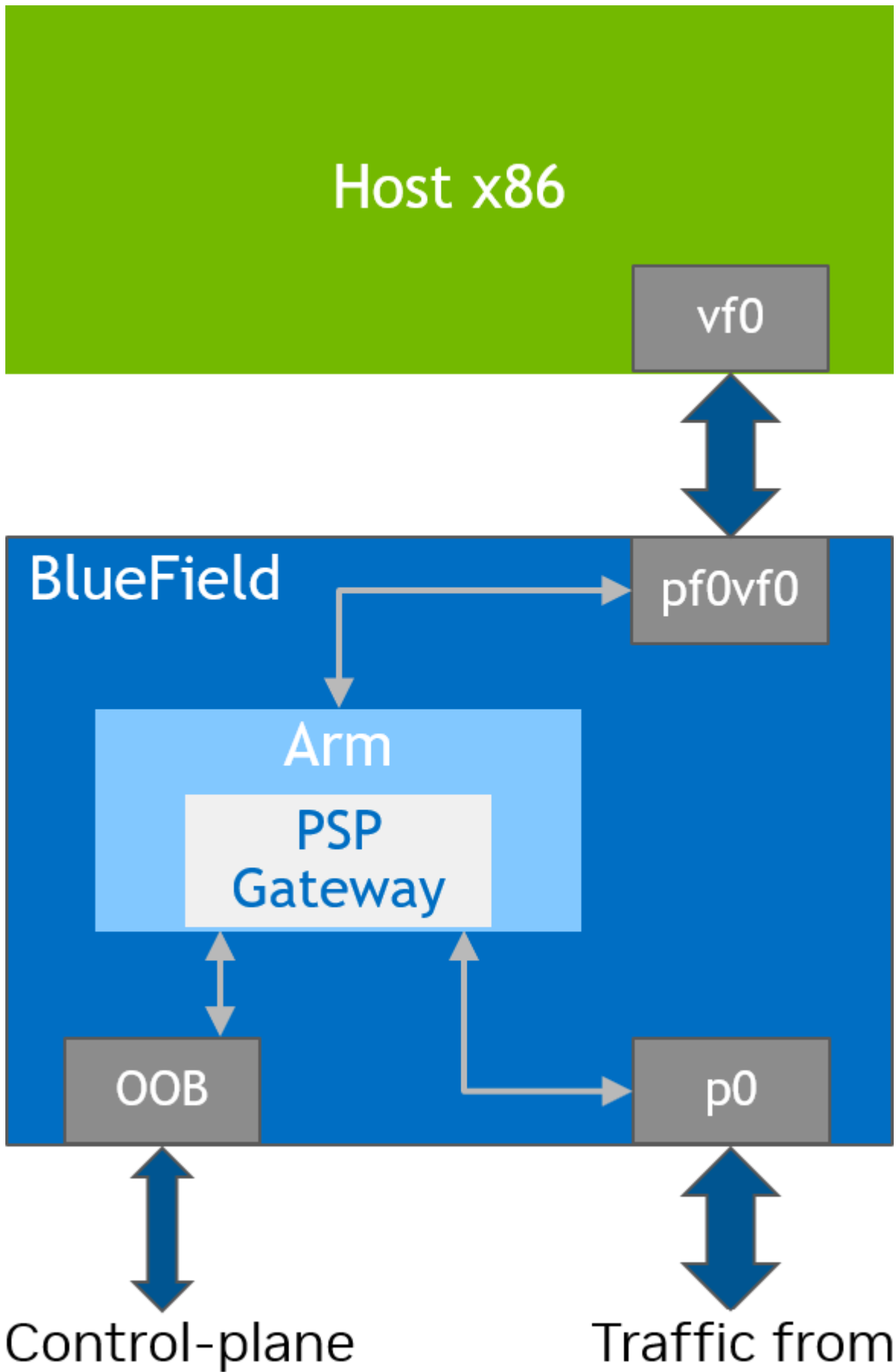
The application can be configured to establish out-bound PSP tunnel connections via individual command-line arguments, or via a text file configured via a command-line argument. The connections are established on-demand by default, but can also be configured to connect at startup.

System Design

The DOCA PSP Gateway is designed to run with three ports:

- A secure (encrypted) uplink netdev (i.e., p0)
- An unsecured (plaintext) netdev representor (VF or SF)
- An out-of-bound (OOB) management port, used to communicate with peer instances using standard sockets

Whether the DOCA PSP Gateway is deployed to a BlueField DPU or a ConnectX NIC, the functionality is the same. The Out of Bounds (OOB) network device carries PSP parameters between peers, the Uplink port carries secure (encrypted) traffic, and the VF carries the unencrypted traffic.

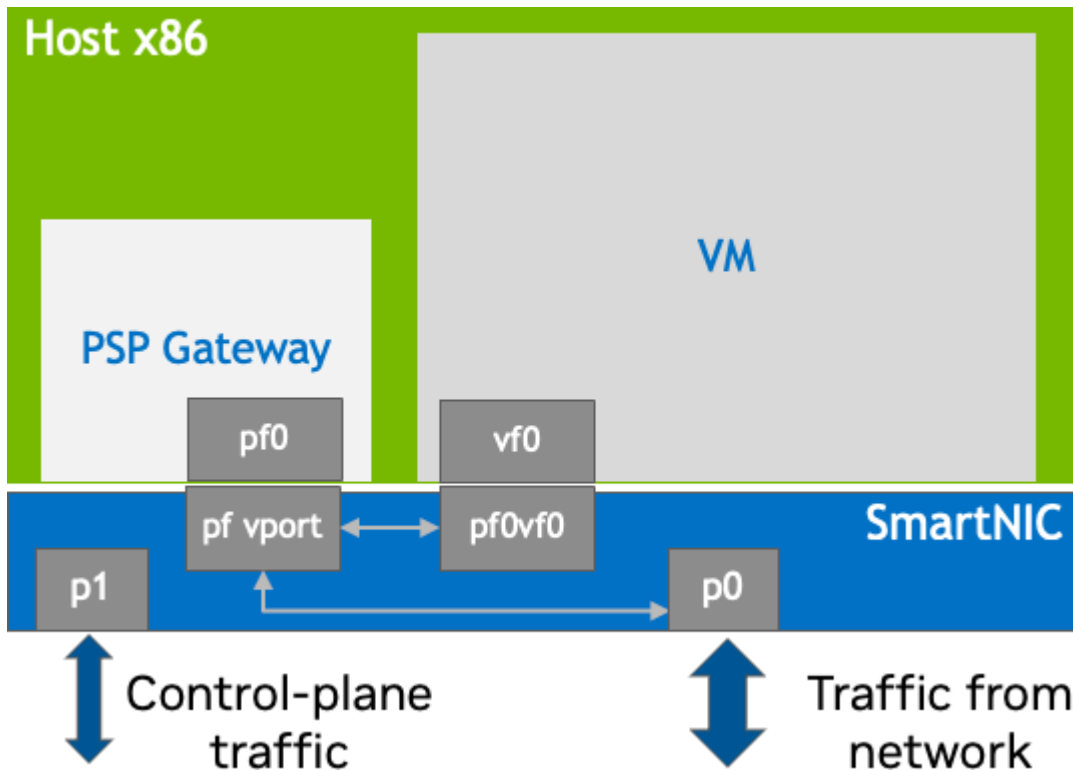


traffic

network

DOCA PSP Gateway - Deployment to DPU

When the application is deployed to a DPU, the operation of the PSP encryption protocol is entirely transparent to the Host. All the resources required to manage the PSP connections are physically located on the DPU.



When the application is deployed to the host, the operation of the PSP encryption protocol is the responsibility of the host, and resources are allocated from the host. However, the operation of the PSP encryption protocol is entirely transparent to any virtual machines and containers attached to the VF network devices.

Application Architecture

The creation of PSP tunnel connections requires two-way communication between peers. Each "sender" must request a unique security parameters index (SPI) and encryption key from the intended "receiver". The receiver derives sequential SPIs and encryption keys

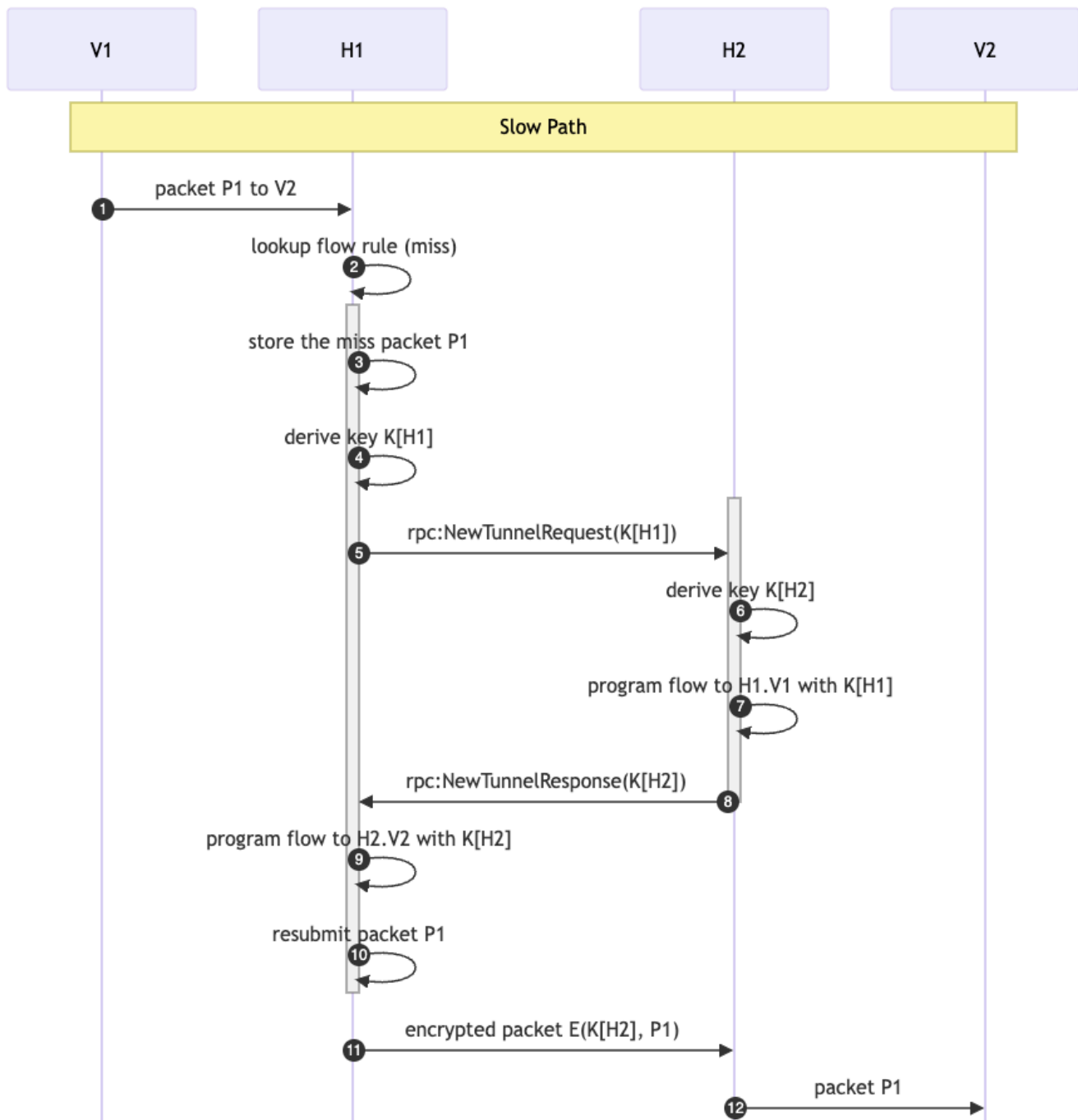
using the hardware resources inside the BlueField or ConnectX device, which manages a secret pair of master keys to produce the SPIs and encryption keys.

One key architectural benefit of PSP over similar protocols (e.g., IPsec) is that the receiver does not incur any additional resource utilization whenever it creates a new SPI and encryption key. This is because the decryption key associated with the SPI is computed on the fly, based on the SPI and master key, for each received packet. This lack of requirement for additional context memory for each additional decryption rule is partly responsible for the ability of the PSP protocol to scale to many thousands of peers.

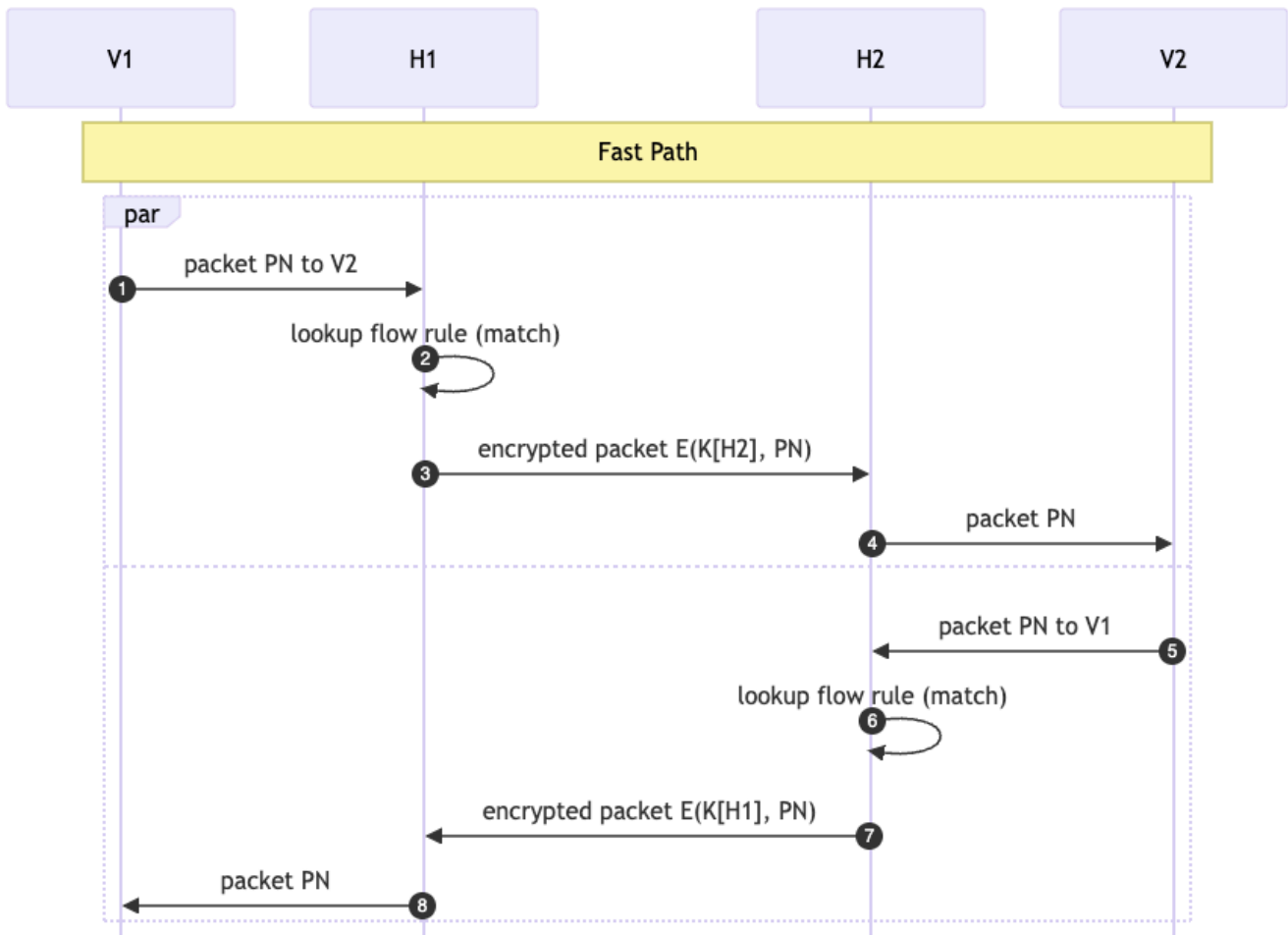
Startup vs. On-Demand Tunnel Creation

The default mode of operation is on-demand tunnel creation. That is, when a packet is received from the unsecured port for which the flow pipeline does not have an encryption rule, the packet misses to RSS, where the CPU must decide how to handle the packet. If the destination IP address in the packet belongs to a known peer's virtual network, the CPU uses gRPC on the OOB network connection to attempt a key exchange with the peer. If the key exchange is successful and a new encryption flow is created successfully, then the packet is then resubmitted to the pipeline, where it is encrypted and sent just as any of the following packets having the same destination IP address.

The following diagram illustrates this sequence (the "Slow Path"), for Virtual Machine V1 which intends to send a packet to Virtual Machine V2. In this case, V1 is hosted on physical host H1 and V2 on physical host H2. The first packet sent (1) results in a miss (2), so the packet is retained (3) while the keys are exchanged in both directions (4-8). Then the pipeline is updated (9) and the original packet is resubmitted (10). From there, the packet follows the same logic as the fast path, below.



Once the tunnel is established, and packets received from the VF (1) match a rule (2) and are encrypted and sent (3-4) without any intervention from the CPU ("Fast Path").



In the case of on-startup tunnel creation, the application's main thread repeatedly attempts to perform the key exchange for each of the peers specified on the command line until the list is completed. Each peer is connected only once and, if a connection to one peer fails, the loop continues onto the next peer and retries the failed connection after all the others have been attempted.

Sampling

The PSP gateway application supports the sample-at-receiver (S) bit in the PSP header. If sampling is enabled, then packets marked with the S bit are mirrored to the RSS queues and logged to the console. In addition, on transmit, a random subset of packets (1 out of 2^N for command-line parameter N) will have the S bit set to 1, and those packets will also be mirrored to RSS.

(i) Note

Sampling packets on transmit is currently supported only following encryption. Sampling of egress packets before encryption will be supported in a future release.

Pipelines

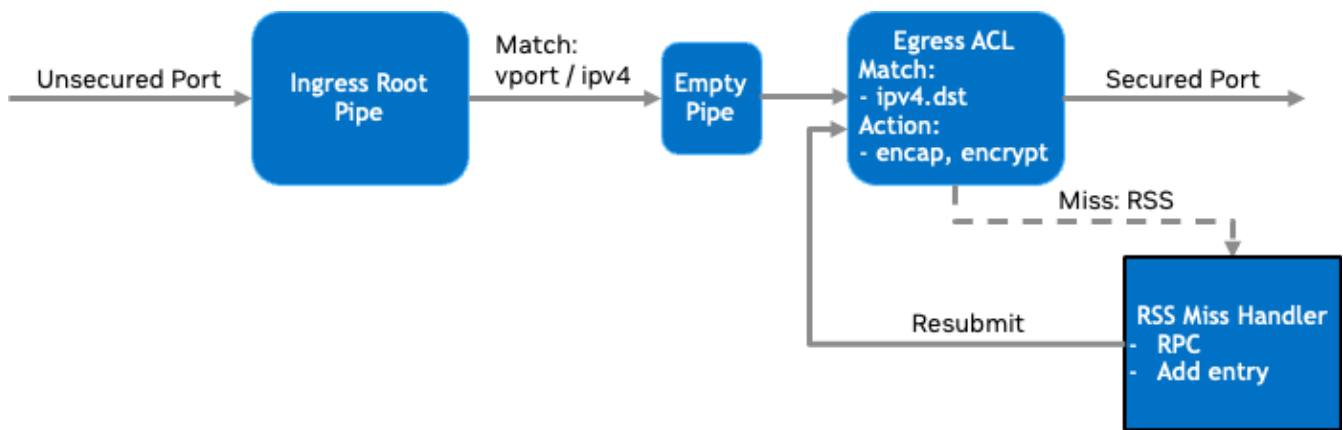
Host-to-Network Flows

Traffic sent from the local, unsecured port (host-to-net) without sampling enabled travels through the pipeline as shown in the diagrams that follow. Note that the Ingress Root Pipe is the first destination for packets arriving from either the VF or the secured uplink port. However, the Egress ACL pipe is the first destination for packets sent via `tx_burst` on the PF (in the switch model's expert mode).

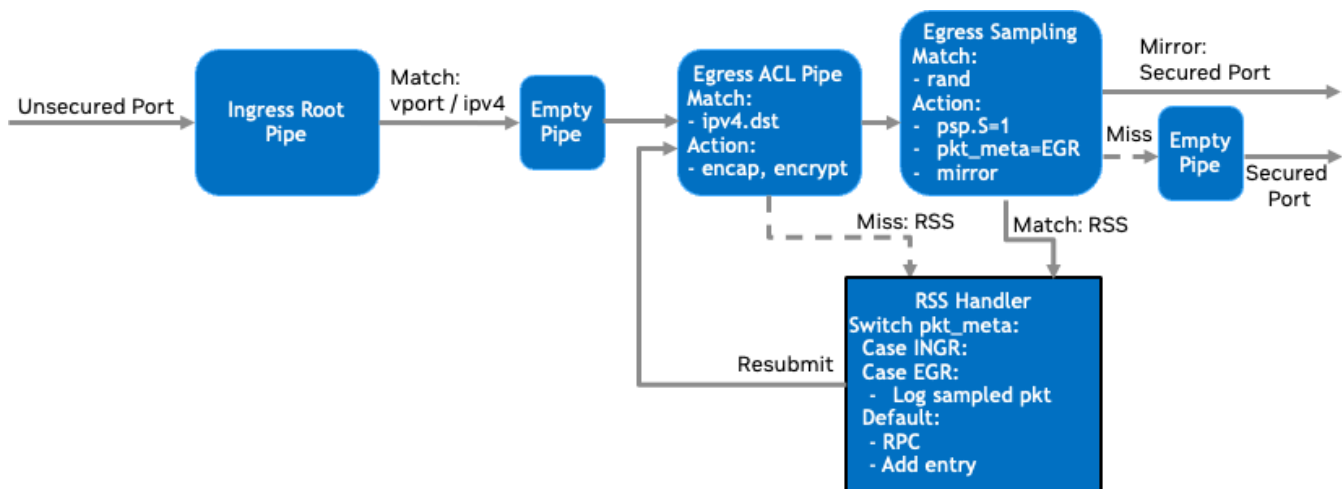
The Empty Pipe is a vestigial transition from the Default Domain, in which the Ingress Root Pipe is created, to the Secure Egress Domain, where the Egress ACL pipe performs encryption.

(i) Note

This pipe may be removed in a future release.



If sampling is enabled, the host-to-net pipeline is modified as shown in the following:



Here, an Egress Sampling Pipe is added between the Egress ACL Pipe and the Secured Port. It performs a match of the random metadata, masked according to command-line parameters, and then:

- On match, the following actions occur:
 1. Packet modifications:
 1. The S bit in the PSP header is set to true.
 2. The pkt_meta field is set to a sentinel value to indicate to CPU software why the packet was sent to RSS.
 2. The original packet is forwarded to RSS.
 3. The mirror action forwards the packet to the secured port.

- On miss, the following actions occur:
 1. No packet modifications are made.
 2. The packet is forwarded to a vestigial pipe which can then forward the packet to the wire.

Info

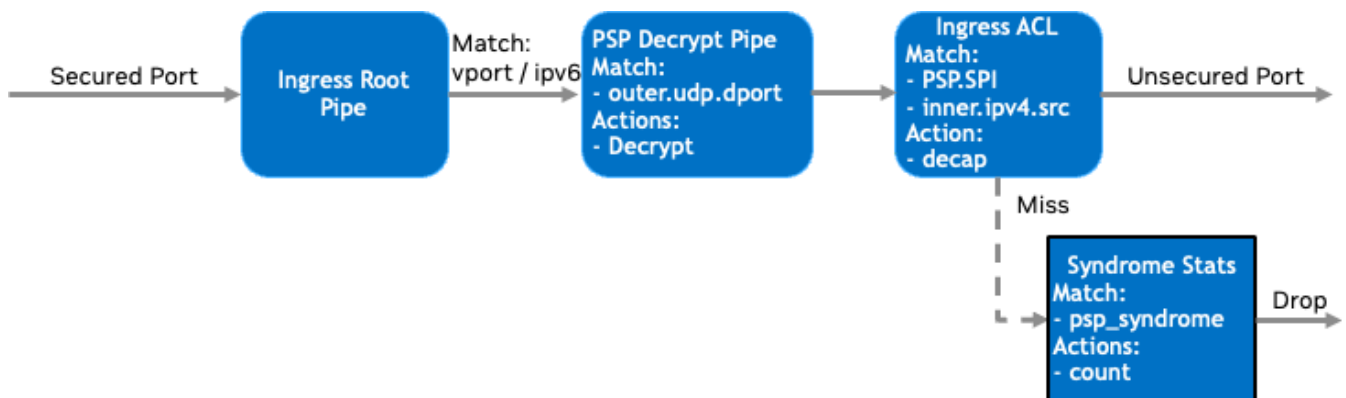
A fwd_miss cannot target a port.

Note

This pipe may be removed in a future release.

Network-to-Host Flows

When a packet arrives from the secured port, the following flows are executed.



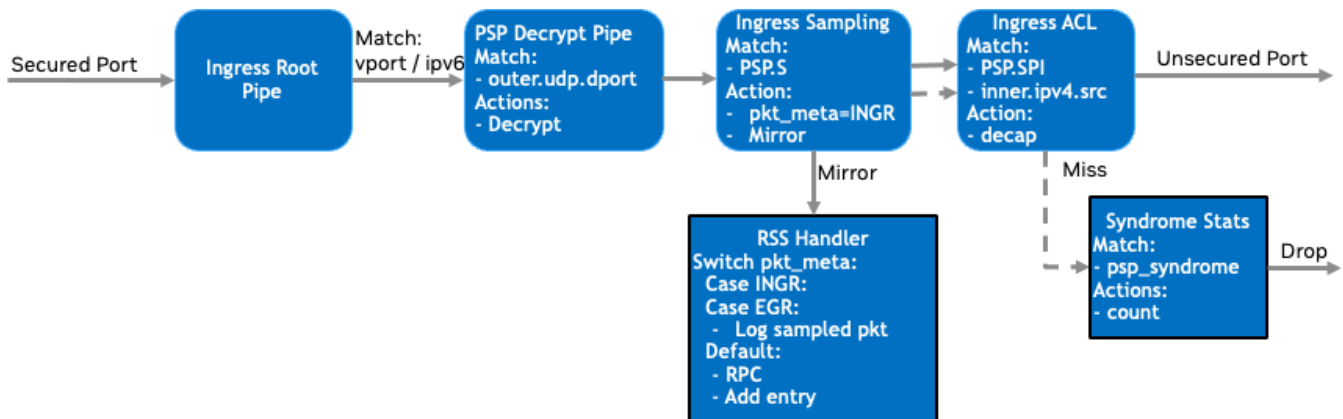
As before, the Ingress Root Pipe is the first destination and, here, the secured port ID as well as IPv6 outer L3 type are matched for. Matching packets flow to the decryption pipe, which matches the outer UDP port number against 1000, the constant specified in the

PSP specification. On match, the packet is decrypted, but not yet de-capped. Then the Ingress ACL pipe checks the following:

- PSP_Syndrome – did the packet decrypt correctly and pass its ICV check?
- PSP SPI and inner IP src address – was this packet encrypted with the key associated with the given source?

If the packet passes the syndrome and ACL check, it is forwarded to the VF. Otherwise, the Syndrome Stats pipe counts the occurrences of the different bits in the PSP Syndrome word.

When sampling is enabled, the Ingress Sampling Pipe is inserted before the ACL. Unlike the Egress Sampling Pipe, no randomness is involved; the match criteria is the sample-on-receive flag in the PSP header. On a match, the incoming packet are mirrored to RSS with `pkt_meta` indicating the reason for forwarding the packet to RSS. On match or miss, the next pipe is the Ingress ACL Pipe.



DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA Flow](#)

Refer to their respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/psp_gateway/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_psp_gateway is created under /tmp/build/psp_gateway/.

Compiling Only the Current Application

To directly build only the PSP Gateway application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_psp_gateway=true  
ninja -C /tmp/build
```

Info

doca_psp_gateway is created under /tmp/build/psp_gateway/.

Alternatively, users can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:

- Set enable_all_applications to false
- Set enable_psp_gateway to true

2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

`doca_psp_gateway` is created under `/tmp/build/psp_gateway/`.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Prerequisites

The PSP gateway application is based on DOCA Flow. Therefore, the user is required to allocate huge pages:

```
echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

Note

On some OSs (e.g., RockyLinux, OpenEuler, CentOS 8.2), the default huge page size on the BlueField (and Arm hosts) is larger than 2MB, and is often 512MB instead. The user can find out the size of the huge pages using the following command:

```
$ grep -i huge /proc/meminfo
```

```
AnonHugePages:    0 kB  
ShmemHugePages:  0 kB  
FileHugePages:   0 kB
```

```
HugePages_Total: 4
HugePages_Free: 4
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 524288 kB
Hugetlb: 6291456 kB
```

Given that the guiding principle is to allocate 4GB of RAM, in such cases instead of allocating 2048 pages, the user should allocate the matching amount (8 pages):

```
echo '8' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-
524288kB/nr_hugepages
```

Application Execution

The PSP Gateway application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_psp_gateway [DOCA Flags] [Program Flags]
```

DOCA Flags:

-h, --help Print a help synopsis

-v, --version Print program version information

-l, --log-level Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

-j, --json <path> Parse all command flags from an input json file

Program Flags:

-p, --pci-addr PCI BDF of the device in BB:DD.F format
-r, --repr Device representor list in vf[x-y]pf[x-y] format
-m, --core-mask EAL Core Mask
-d, --decap-dmac mac_dst addr of the decapped packets
-n, --nexthop-dmac next-hop mac_dst addr of the encapped packets
-s, --svc-addr Service address of locally running gRPC server; port number optional
-t, --tunnel Remote host tunnel(s), formatted 'mac-addr,phys-ip,virt-ip'
-f, --tunnels-file Specifies the location of the tunnels-file. Format: rpc-addr:virt-addr,virt-addr,...
-c, --cookie Enable use of PSP virtualization cookies
-a, --disable-ingress-acl Allows any ingress packet that successfully decrypts
-, --sample-rate Sets the log2 sample rate: 0: disabled, 1: 50%, ... 16: 1.5e-3%
-x, --max-tunnels Specify the max number of PSP tunnels
-o, --crypt-offset Specify the PSP crypt offset
--psp-version Specify the PSP version for outgoing connections (0 or 1)
-z, --static-tunnels Create tunnels at startup using the given local IP addr
-k, --debug-keys Enable debug keys

2. This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_psp_gateway -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

3. CLI example for running the application on the BlueField or host:

```
./doca_psp_gateway -p 03:00.0 -r vf0 -d 11:22:33:44:55:66 -t  
10.1.1.55:192.168.1.55
```

- The PCIe address identifier (-p flag) should match the addresses of the desired PCIe device
 - The -d flag indicates the MAC address that should be applied to incoming packets upon decap. It should match the MAC address of the virtual function specified by the -r argument.
 - The -t flag indicates the mapping of the virtual IP address 192.168.x.y to an out-of-bounds network address 10.1.1.55
4. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_psp_gateway --json [json_file]
```

For example:

```
./doca_psp_gateway --json psp_gateway_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with TRACE log level support) 	<pre>"log-level": 60</pre>
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	<pre>"sdk-log-level": 40</pre>
	j	json	Parse all command flags from an input JSON file	N/A
Program flags	p	pci-addr	PCIe BDF of the device in BB:DD.F format	<pre>"p": "03:00.0"</pre>
	r	repr	Device representor list in vf[x-y]pf[x-y] format	<pre>"r": "vf0"</pre>
	m	core-mask	EAL core mask	<pre>"m": "0xf"</pre>
	d	decap-dmac	mac_dst address of the decapped packets	<pre>"decap-dmac": "11:22:33:"</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
				44:55:66"
	n	nexthop-dmac	Next-hop mac_dst address of the encapped packets	"nexthop-dmac": "77:88:99:aa:bb:cc"
	s	svc-addr	Service address of locally running gRPC server; port number optional	"svc-addr": "10.1.1.50"
	t	tunnel	Remote host tunnel(s), formatted rpc-addr:virt-addr	"tunnel": "10.1.1.55:192.168.1.100"
	f	tunnels-file	Specifies the location of the tunnels-file. Format: rpc-addr:virt-addr,virt-addr,...	"tunnels-file": "tunnels.txt"
	c	cookie	Enable use of PSP virtualization cookies	"cookie": true
	a	disable-ingress-acl	Allows any ingress packet that successfully decrypts	"disable-ingress-acl": true
	N/A	sample-rate	Sets the log2 sample rate: <ul style="list-style-type: none"> • 0 – disabled, • 1 – 50%, ... • 16 – 1.5e-3% 	"sample-rate": 16
	x	max-tunnels	Specify the max number of PSP tunnels	"max-tunnels": 4096
	o	crypt-offset	Specify the PSP crypt offset	"crypt-offset": 7

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	N/A	psp-version	Specify the PSP version for outgoing connections (0 or 1)	"psp-version": 0
	z	static-tunnels	Create tunnels at startup using the given local IP address	"static-tunnels": "192.168.1.99"
	k	debug-keys	Enable debug keys	"debug-keys": true

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Tunnel Mappings File

A text file which maps an OOB network address to a list of virtual IP addresses behind that physical address can be specified on the command line. The format is as follows:

```
# (Comments are allowed)
# Format:
# svc-oob-ip-addr:virt-addr,virt-addr,...
# Specify a service address of 10.1.1.55 which hosts virtual addresses 192.168.1.101 and
others.
10.1.1.55:192.168.1.101,192.168.1.102,192.168.1.103,192.168.1.104
# Specify a service address of 10.1.1.56 which hosts virtual addresses 192.168.1.201 and
others.
10.1.1.56:192.168.1.201,192.168.1.202,192.168.1.203,192.168.1.204
```

When a packet from the VF does not match any existing flows, this table defines the physical host which should provide the tunnel to the given (virtual) destination.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications .

Application Code Flow

1. Main loop code flow

1. Initialize the logger facility.

1. The standard logger and the SDK logger are created, and the SDK logger default log level is selected.

```
doca_log_backend_create_standard();  
doca_log_backend_create_with_file_sdk(stdout, &sdk_log);  
doca_log_backend_set_sdk_level(sdk_log,  
DOCA_LOG_LEVEL_WARNING);
```

2. The signal handler is connected to enable a clean shutdown.

```
signal(SIGINT, signal_handler);  
signal(SIGTERM, signal_handler);
```

2. Parse application arguments. The main function invokes `psp_gw_argp_exec()`, which initializes the arg parser resources and registers DOCA general parameters, and then registers the PSP application-specific parameters. Then the parser is invoked.

```
doca_argp_init();  
psp_gw_register_params();  
doca_argp_start();
```

3. DPDK initialization. Call `rte_eal_init()` to initialize EAL resources with the provided EAL flags for not probing the ports (`-a00:0.0`).

```
rte_eal_init(n_eal_args, (char **)eal_args);
```

4. Initialize devices and ports.

1. Open DOCA devices with input PCIe addresses/interface names.
2. Probe DPDK port from each opened device.

```
open_doca_device_with_pci(...); // not part of doca_flow; see  
doca/samples/common.c  
doca_dpdk_port_probe(...);
```

3. The MAC and IP addresses of the PF are queried and logged.

```
rte_eth_macaddr_get(...);  
doca_devinfo_get_ipv6_addr(...);  
DOCA_LOG_INFO("Port %d: Detected PF mac addr: %s, IPv6 addr: %s, total  
ports: %d", ...);
```

5. Initialize and start DPDK ports. Initialize DPDK ports, including mempool allocation. No hairpin queues are created.

```
dpdk_queues_and_ports_init(); // not part of doca_flow; see  
doca/applications/common/dpdk_utils.c
```

6. Initialize DOCA Flow objects used by the PSP Gateway application. The DOCA Flow library is initialized with the string `"switch,hws,isolated,expert"`, because it is desirable for the application to act as an intermediary between the uplink physical port and some number of VF representors (switch mode), and `hws` (hardware steering mode) and `isolated` mode are mandatory for switch mode. The optional `expert` flag prevents DOCA Flow from automating certain packet operations and gives more control to the application, as described in the [DOCA Flow](#) page.

```
PSP_GatewayFlows psp_flows(&pf_dev, vf_port_id, &app_config);
psp_flows.init();
```

1. Initialize DOCA Flow library.
 2. Start the ports.
 3. Allocate shared resources (PSP crypto objects and Mirror actions).
 4. Create the ingress and egress pipes.
7. Create the gRPC service.

```
PSP_GatewayImpl psp_svc(&app_config, &psp_flows);
```

8. Launch the L-Core threads to handle RSS packets.

```
rte_eal_remote_launch(lcore_pkt_proc_func, &lcore_params, lcore_id);
```

9. Launch the gRPC service.

1. This implementation uses `InsecureServerCredentials`. Update as needed.

```
grpc::ServerBuilder builder;
builder.AddListeningPort(server_address,
grpc::InsecureServerCredentials());
builder.RegisterService(&psp_svc);
auto server_instance = builder.BuildAndStart();
```

10. Wait for traffic. If configured to connect at startup, process the list of remaining connections. Then display the flow pipe counters.

```
while (!force_quit) {
psp_svc.try_connect(remotes_to_connect, local_vf_addr);
...
psp_flows.show_static_flow_counts();
```

```
    psp_svc.show_flow_counts();  
}
```

- Wait in a loop until the user terminates the program.

11. PSP Gateway cleanup:

1. Destroy DPDK ports and queues.

```
    dpdk_queues_and_ports_fini();
```

2. DPDK finish.

```
    dpdk_fini();
```

Calls `rte_eal_destroy()` to destroy initialized EAL resources.

3. Arg parser destroy.

```
    doca_argp_destroy();
```

2. Miss-packet code flow.

1. The L-Core launch routine from the main loop pointed to the `lcore_pkt_proc_func` routine.
2. The `force_quit` flag is polled to respond to the signal handler.

```
while (!*params->force_quit) { ... }
```

3. The `rte_eth_rx_burst` function polls the PF queue for received packets.

```
nb_rx_packets = rte_eth_rx_burst(port_id, queue_id, rx_packets,  
MAX_RX_BURST_SIZE);
```

4. Inside `handle_packet()`, the packet metadata is inspected to detect whether this packet is sampled on ingress, sampled on egress, or a miss packet.

```
uint32_t pkt_meta = rte_flow_dynf_metadata_get(packet);
```

1. Sampled packets are simply logged using the `rte_pktmbuf_dump` function.
5. Miss packets are passed to the `handle_miss_packet` method of the gRPC service. This method handles cases where an application attached to the VF wishes to send a packet to another virtual address, but a PSP tunnel must first be established by exchanging SPI and key information between hosts.
6. The service acts as a gRPC client, and the appropriate server is looked up from the `config->net_config.hosts` vector, which is comprised of hosts passed via the `-t tunnels` arguments or the `-f tunnels` file argument.
7. Once the client connection exists, the `request_tunnel_to_host` method takes care of invoking the the `RequestTunnelParams` operation defined in the schema.
 - Optionally, this function generates a corresponding set of tunnel parameters appropriate for the server host to send traffic back via `generate_tunnel_params()`.

```
doca_flow_crypto_psp_spi_key_bulk_generate(bulk_key_gen);  
doca_flow_crypto_psp_spi_key_bulk_get(bulk_key_gen, 0, &spi, key);  
doca_flow_crypto_psp_spi_key_wipe(bulk_key_gen, 0);
```

8. The RPC operation is invoked, and if successful, `create_tunnel_flow` is called to create the egress flow:

```
status = stub->RequestTunnelParams(&context, request, &response);
```

9. The `create_tunnel_flow` method translates the resulting Protobuf objects to application-specific data structures and passes them to the `add_encrypt_entry` method of the flows object. Here, the PSP SPI and key are programmed into an available `crypto_id` index as follows.

(i) Note

SPI and `crypto_id` are two independent concepts:

- The SPI value in the PSP packet header indicates to the receiver which key was used by the sender to encrypt the data. Each receiver computes an SPI and key to provide to a sender. Since each receiver is responsible for tracking its next SPI, multiple receivers may provide the same SPI to a sender, so one sender may send the same SPI to multiple different peers. This is allowed, as each of the receiving peers has its own decryption key to handle that SPI.
- The `crypto_id` acts as an index into the bucket of PSP keys allocated by DOCA Flow. The `doca_flow_shared_resource_cfg()` function writes a given PSP encryption key to a given slot in the bucket of keys in NIC memory. These slots can be overwritten as needed by the application.
- There is no explicit association between `crypto_id` and SPI. The `doca_flow_shared_resource_cfg()` function writes a key at the slot provided by the `crypto_id` argument, then the flow pipe entry `actions.crypto.crypto_id` references this key, and `actions.crypto_encap.encap_data` includes a PSP header with the desired SPI.

```
struct doca_flow_shared_resource_cfg res_cfg = {};  
res_cfg.domain = DOCA_FLOW_PIPE_DOMAIN_SECURE_EGRESS;  
res_cfg.psp_cfg.key_cfg.key_type = DOCA_FLOW_CRYPTOKEY_256;  
res_cfg.psp_cfg.key_cfg.key = (uint32_t *)encrypt_key;  
doca_flow_shared_resource_cfg(DOCA_FLOW_SHARED_RESOURCE_PSP,  
session->crypto_id, &res_cfg);
```

10. A flow pipe entry which references the newly programmed PSP encryption key (via its index `crypto.crypto_id`) must be inserted. Additionally, this pipe entry must specify all the outer Ethernet, IP, UDP, and PSP header fields to insert.

```
format_encap_data(session, actions.crypto_encap.encap_data);
actions.crypto.action_type = DOCA_FLOW_CRYPT_ACTION_ENCRYPT;
actions.crypto.resource_type = DOCA_FLOW_CRYPT_RESOURCE_PSP;
actions.crypto.crypto_id = session->crypto_id;
...

doca_flow_pipe_add_entry(pipe_queue, pipe, match, actions, mon, fwd,
flags, &status, entry);
...

doca_flow_entries_process(port, 0, DEFAULT_TIMEOUT_US,
num_of_entries);
```

11. The original packet received via `rte_ethdev_rx_burst` is sent back through the newly updated pipelines via `rte_ethdev_tx_burst`. Since the `port_id` argument is that of the PF, and since DOCA Flow has been initialized in expert mode, the packet is transferred to the root of the egress domain (the "empty pipe" before `egress_acl_pipe`).

```
nsent = rte_eth_tx_burst(port_id, queue_id, &packet, 1);
```

3. Tunnel parameter request handling

1. The gRPC service provided by the PSP Gateway implements the `RequestTunnelParams` operation referenced above. A client uses this operation to request an SPI and key to encrypt traffic to send to the server's NIC device. The request indicates the virtual remote address for which the tunnel will be created.
2. This operation begins by generating a new SPI and key inside `generate_tunnel_params()` as described previously.
3. The operation creates an ACL entry permitting the new SPI and the remote virtual address using the `add_ingress_acl_entry` method of the Flows object.

```
doca_flow_match match = {};  
match.parser_meta.psp_syndrome = 0;  
match.tun.type = DOCA_FLOW_TUN_PSP;  
match.tun.psp.spi = RTE_BE32(session->spi_ingress);  
match.inner.l3_type = DOCA_FLOW_L3_TYPE_IP4;  
match.inner.ip4.src_ip = session->src_vip;  
...  
  
doca_flow_pipe_add_entry(pipe_queue, pipe, match, actions, mon, fwd,  
flags, &status, entry);  
...  
  
doca_flow_entries_process(port, 0, DEFAULT_TIMEOUT_US,  
num_of_entries);
```

4. If the request included parameters for traffic in the reverse direction (traffic to encrypt and send to the client), these parameters are translated and passed to the Flows object by calling `create_tunnel_flow` described [above](#).

References

- [PSP Security Protocol Specification](#)
- [Google's Open-Source PSP tools](#)
- [Google Remote Procedure Calls library](#)

NVIDIA DOCA Secure Channel Application Guide

This guide provides a secure channel implementation on top of NVIDIA® BlueField® DPU.

Introduction

The DOCA Secure Channel reference application leverages the [DOCA Comch](#) API which creates a secure, network independent communication channel between the host and the NVIDIA BlueField DPU.

Comm channel allows the host to control services on the DPU, activate certain offloads, or exchange messages using client-server framework.

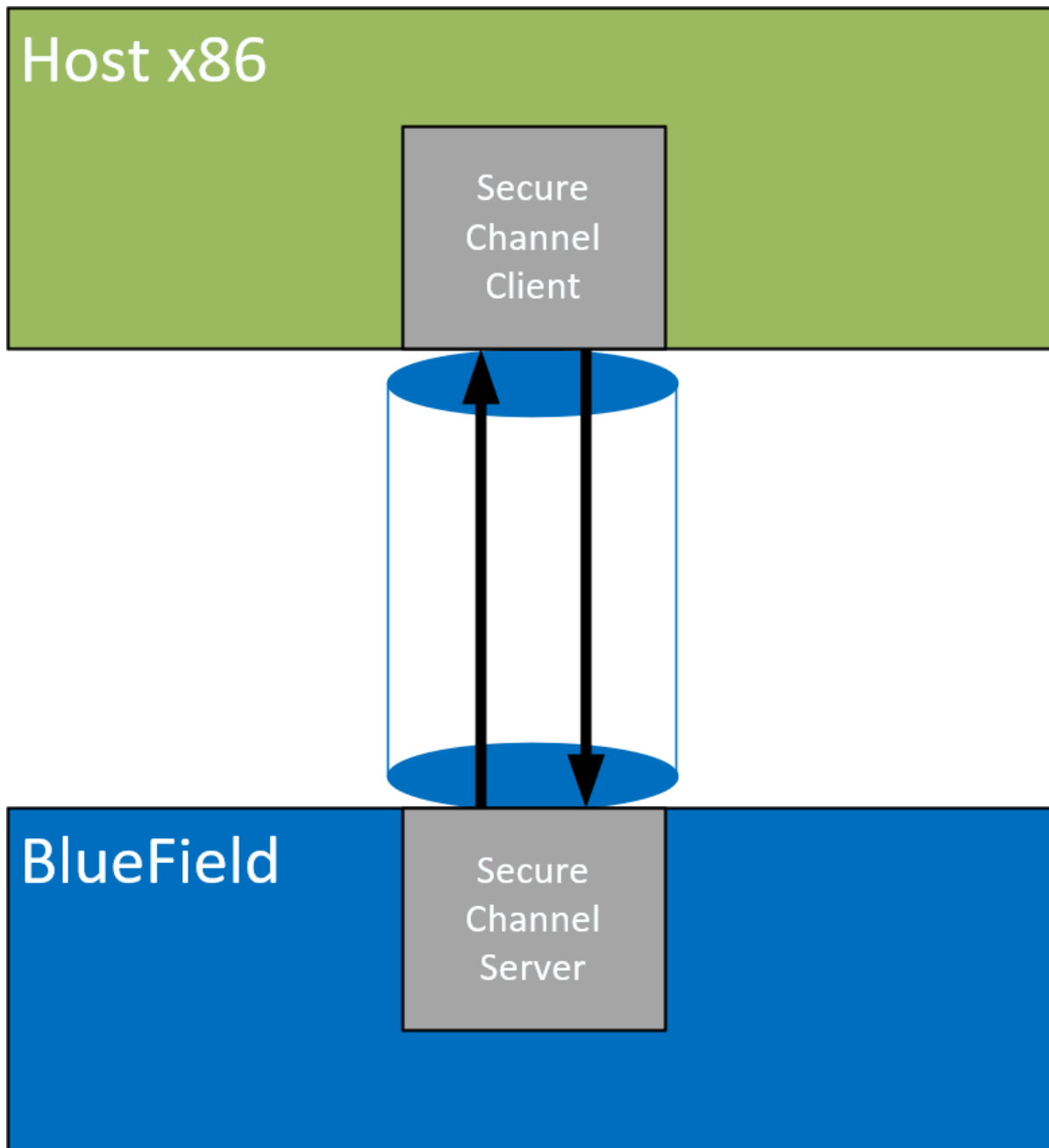
The client (host) side is able to communicate only with one server at a time while the server side is able to communicate with multiple clients.

The API allows communication between any PF/VF/SF on the host to the server located on the DPU.

Secure channel allows the user to select the message size and amount to be exchanged between the client and the server to simulate heavy load on the channel.

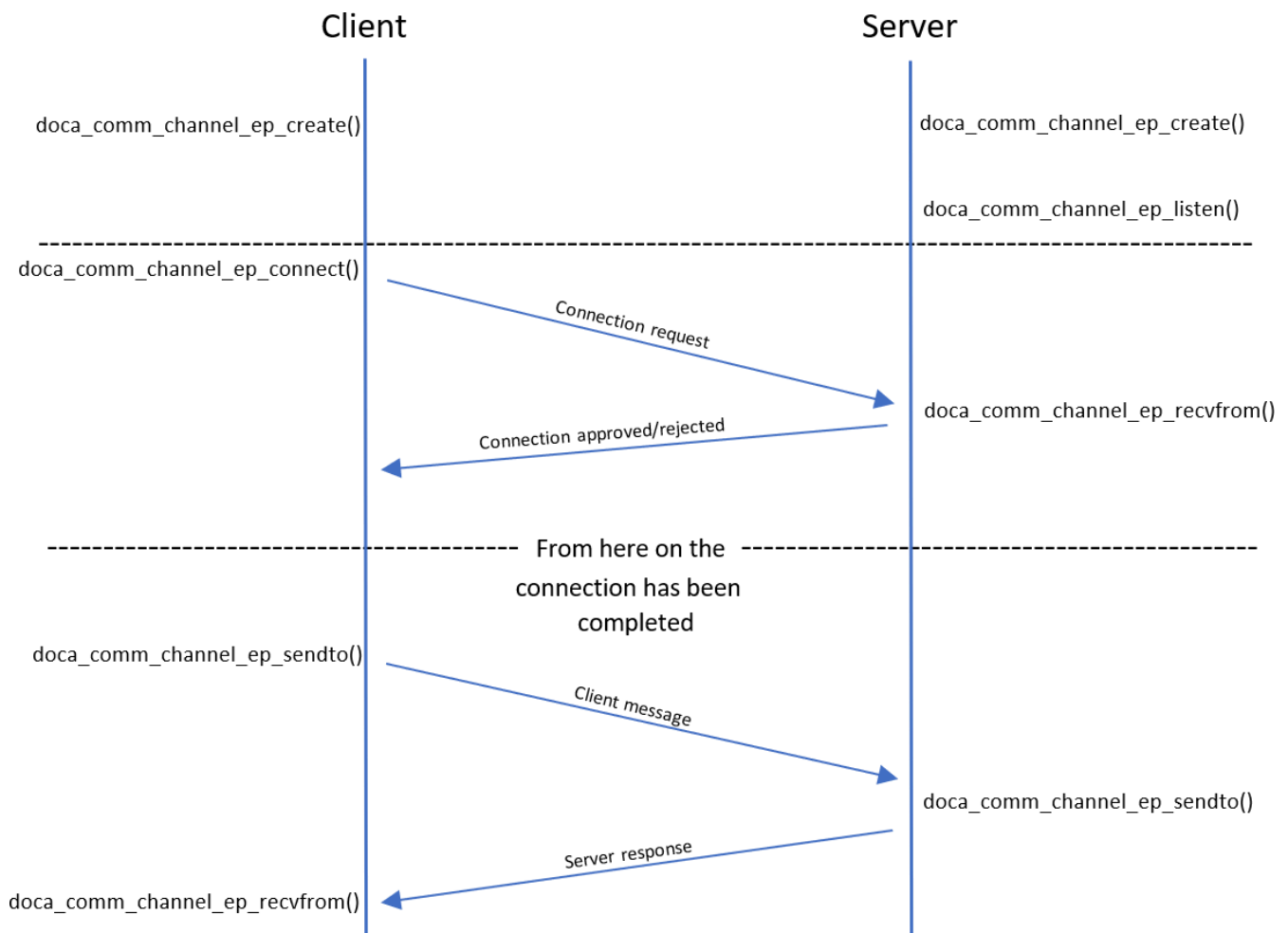
System Design

A secure channel application runs on client mode (host) and server mode (DPU). Once a channel is open, messages can flow from both sides.



Application Architecture

The secure channel application runs on top of the DOCA Comm Channel API. Full connection flow between the client and the server is illustrated in the following:



1. Both sides initiate create().
2. Server listens and waits for new connections.
3. Server initiates rcvfrom() to indicate it is ready to exchange messages
4. Client executes connect() to server and starts connection initialization.
5. Client sends first message to server.
6. Server sends a response.

DOCA Libraries

This application leverages the following DOCA library:

- [DOCA Comch](#)

Refer to its respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/secure_channel/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build
```

```
ninja -C /tmp/build
```

Info

`doca_secure_channel` is created under `/tmp/build/secure_channel/`.

Compiling Only the Current Application

To directly build only the secure channel application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_secure_channel=true  
ninja -C /tmp/build
```

Info

`doca_secure_channel` is created under `/tmp/build/secure_channel/`.

Alternatively, users can set the desired flags in the `meson_options.txt` file instead of providing them in the compilation command line:

1. Edit the following flags in `/opt/mellanox/doca/applications/meson_options.txt`:
 - Set `enable_all_applications` to `false`
 - Set `enable_secure_channel` to `true`
2. Run the following compilation commands :


```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_secure_channel is created under /tmp/build/secure_channel/.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Application Execution

The secure channel application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_secure_channel [DOCA Flags] [Program Flags]
```

DOCA Flags:

-h, --help Print a help synopsis

-v, --version Print program version information

-l, --log-level Set the (numeric) log level for the program <10=DISABLE,
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
-j, --json <path> Parse all command flags from an input json file

Program Flags:

-s, --msg-size Message size to be sent
-n, --num-msgs Number of messages to be sent
-p, --pci-addr DOCA Comm Channel device PCI address
-r, --rep-pci DOCA Comm Channel device representor PCI address (needed only on DPU)

i Info

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_secure_channel -h
```

i Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the BlueField:

```
./doca_secure_channel -s 256 -n 10 -p 03:00.0 -r 3b:00.0
```

i Note

Both the DOCA Comm Channel device PCIe address (03:00.0) and the DOCA Comm Channel device representor PCIe address (3b:00.0) should match the addresses of the desired PCIe devices.

3. CLI example for running the application on the host:

```
./doca_secure_channel -s 256 -n 10 -p 3b:00.0
```

Note

The DOCA Comm Channel device PCIe address (3b:00.0) should match the address of the desired PCIe device.

4. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_secure_channel --json [json_file]
```

For example:

```
./doca_secure_channel --json ./sc_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with TRACE log level support) 	<pre>"log-level": 60</pre>
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	<pre>"sdk-log-level": 40</pre>
	j	json	Parse all command flags from an input JSON file	N/A

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
Program flags	s	msg-size	<p>Message size in bytes</p> <p>Note This is a mandatory flag.</p>	<pre>"msg-size": 128</pre>
	n	num-msgs	<p>Number of messages to send on both sides</p> <p>Note This is a mandatory flag.</p>	<pre>"num-msgs": 256</pre>
	p	pci-addr	<p>DOCA Comm Channel device PCIe address</p> <p>Note This is a mandatory flag.</p>	<pre>"pci-addr": 03:00.1</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	r	rep-pci	DOCA Comm Channel device representor PCIe address <div style="background-color: #ffffcc; padding: 10px; border: 1px solid #ccc;"> <p>i Note This is a mandatory flag only on the DPU.</p> </div>	<pre>"rep-pci": b1:00.1</pre>

i Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications .

Application Code Flow

1. Parse application argument.
 1. Initialize the arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register secure channel application parameters.

```
register_secure_channel_params();
```

3. Parse application parameters:

```
doca_argp_start();
```

2. Run main logic.

```
sc_start();
```

1. Initiate synchronization mechanism between send and receive threads.
2. Initiate Comm Channel endpoint.
3. Server side starts listening for new connections and client side connects to server.
4. Initiate signal masking and epoll file descriptor.
5. Start send and receive threads. Both threads share the same Comm Channel so each one must "lock" the channel before any send/receive operation.
6. Send thread prints total number of messages successfully sent.
7. Once Ctrl+C is entered in the shell, receive thread prints the total number of messages successfully received.
8. Close and destroy resources.

References

- [/opt/mellanox/doca/applications/secure_channel/](#)

- /opt/mellanox/doca/applications/secure_channel/sc_params.json

NVIDIA DOCA Simple Forward VNF Application Guide

This guide provides a Simple Forward implementation on top of NVIDIA® BlueField® DPU.

Introduction

Simple forward is a forwarding application that leverages the [DOCA Flow](#) API to take either VXLAN, GRE, or GTP traffic from a single RX port and transmits it on a single TX port.

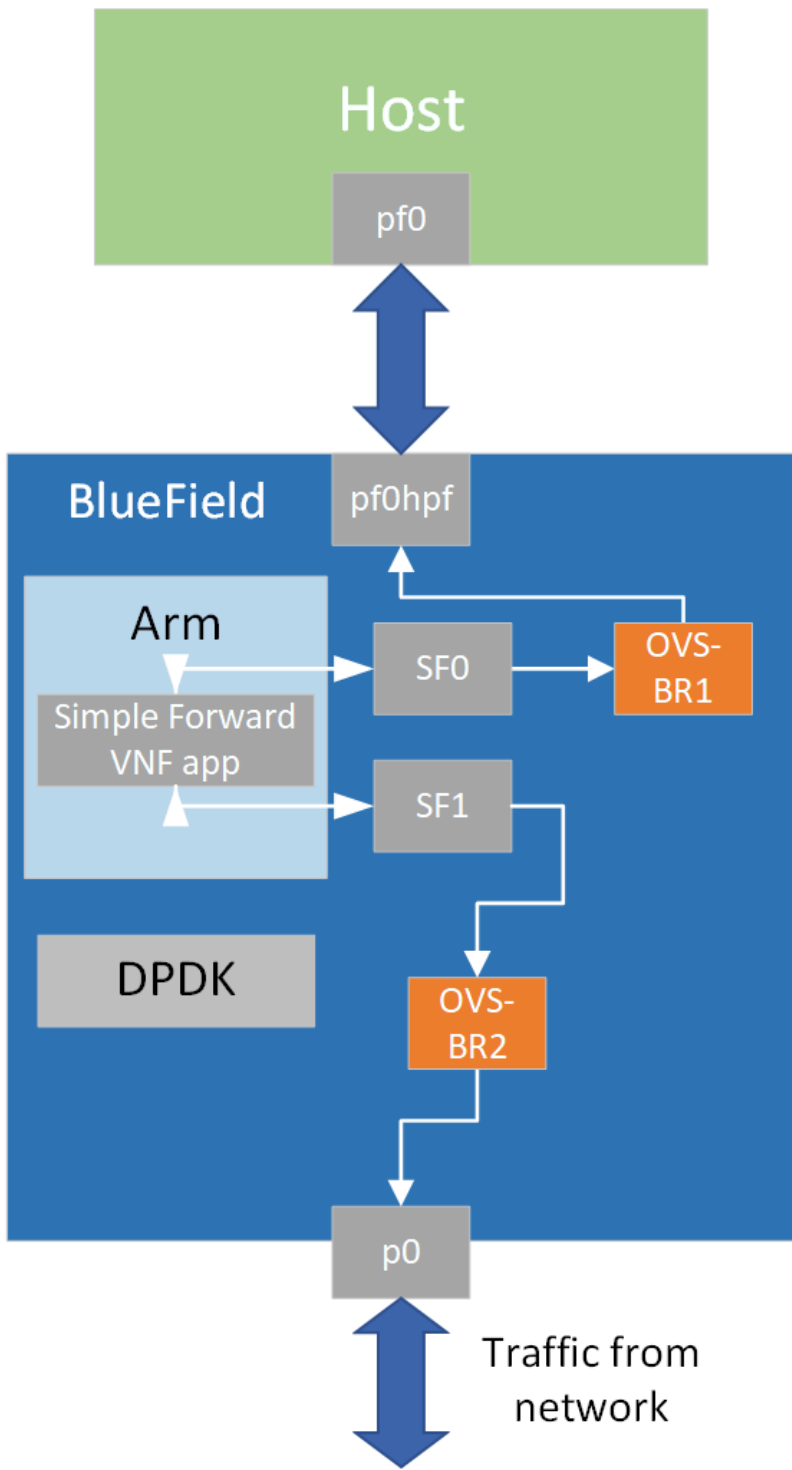
For every packet received on an RX queue on a given port, DOCA Simple Forward checks the packet's key, which consists of a 5-tuple. If it finds that the packet matches an existing flow, then it does not create a new one. Otherwise, a new flow is created with a FORWARDING component. Finally, the packet is forwarded to the TX queue of the egress port if the "rx-only" mode is not set.

The FORWARDING component type depends on the flags delivered when running the application. For example, if the `hairpinq` flag is provided, then the FORWARDING component would be hairpin. Otherwise, it would be RSS'd to software, and hence every VXLAN, GTP, or GRE packet would be received on RX queues.

Simple forward should be run with dual ports. By using a traffic generator, the RX port receives the VXLAN, GRE, or GTP packets and forwarding forwards them back to the traffic generator.

System Design

The following diagram illustrates simple forward's packet flows. It receives traffic coming from the wire and passes it to the other port.



Application Architecture

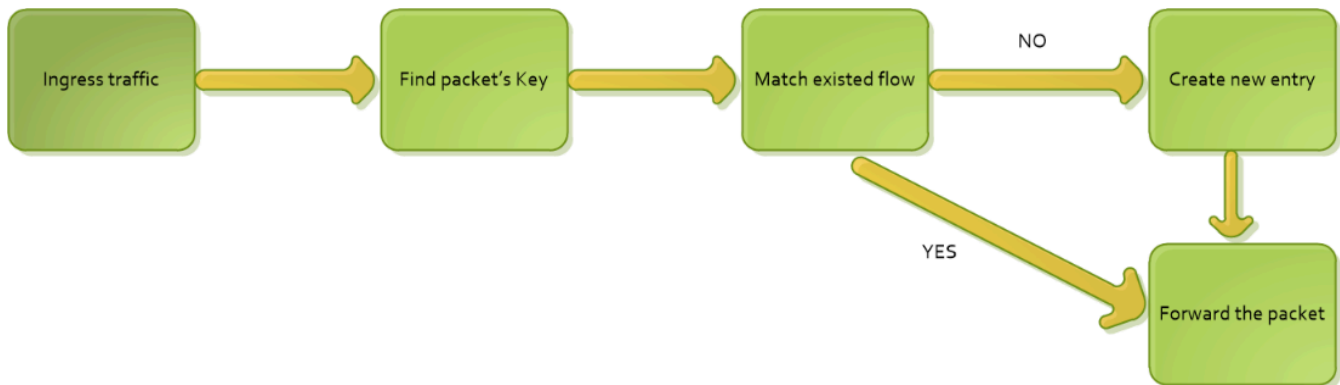
Simple forward first initializes DPDK, after which the application handles the incoming packets.

The following diagram illustrates the initialization process.



1. Init_DPDK – EAL init, parse argument from command line and register signal.
2. Start port – mbuf_create, dev_configure, rx/tx/hairpin queue setup and start the port.
3. Simple_fwd INIT – create flow tables, build default forward pipes.

The following diagram illustrates how to process the packet.



1. Based on the packet's info, find the key values (e.g. src/dst IP, src/dst port, etc).
2. Traverse the inner flow tables, check if the keys exist or not.
 - If yes, update inner counter
 - If no, a new flow table is added to the DPU
3. Forward the packet to the other port.

DOCA Libraries

This application leverages the following DOCA library:

- [DOCA Flow](#)

Refer to its respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/simple_fwd_vnf/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_simple_fwd_vnf is created under /tmp/build/simple_fwd_vnf/.

Compiling Simple Forward Application Only

To directly build only the simple forward application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_simple_fwd_vnf=true  
ninja -C /tmp/build
```

Info

doca_simple_fwd_vnf is created under /tmp/build/simple_fwd_vnf/.

Alternatively, users can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
 - Set enable_all_applications to false
 - Set enable_simple_fwd_vnf to true
2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build
```

```
ninja -C /tmp/build
```

Info

`doca_simple_fwd_vnf` is created under `/tmp/build/simple_fwd_vnf/`.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Prerequisites

1. A FLEX profile number should be manually set to 3 on the system for the application to build the GRE, standard VXLAN and GRE pipes.

1. Set FLEX profile number to 3 from the DPU.

```
sudo mlxconfig -d <pcie_address> s FLEX_PARSER_PROFILE_ENABLE=3
```

2. Perform a [BlueField system reboot](#) for the `mlxconfig` settings to take effect.

Info

Resetting the firmware can be done from the BlueField as well. For more information, refer to step 3.b of the

"Upgrading Firmware" section of the [NVIDIA DOCA Installation Guide for Linux](#).

2. The Simple Forward application is based on DOCA Flow. Therefore, the user is required to allocate huge pages.

```
echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

On some operating systems (RockyLinux, OpenEuler, CentOS 8.2) the default huge page size on the DPU (and Arm hosts) is larger than 2MB, and is often 512MB instead. One can find out the size of the huge pages using the following command:

```
$ grep -i huge /proc/meminfo
```

```
AnonHugePages: 0 kB  
ShmemHugePages: 0 kB  
FileHugePages: 0 kB  
HugePages_Total: 4  
HugePages_Free: 4  
HugePages_Rsvd: 0  
HugePages_Surp: 0  
Hugepagesize: 524288 kB  
Hugetlb: 6291456 kB
```

Given that the guiding principal is to allocate 4GB of RAM, in such cases instead of allocating 2048 pages, one should allocate the matching amount (8 pages):

```
echo '8' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-524288kB/nr_hugepages
```

Application Execution

The simple forward application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_simple_forward_vnf [DPDK Flags] -- [DOCA Flags] [Program Flags]
```

DOCA Flags:

```
-h, --help Print a help synopsis  
-v, --version Print program version information  
-l, --log-level Set the (numeric) log level for the program <10=DISABLE,  
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>  
--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE,  
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>  
-j, --json <path> Parse all command flags from an input json file
```

Program Flags:

```
-t, --stats-timer <time> Set interval to dump stats information  
-q, --nr-queues <num> Set queues number  
-r, --rx-only Set rx only  
-o, --hw-offload Set PCI address of the RXP engine to use  
-hq, --hairpinq Set forwarding to hairpin queue  
-a, --age-thread Start thread do aging
```

Info

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_simple_fwd_vnf -- -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the BlueField:

```
./doca_simple_fwd_vnf -a auxiliary:mlx5_core.sf.4 -a auxiliary:mlx5_core.sf.5 -- -l 60
```

Note

SFs must be enabled according to the [NVIDIA BlueField DPU Scalable Function User Guide](#).

Before creating SFs on a specific physical port, it is important to verify the encap mode on the respective PF FDB. The default mode is basic. To check the encap mode, run:

```
cat /sys/class/net/p0/compat/devlink/encap
```

In this case, disable encap on the PF FDB before creating the SFs by running:

```
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode legacy
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode legacy
echo none > /sys/class/net/p0/compat/devlink/encap
echo none > /sys/class/net/p1/compat/devlink/encap
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode switchdev
/opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode switchdev
```


If the encap mode is set to basic then the application fails upon initialization.

Note

The flag `-a auxiliary:mlx5_core.sf.4 -a auxiliary:mlx5_core.sf.5` is mandatory for proper usage of the application.

1. Modifying this flag results unexpected behavior as only 2 ports are supported.
2. The SF number is arbitrary and configurable.

Note

The SF numbers must match the desired SF devices.

3. CLI example for running the application on the host:

```
./doca_simple_fwd_vnf -a 04:00.3 -a 04:00.4 -- -l 60
```

Note

The device identifiers must match the desired network devices.

Info

For more information, refer to section "Running DOCA Application on Host" in [NVIDIA DOCA Virtual Functions User Guide](#).

4. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_simple_fwd_vnf --json [json_file]
```

For example:

```
./doca_simple_fwd_vnf --json ./simple_fwd_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
DPDK Flags	a	devices	Add a PCIe device into the list of devices to probe.	<pre>"devices": [{"device": "sf", "id": "4", "sft": true},</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
				<pre>{ "device": "sf", "id": "5", "sft": true, }</pre>
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with TRACE log level support) 	<pre>"log-level": 60</pre>
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	<pre>"sdk-log-level": 40</pre>
	j	json	Parse all command flags from an input JSON file	N/A
Program flags	t	stats-timer	Set interval to dump stats information.	<pre>"stats-timer": 2</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	q	nr-queues	Set queues number.	<code>"nr-queues": 4</code>
	r	rx-only	Set RX only. When set, the packets will not be sent to the TX queues.	<code>"rx-only": false</code>
	o	hw-offload	Set HW offload of the RXP engine to use.	<code>"hw-offload": false</code>
	hq	hairpinq	Set forwarding to hairpin queue.	<code>"hairpinq": false</code>
	a	age-thread	Start a dedicated thread that handles the aged flows.	<code>"age-thread": false</code>

Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications .

Application Code Flow

1. Parse application argument.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register application parameters.

```
register_simple_fwd_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

1. Parse DPDK flags and invoke handler for calling the `rte_eal_init()` function.

2. Parse app parameters.

2. DPDK initialization.

```
dpdk_init();
```

Calls `rte_eal_init()` to initialize EAL resources with the provided EAL flags.

3. DPDK port initialization and start.

```
dpdk_queues_and_ports_init();
```

1. Initialize DPDK ports.
2. Create mbuf pool using `rte_pktmbuf_pool_create`.
3. Driver initialization – use `rte_eth_dev_configure` to configure the number of queues.
4. Rx/Tx queue initialization – use `rte_eth_rx_queue_setup` and `rte_eth_tx_queue_setup` to initialize the queues.

5. Rx hairpin queue initialization – use `rte_eth_rx_hairpin_queue_setup` to initialize the queues.

6. Start the port using `rte_eth_dev_start`.

4. Simple forward initialization.

```
simple_fwd_init();
```

1. `simple_fwd_create_ins` – create flow tables using `simple_fwd_ft_create`.

2. `simple_fwd_init_ports_and_pipes` – initialize DOCA port using `simple_fwd_init_doca_port` and build default pipes for each port.

5. Main loop.

```
simple_fwd_process_pkts();
```

1. Receive packets using `rte_eth_rx_burst` in a loop.

2. Process packets using `simple_fwd_process_offload`.

3. Transmit the packets on the other port by calling `rte_eth_tx_burst`. Or free the packet mbuf if `rx_only` is set to true.

6. Process packets.

```
simple_fwd_process_offload();
```

1. Parse the packet's `rte_mbuf` using `simple_fwd_pkt_info`.

2. Handle the packet using `simple_fwd_handle_packet`. If the packet's key does not match the existed the flow entry, create a new flow entry and PIPE using `simple_fwd_handle_new_flow`. Otherwise, increase the total packet's counter.

7. Simple forward destroy.

```
simple_fwd_destroy();
```

Simple forward close port and clean the flow resources.

8. DPDK ports and queues destruction.

```
dpdk_queues_and_ports_fini();
```

9. DPDK finish.

```
dpdk_fini();
```

Calls `rte_eal_destroy()` to destroy initialized EAL resources.

10. Arg parser destroy.

```
doca_argp_destroy();
```

- Free DPDK resources by call `rte_eal_cleanup()` function.

References

- `/opt/mellanox/doca/applications/simple_fwd_vnf/`
- `/opt/mellanox/doca/applications/simple_fwd_vnf/simple_fwd_params.json`

NVIDIA DOCA Switch Application Guide

This guide provides an example of switch implementation on top of NVIDIA® BlueField® DPU.

Introduction

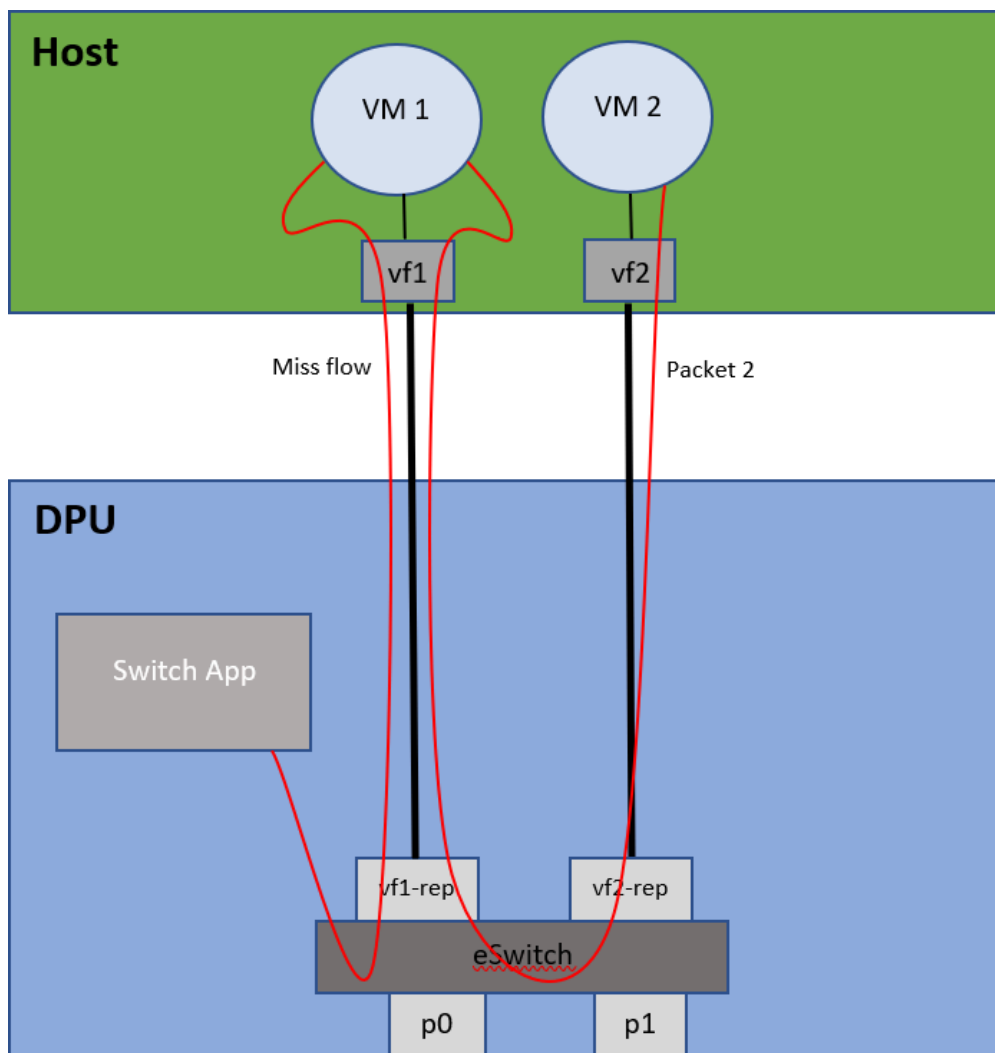
DOCA Switch is a network application that leverages the DPU's hardware capability for internal switching between representor ports on the DPU.

DOCA Switch is based on the [DOCA Flow](#) library. As such, it exposes a command line interface which receives DOCA Flow like commands to allow adding rules in real time.

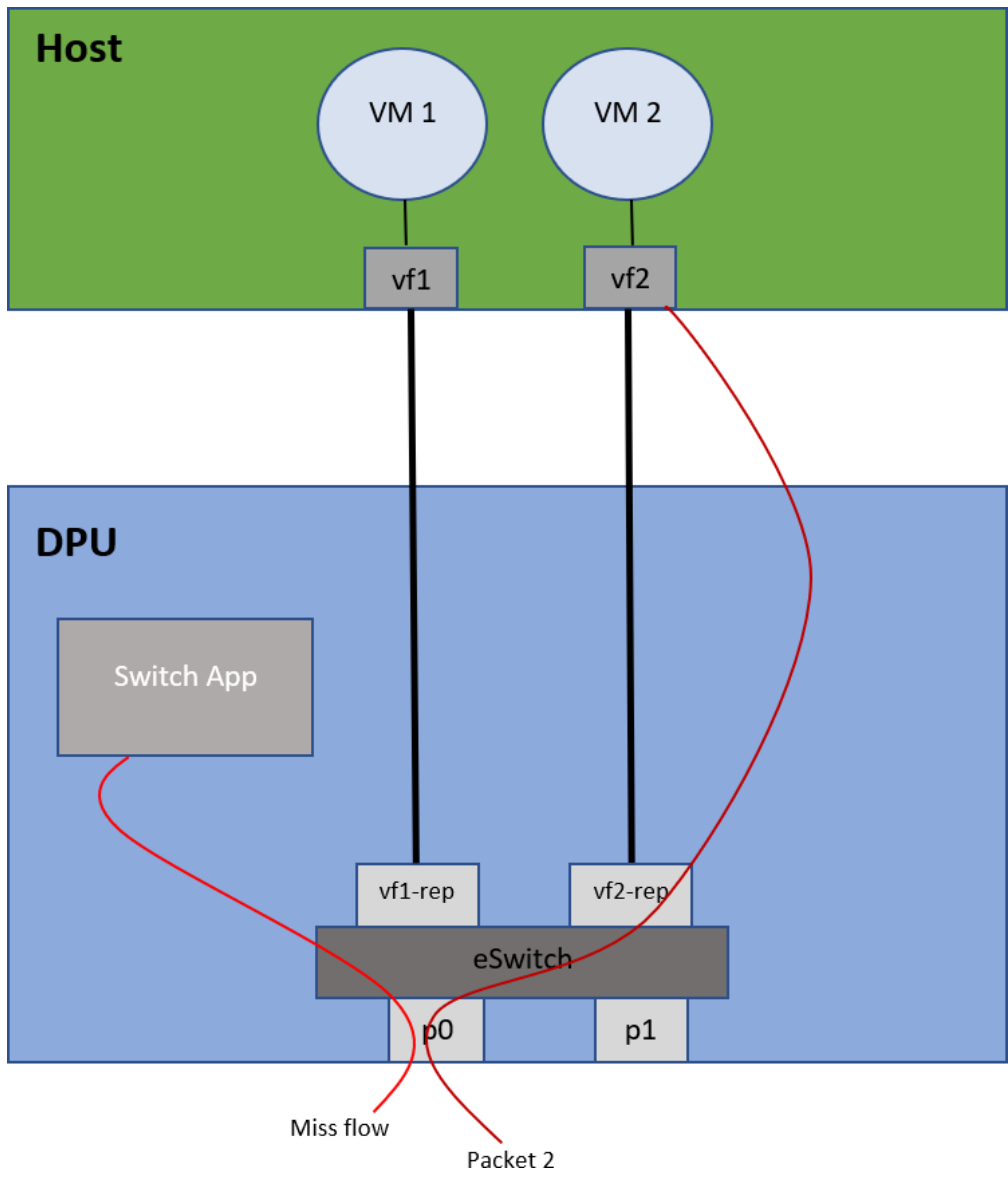
System Design

DOCA Switch is designed to run on the DPU as a standalone application (all network traffic goes directly through it).

Traffic flows between two VMs on the host:



Traffic flow from a physical port to a VM on the host:

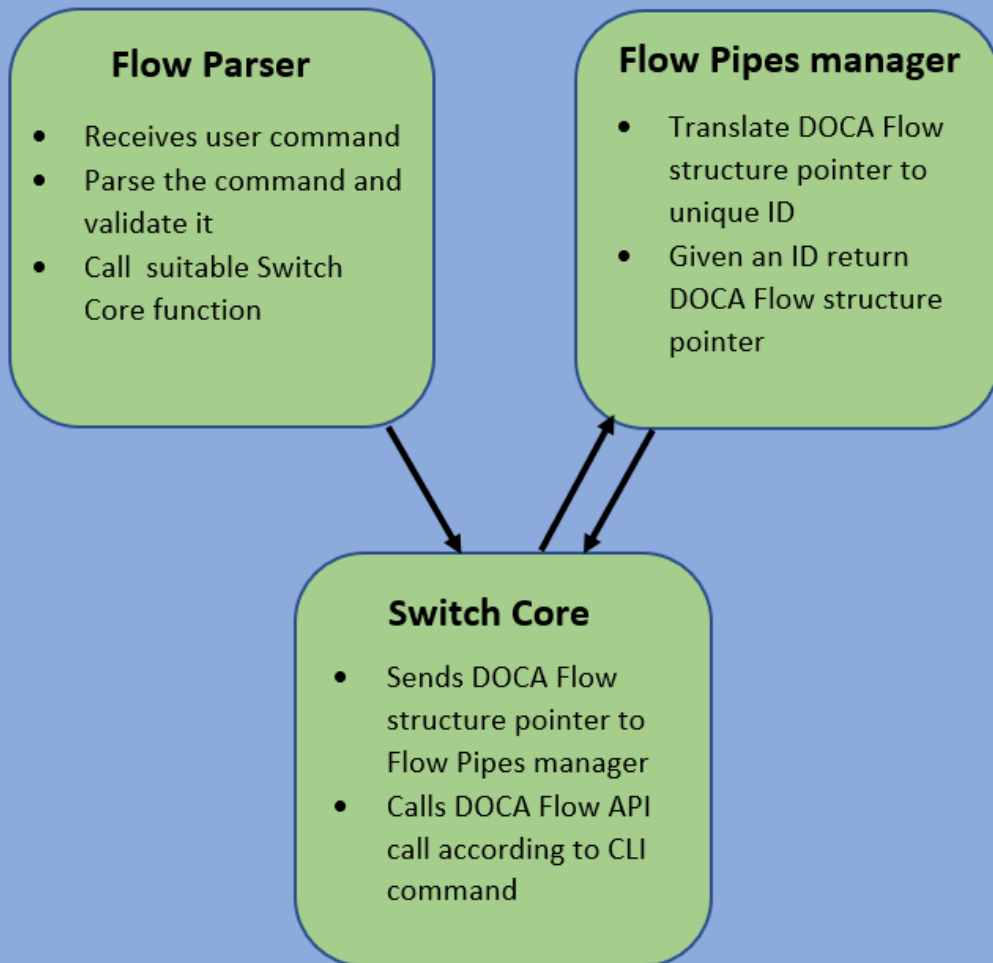


Application Architecture

DOCA Switch is based on 3 modules:

- Command line interface – receives pre-defined DOCA Flow-like commands and parses them
- Flow pipes manger – generates a unique identification number for each DOCA Flow structure created
- Switch core – combines all modules together and calls necessary DOCA Flow API

Switch



Port initialization cannot be made dynamically. All ports must be defined when running the application with standard DPDK flags.

- When adding a pipe or an entry, the user must run commands to create the relevant structs beforehand
- Optional parameters must be specified by the user in the command line; otherwise, NULL is used
- After a pipe or an entry is created successfully, the relevant ID is printed for future use

DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA Flow](#)

Refer to its respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory: `/opt/mellanox/doca/applications/switch/` .

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_switch is created under /tmp/build/switch/.

Recompiling Only the Current Application

To directly build only the switch application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_switch=true  
ninja -C /tmp/build
```

Info

doca_switch is created under /tmp/build/switch/.

Alternatively, one can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:
 - o Set enable_all_applications to false
 - o Set enable_switch to true

2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_switch is created under /tmp/build/switch/.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Prerequisites

The switch application is based on DOCA Flow. Therefore, the user is required to allocate huge pages.

```
echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-  
2048kB/nr_hugepages
```

Note

On some operating systems (RockyLinux, OpenEuler, CentOS 8.2) the default huge page size on the DPU (and Arm hosts) is larger than

2MB, and is often 512MB instead. Once can find out the size of the huge pages using the following command:

```
$ grep -i huge /proc/meminfo
```

```
AnonHugePages: 0 kB  
ShmemHugePages: 0 kB  
FileHugePages: 0 kB  
HugePages_Total: 4  
HugePages_Free: 4  
HugePages_Rsvd: 0  
HugePages_Surp: 0  
Hugepagesize: 524288 kB  
Hugetlb: 6291456 kB
```

Given that the guiding principal is to allocate 4GB of RAM, in such cases instead of allocating 2048 pages, one should allocate the matching amount (8 pages):

```
echo '8' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-  
524288kB/nr_hugepages
```

Application Execution

The switch application is provided in source form. Therefore, hence a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_switch [DPDK Flags] -- [DOCA Flags]
```

```
DOCA Flags:
```

```
-h, --help Print a help synopsis
-v, --version Print program version information
-l, --log-level Set the (numeric) log level for the program <10=DISABLE,
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE,
20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
-j, --json <path> Parse all command flags from an input json file
```

Info

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_switch -- -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the BlueField:

```
./doca_switch -a 03:00.0,representor=[0-2],dv_flow_en=2 -- -l 60
```

Note

dv_flow_en=2 is necessary to run the application with hardware steering.

Note

The PCIe address (03:00.0) should match the address of the desired PCIe device.

Command Line Flags

Flag Type	Short Flag	Long Flag	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none">• DISABLE=10• CRITICAL=20• ERROR=30• WARNING=40• INFO=50• DEBUG=60• TRACE=70 (requires compilation with TRACE log level support)	<pre>"log-level": 60</pre>
	N/A	sdk-log-level	Sets the log level for the program: <ul style="list-style-type: none">• DISABLE=10• CRITICAL=20• ERROR=30• WARNING=40• INFO=50• DEBUG=60• TRACE=70	<pre>"sdk-log- level": 40</pre>
	j	json	Parse all command flags from an input JSON file	N/A

Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Supported Commands

- create pipe port_id=[port_id][,<optional_parameters>] Available optional parameters:
 - name=<pipe-name>
 - root_enable=[1 | 0]
 - monitor=[1 | 0]
 - match_mask=[1 | 0]
 - fwd=[1 | 0]
 - fwd_miss=[1 | 0]
 - type=[basic | control]
- add entry pipe_id=<pipe_id>,pipe_queue=<pipe_queue>[,<optional_parameters>] Available optional parameters:
 - monitor=[1 | 0]
 - fwd=[1 | 0]
- add control_pipe entry priority=<priority>,pipe_id=<pipe_id>,pipe_queue=<pipe_queue>[,<optional_parameters>] Available optional parameters:
 - match_mask=[1 | 0]
 - fwd=[1 | 0]

- destroy pipe pipe_id=<pipe_id>
- rm entry pipe_queue=<pipe_queue>,entry_id=[entry_id]
- port pipes flush port_id=[port_id]
- port pipes dump port_id=[port_id],file=[file_name]
- query entry_id=[entry_id]
- create [struct] [field=value,...]
Struct options: pipe_match, entry_match, match_mask, actions, monitor, fwd, fwd_miss

- Match struct fields:

Fields	Field Options
flags	
port_meta	
outer.eth.src_mac	
outer.eth.dst_mac	
outer.eth.type	
outer.vlan_tci	
outer.l3_type	ipv4, ipv6
outer.src_ip_addr	
outer.dst_ip_addr	
outer.l4_type_ext	tcp, udp , gre
outer.tcp.flags	FIN, SYN, RST, PSH, ACK, URG, ECE, CWR
outer.tcp_src_port	
outer.tcp_dst_port	
outer.udp_src_port	
outer.udp_dst_port	
tun_type	
vxlan_tun_id	
gre_key	
gtp_teid	

Fields	Field Options
inner.eth.src_mac	
inner.eth.dst_mac	
inner.eth.type	
inner.vlan_tci	
inner.l3_type	ipv4, ipv6
inner.src_ip_addr	
inner.dst_ip_addr	
inner.l4_type_ext	tcp, udp
inner.tcp.flags	FIN, SYN, RST, PSH, ACK, URG, ECE, CWR
inner.tcp_src_port	
inner.tcp_dst_port	
inner.udp_src_port	
inner.udp_dst_port	

- o Actions struct fields:

Fields	Field Options
decap	true, false
mod_src_mac	
mod_dst_mac	
mod_src_ip_type	ipv4, ipv6
mod_src_ip_addr	
mod_dst_ip_type	ipv4, ipv6
mod_dst_ip_addr	
mod_src_port	
mod_dst_port	
ttl	
has_encap	true, false
encap_src_mac	

Fields	Field Options
encap_dst_mac	
encap_src_ip_type	ipv4, ipv6
encap_src_ip_addr	
encap_dst_ip_type	ipv4, ipv6
encap_dst_ip_addr	
encap_tup_type	vxlan, gtpu, gre
encap_vxlan-tun_id	
encap_gre_key	
encap_gtp_teid	

o FWD struct fields:

Fields	Field Options
type	rss, port, pipe, drop
rss_flags	
rss_queues	
num_of_queues	
port_id	
next_pipe_id	

o Monitor struct fields:

- flags
- cir
- cbs
- aging

The physical port number (only one physical port is supported) will always be 0 and all representor ports are numbered from 1 to N where N is the number of representors being used. For example:

- Physical port ID: 0
- VF0 representor port ID: 1
- VF1 representor port ID: 2
- VF2 representor port ID: 3

The following is an example of creating a pipe and adding one entry into it:

```
create fwd type=port,port_id=0xffff
create pipe port_id=0,name=p0_to_vf1,root_enable=1,fwd=1
create fwd type=port,port_id=1
add entry pipe_queue=0,fwd=1,pipe_id=1012
....
rm entry pipe_queue=0,entry_id=447
```

1. Pipe is configured on port ID 0 (physical port).
2. Entry is configured to forward all traffic directly into port ID 1 (VF0).
3. When the forwarding rule is no longer needed, the entry is deleted.
4. Ultimately, both entries are deleted, each according to the unique random ID it was given:

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.

Application Code Flow

1. Parse application argument.

1. Initialize the arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register application parameters.

```
register_switch_params();
```

3. Parse app parameters.

```
doca_argp_start();
```

2. Count total number of ports.

```
switch_ports_count();
```

1. Check how many ports are entered when running the application.

3. Initialize DPDK ports and queues.

```
dpdk_queues_and_ports_init();
```

4. Initialize DOCA Switch.

```
switch_init();
```

1. Initialize DOCA Flow.

2. Create port pairs.

3. Create Flow Pipes Manger module.

4. Register an action for each relevant CLI command.

5. Initialize Flow Parser.

```
flow_parser_init();
```

1. Reset all internal Flow Parser structures.
2. Start the command line interface.
3. Receive user commands, parse them, and call the required DOCA Flow API command.
4. Close the interactive shell once a "quit" command is entered.

6. Clean Flow Parser resources.

```
flow_parser_cleanup();
```

7. Destroy Switch resources.

```
switch_destroy();
```

1. Destroy Flow Pipes Manager resources.

8. Destroy DOCA Flow.

```
switch_destroy();
```

9. Destroy DPDK ports and queues.

```
dpdk_queues_and_ports_fini();
```

10. DPDK finish.

```
dpdk_fini();
```

1. Call `rte_eal_destroy()` to destroy initialized EAL resources.

11. Arg parser destroy.

```
doca_argp_destroy();
```

References

- `/opt/mellanox/doca/applications/switch/`

NVIDIA DOCA UROM RDMO Application Guide

This guide provides a DOCA Remote Direct Memory Operation implementation on top of NVIDIA® BlueField® DPU using Unified Communication X (UCX) .

Introduction

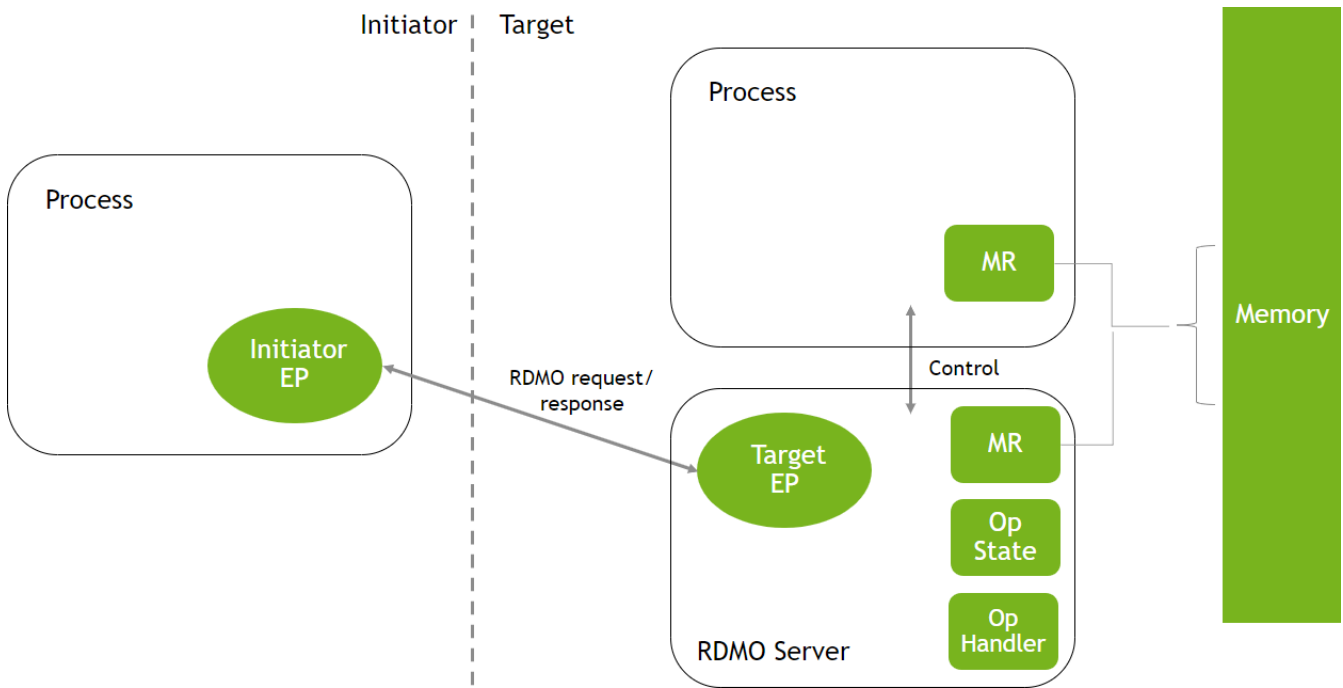
A remote direct memory operation (RDMO) is conceptionally an active message which is executed outside the context of the target process.

An RDMO involves the following entities:

- Target – establishes a connection to the server to use as the control path. The target interacts with the server to define target endpoints and memory regions. The target exchanges endpoint and memory region information with an initiator to facilitate its connection.
- Initiator – establishes a connection to the server to use as the data path. An RDMO is initiated by sending an RDMO command with an optional payload to the server. The server parses the commands and runs an associated RDMO handler. An RDMO handler interacts with the target process by performing one-sided memory accesses to target-defined memory regions.
- Server – responsible for executing RDMOs asynchronously from the target process. The server implements an RDMO handler for each supported operation. RDMO handlers may maintain a state within the server for optimization.

The DOCA UROM RDMO application includes the above three entities, split into the following parts:

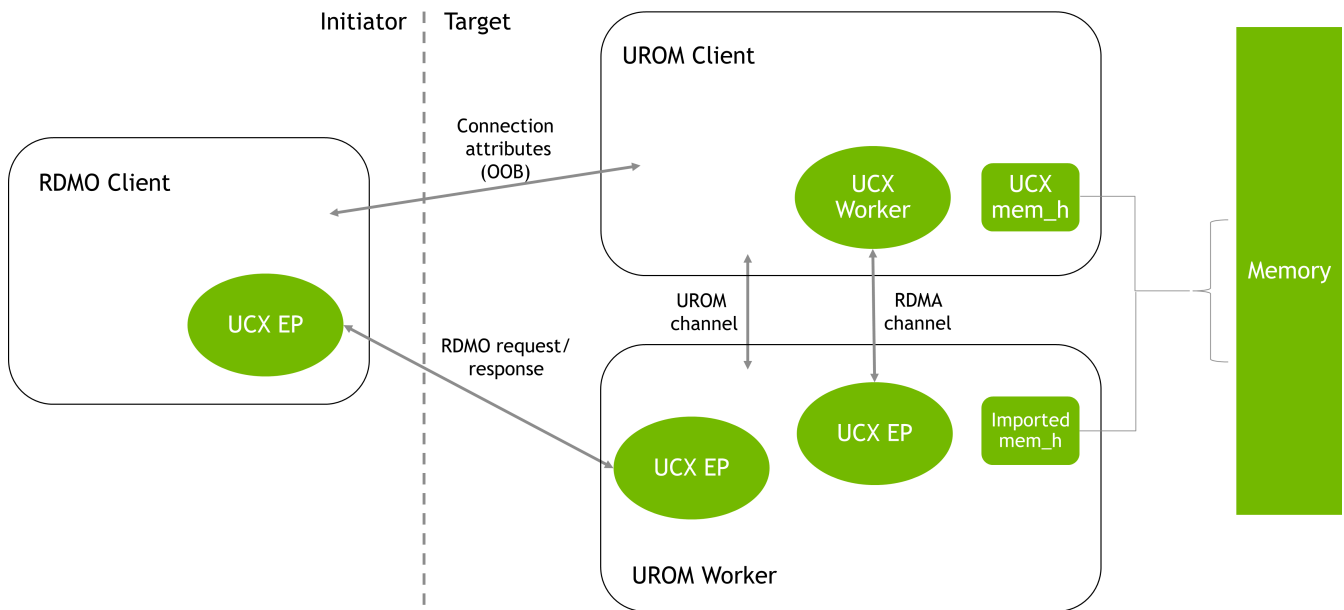
- BlueField side – the implementation of RDMO plugin component to be loaded by the DOCA UROM worker (which is the RDMO server)
- Host side – host application that runs using two modes: target and initiator



RDMOs are designed to take advantage of extra computing resources on a platform. While application processes run on the primary compute resources, an RDMO server can run on idle resources on the same host or be offloaded to run on a separate device (i.e., BlueField).

System Design

The application demonstrates the implementation of RDMO operations as a DOCA UROM worker plugin component. A target process would use the DOCA UROM API to create a worker with RDMO capabilities. An initiator process establishes an RDMO connection to the UROM worker. The plugin uses UCX as its transport.



Bootstrap Procedure

To connect the RDMO initiator and target, on the target side, UROM is used to retrieve an address for each created RDMO worker. This address would need to be delivered to the RDMO initiator side for connection establishment. The initiator address is obtained from the UCX worker created explicitly by the RDMO application. Both addresses are exchanged over the out-of-band (OOB) network and used to establish the connection:

- On the RDMO initiator side, a UCX endpoint is created using UCX API
- On the RDMO target side, the initiator's address is communicated to the RDMO worker using the UROM command channel

Memory Management

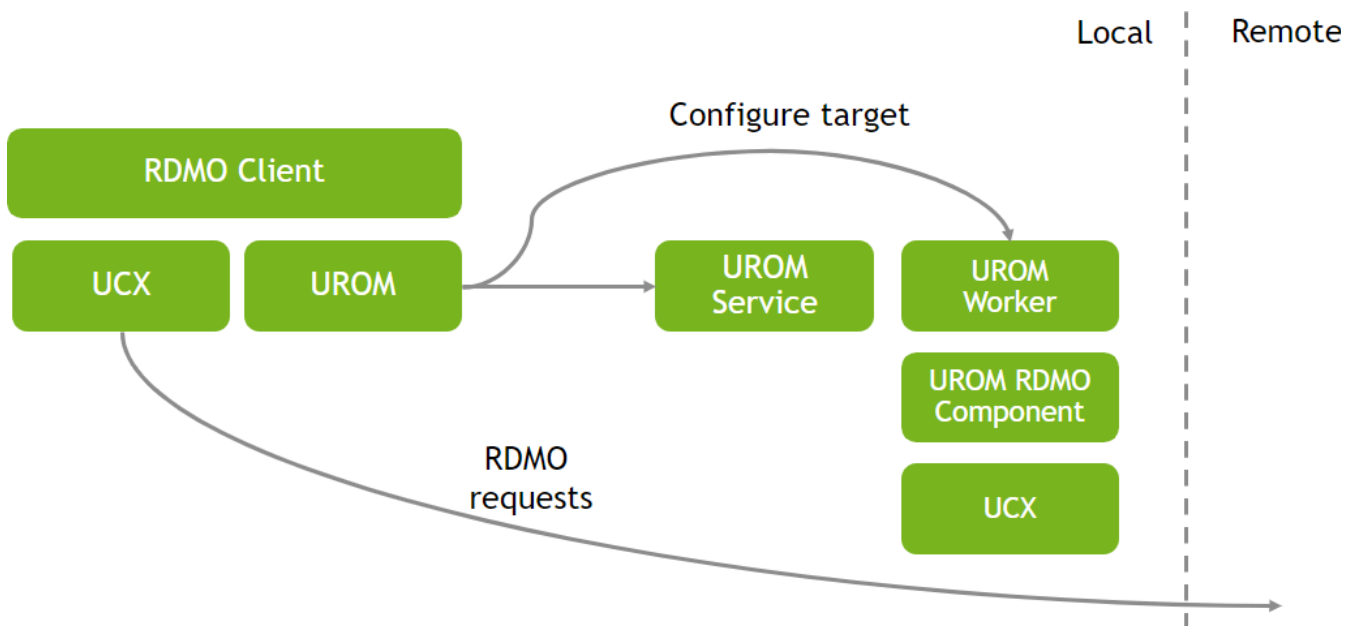
UROM returns an identifier (ID) for each memory region imported to the RDMO plugin component. This ID is used to refer to a target memory region in RDMO requests. It must be exchanged with the initiator process OOB.

RDMO UROM Worker Operation

Communication between the RDMO initiator and worker is implemented on top of UCX active messages. The worker's active message handler is the entry point that identifies the type of the RDMO operation based on the RDMO request header. The request is then forwarded to the corresponding RDMO operation handler which determines the operation parameters by inspecting the operation-specific sub-header in the request.

UCX active messages support eager and rendezvous protocols. When using a rendezvous protocol, the worker can choose whether to pull data to the server or move it directly to a target memory using a UCX-imported memory handle.

An RDMO operation handler may perform any combination of computation, initiator and target memory accesses, server state updates, or responses.



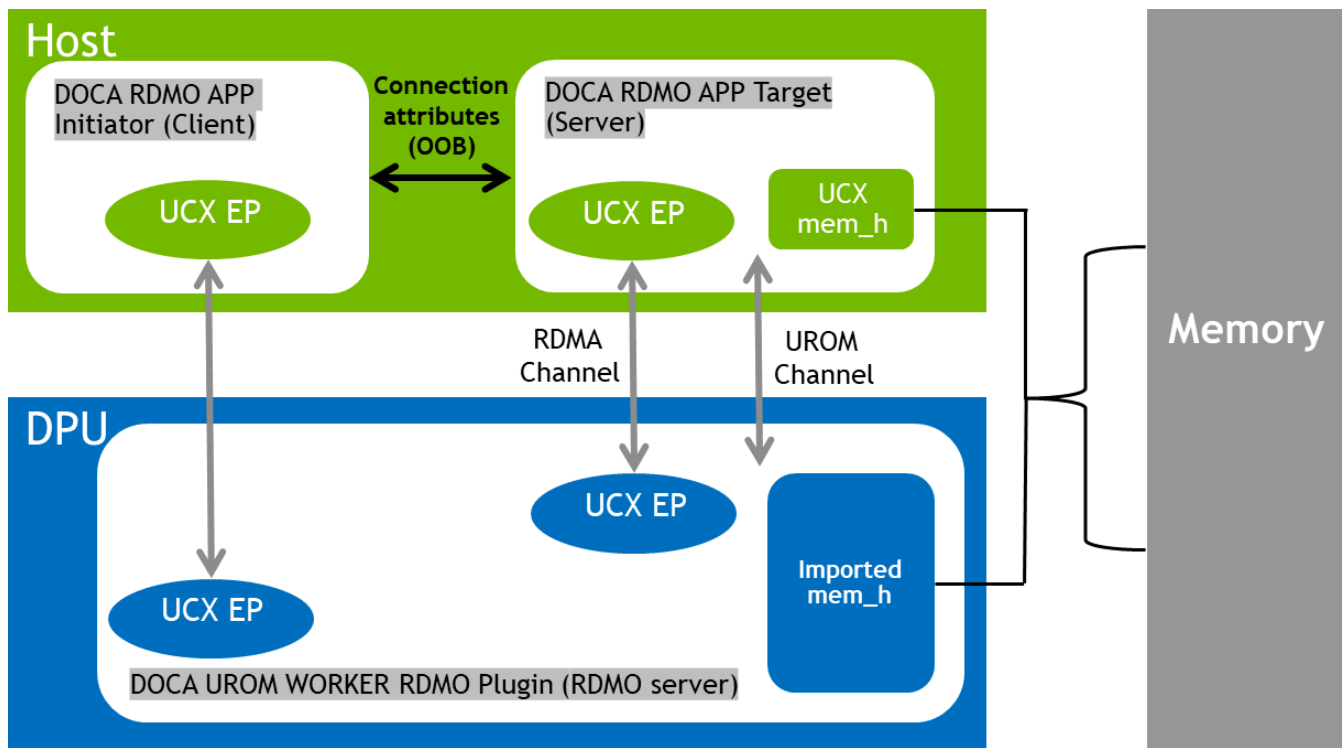
The RDMO client uses UROM to instantiate an RDMO worker and to configure target endpoints and memory regions. The client uses UCX directly to connect endpoints to the RDMO server. The client uses UCX to send formatted RDMO messages.

Application Architecture

DOCA's UROM RDMO application implementation uses UCX to support data exchange between endpoints. It utilizes UCX's sockaddr-based connection establishment and the UCX active messages (AM) API for communications, and UCX is responsible for all RDMO communications (control and data path).

The RDMO server application initiates a DOCA UROM worker RDMO component via the DOCA UROM service and shares the UROM worker UCX EP with the DOCA UROM RDMO client application. The RDMO server application imports memory regions into the UROM worker to facilitate RDMA operations from the BlueField on host memory.

The RDMO client application performs RDMO operations via the DOCA UROM worker. Upon receiving the UCX EP address from the server, the client application initially establishes a connection with the worker. It then proceeds to request the worker to execute the operation without the server application's awareness.



UROM RDMO Worker Component

The UROM RDMO worker plugin component defines a small set of commands to enable the target to:

- Establish a UCX communication channel between the client and the worker
- Create a UCX endpoint capable of receiving RDMO request
- Import memory regions that can be used as a source or target for RDMA initiated by the worker

The set of commands are:

```
enum urom_worker_rdmcmd_type {
    UROM_WORKER_CMD_RDMD_CLIENT_INIT,
    UROM_WORKER_CMD_RDMD_RQ_CREATE,
    UROM_WORKER_CMD_RDMD_RQ_DESTROY,
    UROM_WORKER_CMD_RDMD_MR_REG,
    UROM_WORKER_CMD_RDMD_MR_DEREG,
};
```

The associated notification types are:

```
enum urom_worker_rdmnotify_type {
    UROM_WORKER_NOTIFY_RDMD_CLIENT_INIT,
    UROM_WORKER_NOTIFY_RDMD_RQ_CREATE,
    UROM_WORKER_NOTIFY_RDMD_RQ_DESTROY,
    UROM_WORKER_NOTIFY_RDMD_MR_REG,
    UROM_WORKER_NOTIFY_RDMD_MR_DEREG,
};
```

Init

The C client Init command initializes the client to receive RDMOs. This includes establishing a connection between worker and host to allow the RDMO worker to access client memory.

The command is of type `UROM_WORKER_CMD_RDMD_CLIENT_INIT`. Command format:

```
struct urom_worker_rdmcmd_client_init {
    uint64_t id;
    void *addr;
    uint64_t addr_len;
};
```

- id – client ID used to identify the target process in RDMO commands
- addr – pointer to the client's UCP worker address to use for a worker-to-host connection

- `addr_len` – length of the address

This command returns a notification of type `UROM_WORKER_NOTIFY_RDMO_CLIENT_INIT` .
Notification format:

```
struct urom_worker_rdm_notify_client_init {  
    void *addr;  
    uint64_t addr_len;
```

- `addr` – pointer to the component's UCP worker address to use for initiator-to-server connections
- `addr_len` – length of the address

RQ Create

This Receive Queue (RQ) Create command creates and connects a new endpoint on the server. The endpoint may be targeted by formatted RDMO messages.

This command is of type `UROM_WORKER_CMD_RDMO_RQ_CREATE` . Command format:

```
struct urom_worker_rdm_cmd_rq_create {  
    void *addr;  
    uint64_t addr_len;  
};
```

- `addr` – the UCP worker address to use to connect the new endpoint
- `addr_len` – the length of address

The command returns a notification of type `UROM_WORKER_NOTIFY_RDMO_RQ_CREATE` . Notification format:

```
struct urom_worker_rdm_notify_rq_create {  
    uint64_t rq_id;
```

```
};
```

- `rq_id` – the RQ ID to use to destroy the RQ

RQ Destroy

The RQ Destroy command destroys an RQ.

The RQ Destroy command is of type `UROM_WORKER_CMD_RDMO_RQ_DESTROY`. Command format:

```
struct urom_worker_rdmcmd_rq_destroy {  
    uint64_t rq_id;  
};
```

- `rq_id` – the ID of a previously created RQ

The RQ destroy command returns a notification of type `UROM_WORKER_NOTIFY_RDMO_RQ_DESTROY`. Notification format:

```
struct urom_worker_rdmcmd_notify_rq_destroy {  
    uint64_t rq_id;  
};
```

- `rq_id` – the destroyed receive queue id

MR Register

The Memory Region (MR) Register command registers a UCP memory handle with the RDMO component. An MR must be registered with the RDMO component before use in RDMOs.

The command is of type `UROM_WORKER_CMD_RDMO_MR_REG`. Command format:

```

struct urom_worker_rdmcmd_mr_reg {
    uint64_t va;
    uint64_t len;
    void *packed_rkey;
    uint64_t packed_rkey_len;
    void *packed_memh;
    uint64_t packed_memh_len;
};

```

- va – the virtual address of the MR
- len – the length of the MR
- packed_rkey – pointer to the UCP packed R-key for the MR
- packed_rkey_len – the length of packed_rkey
- packed_mem_h – pointer to the UCP-packed memory handle for the MR. The memory handle must be packed with flag UCP_MEMH_PACK_FLAG_EXPORT.
- packed_memh_len – the length of packed_memh

The command returns a notification of type UROM_WORKER_NOTIFY_RDMDO_MR_REG .
Notification format:

```

struct urom_worker_rdmcmd_notify_mr_reg {
    uint64_t rkey;
};

```

- rkey – the ID used in RDMDOs to refer to the MR

MR Deregister

The MR deregister command deregisters an MR from the RDMDO component.

The command is of type UROM_WORKER_CMD_RDMDO_MR_DEREG . Command format:


```
struct urom_worker_rdmcmd_mr_dereg {
    uint64_t rkey;
};
```

- rkey – the ID of a previously registered MR

The command returns a notification of type `UROM_WORKER_NOTIFY_RDMO_MR_DEREG` .
Notification format:

```
struct urom_worker_rdmcmd_notify_mr_dereg {
    uint64_t rkey;
};
```

- rkey – the deregistered memory region remote key

Command Format

An RDMO is initiated by sending an RDMO request via UCP active message to a UROM RDMO worker server.

The RDMO request format is:

RDMO header

Op header

Payload (optional)

The RDMO header identifies the operation type and flags, modifying how the RDMO is processed. The operation (op) header includes arguments specific to the operation type. Optionally, the operation type may include an arbitrary-sized payload.

RDMO header format:

```
struct urom_rdmcmd_hdr {
    uint32_t id;
    uint32_t op_id;
};
```

```
uint32_t flags;  
};
```

- id – the client ID
- op_id – the RDMO operation type ID
- flags – flags modifying how the RDMO is processed by the server

Valid flag values:

```
enum urom_rdma_req_flags {  
    UROM_RDMA_REQ_FLAG_FENCE,  
};
```

- UROM_RDMA_REQ_FLAG_FENCE – Complete all outstanding RDMA requests on the connection before executing this request. This flag is required to implement a flush operation that guarantees remote completion.

Optionally, an operation may return a response to the initiator.

Response header format:

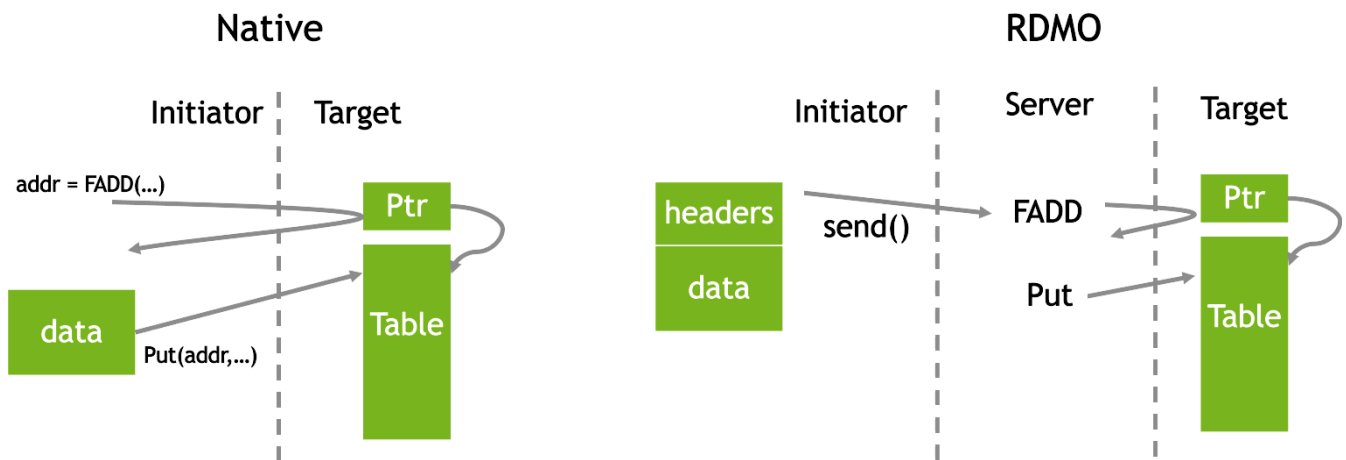
```
struct urom_rdma_rsp_hdr {  
    uint16_t op_id;  
};
```

- op_id – the RDMA response type ID

Append

RDMA Append atomically appends data to a queue in remote memory. This can be achieved in a one-sided programming model with a Fetching-Add operation to the location of a pointer in remote memory, followed by a Put to the fetched address. RDMA Append allows these dependent operations to be offloaded to the target.

The following diagram provides a comparison of native and RDMA approaches to the Append operation:



Combining two dependent operations into a single RDMO allows the non-blocking implementation of Append, as the initiator does not need to wait between the Fetching Atomic and the data write operations. Using RDMO, the initiator can create a pipeline of operations and achieve a higher message rate.

The rate at which the RDMO server can perform operations on the target memory is expected to be a bottleneck. To improve the rate, the following optimizations can be looked at:

- The result of the Fetch-and-ADD (FADD) after the initial Append is performed can be cached in the server. Subsequent Appends can re-use the cached value, eliminating the atomic FADD operation. The modified pointer value is required to be synchronized during the flush command.
- For small Append sizes, the Append data can be cached in the RDMO server and coalesced into a single Put. As a result, the server requires, on average, a single Put access to target memory to execute several RDMOs.
- To avoid extra memory usage and lost bandwidth for large Append operations, the RDMO server may initiate direct transfers from the initiator to the target memory bypassing the acceleration device memory.

The Append operation uses an operation of type `UROM_RDMO_OP_APPEND`. Append header format:

```
struct urom_rdmio_append_hdr {
    uint64_t ptr_addr;
    uint16_t ptr_rkey;
    uint16_t data_rkey;
```

```
};
```

- ptr_addr – the address of the queue pointer in target memory
- ptr_rkey – the R-key used to access ptr_addr
- data_rkey – the R-key used to access the queue data

The RDMO payload is the local data buffer.

Flush

RDMO Flush is used to implement synchronization between the initiator and server. On execution, Flush sends a response message back to the initiator. Flush can be used to guarantee remote completion of a previously issued RDMO.

To achieve this, the initiator sends an in-order Flush command including the RDMO flag UROM_RDMO_REQ_FLAG_FENCE. This flag causes the server to complete all previously received RDMOs before executing the Flush. To complete previous operations, the server must write any cached data and make it visible in the target memory. Once complete, the server executes the Flush. Flush sends a response to the initiator. When the initiator receives the flush message, the result of all previously sent RDMOs is guaranteed to be visible in the target memory.

The Flush operation uses operation type UROM_RDMO_OP_FLUSH. Flush header format:

```
struct urom_rdma_flush_hdr {  
    uint64_t flush_id;  
};
```

- flush_id – local ID used to track completion

Flush returns a response with the following header format:

```
struct urom_rdma_flush_rsp_hdr {  
    uint64_t flush_id;
```

```
};
```

- flush_id – the ID of the completed Flush

Flush requests and responses do not include a payload.

Scatter

RDMO Scatter is used to support aggregating non-contiguous memory Puts. A n RDMO may be defined to map non-contiguous virtual addresses into a single memory region using a network interface at the target platform, and then return a memory key for this region. The initiator may then perform Puts to this memory region, which are scattered by target hardware. Alternatively, an RDMO may be defined to post an IOV Receive. The initiator could then post a matching Send to scatter data at the target.

The Scatter operation uses operation type UROM_RDMO_OP_SCATTER. Scatter header format:

```
struct urom_rdmio_scatter_hdr {  
    uint64_t count; /* Number of IOVs in the payload */  
};
```

- count – Number of IOVs in the RDMO payload

IOVs are packed into the Scatter request payload, descriptor followed by data:

```
struct urom_rdmio_scatter_iov {  
    uint64_t addr; /* Scattered data address */  
    uint64_t rkey; /* Data remote key */  
    uint16_t len; /* Data length */  
};
```

- addr – scattered data address
- rkey – data remote key
- len – data length

DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA UROM](#)
- [UCX framework DOCA driver](#)

Refer to their respective programming guide for more information.

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/urom_rdma/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

On the host, `doca_urom_rdm0` is created under `/tmp/build/urom_rdm0/host/`.
On the BlueField side, the RDMO worker plugin `worker_rdm0.so` is created under `/tmp/build/urom_rdm0/dpu/`.

Compiling Only the Current Application

To directly build only the UROM RDMO application (host) or plugin (DPU):

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_urom_rdm0=true  
ninja -C /tmp/build
```

Info

On the host, `doca_urom_rdm` is created under `/tmp/build/urom_rdm/host/`.
On the BlueField side, the RDMO worker plugin `worker_rdm.so` is created under `/tmp/build/urom_rdm/dpu/`.

Alternatively, one can set the desired flags in the `meson_options.txt` file instead of providing them in the compilation command line:

1. Edit the following flags in `/opt/mellanox/doca/applications/meson_options.txt`:

- Set `enable_all_applications` to `false`
- Set `enable_urom_rdm` to `true`

2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

On the host, `doca_urom_rdm` is created under `/tmp/build/urom_rdm/host/`. On the BlueField side, the RDMO worker plugin `worker_rdm.so` is created under `/tmp/build/urom_rdm/dpu/`.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Host Application Execution

The UROM RDMO application is provided in source form; therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_urom_rdm0 [DOCA Flags] [Program Flags]
```

DOCA Flags:

-h, --help Print a help synopsis

-v, --version Print program version information

-l, --log-level Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

-j, --json <path> Parse all command flags from an input json file

Program Flags:

-d, --device <IB device name> IB device name.

-s, --server-name <server name> server name.

-m, --mode {server, client} Set mode type {server, client}

Info

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_urom_rdm0 -h
```

Info

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application with server mode:

```
./doca_urom_rdm0 -d mlx5_0 -m server
```

3. CLI example for running the application with client mode:

```
./doca_urom_rdm0 -m clinet -s <server_host_name>
```

4. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_urom_rdm0 --json [json_file]
```

For example:

```
./doca_urom_rdm0 --json ./urom_rdm0_params.json
```

RDMO DPU Plugin Component

The UROM RDMO plugin component is provided in source form, hence a compilation is required before the application can be executed in order when spawning UROM worker could load the plugin in runtime and it is compiled as .so file.

The plugin exposes the following symbols:

- Get DOCA worker plugin interface for RDMO plugin:

```
doca_error_t urom_plugin_get_iface(struct urom_plugin_iface *iface);
```

- Get the RDMO plugin version which will be used to verify that the host and DPU plugin versions are compatible:

```
doca_error_t urom_plugin_get_version(uint64_t *version);
```

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Print a help synopsis	N/A
	v	version	Print program version information	N/A
	l	log-level	Set the log level for the application: <ul style="list-style-type: none">• DISABLE=10• CRITICAL=20• ERROR=30• WARNING=40• INFO=50• DEBUG=60• TRACE=70 (requires compilation with TRACE log level support)	<pre>"log-level": 60</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	N/A	sdk-log-level	Set the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	<pre>"sdk-log-level": 40</pre>
	j	json	Parse all command flags from an input JSON file	N/A
Program flags	d	device	DOCA UROM IB device name	<pre>"device": "mlx5_0"</pre>
	s	server-name	RDMO server name	<pre>"server-name": "<host-name>-oob"</pre>
	m	mode	RDMO application mode [server, client]	<pre>"mode": "client"</pre>

Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.

Application Code Flow

1. Parse application argument.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register UROM RDMO application parameters.

```
register_urom_rdmoparams();
```

3. Parse the arguments.

```
doca_argp_start();
```

2. Run main logic:

o If the application mode is server:

1. Create UROM objects and spawn UROM worker on the BlueField.

2. Initialize UCP with features: UCP_FEATURE_AM, UCP_FEATURE_EXPORTED_MEMH.

3. Create a UCP worker and query the worker address

4. Initialize the RDMO worker client with the command
UROM_WORKER_CMD_RDMO_CLIENT_INIT.

5. Send UROM RDMO worker address to the initiator via OOB channel and
receive the initiator's UCP worker address

6. Create a UCP memory handle and register it with the RDMO server using
the command UROM_WORKER_CMD_RDMO_MR_REG. Receive an R-key in
return.

7. Send the RDMO key to the initiator
 8. Create an RDMO RQ by passing the initiator's UCP worker address to the UROM command `UROM_WORKER_CMD_RDMO_RQ_CREATE`.
 9. Wait till the RDMO append operation is done and next validate the memory data.
 10. Wait till the RDMO scatter operation is done and next validate the memory data.
 11. Destroy the UCP resources.
 12. Destroy UROM RDMO worker and UROM objects.
- If the application mode is client:
 1. Create UCP worker using UCX API directly.
 2. Receive the UROM RDMO worker address via OOB channel and send the initiator's UCP worker address.
 3. Create a UCP endpoint using the RDMO worker address.
 4. Install an Active Message handler on the endpoint to receive RDMO responses.
 5. Send an RDMO requests via UCP Active Message protocol with the header pointing to the serialized RDMO and Op headers, and data pointing to the payload. The request parameter flag: `UCP_AM_SEND_FLAG_REPLY` will be set to allow the RDMO server to identify the sender.
 6. Once the RDMO operations are done, Destroy UCP resources.
3. Arg parser destroy.

```
doca_argp_destroy();
```

References

- /opt/mellanox/doca/applications/urom_rdm0/
- /opt/mellanox/doca/applications/urom_rdm0/urom_rdm0_params.json

NVIDIA DOCA YARA Inspection Application Guide

This guide provides YARA inspection implementation on top of NVIDIA® BlueField® DPU.

Introduction

YARA inspection monitors all processes in the host system for specific YARA rules using the [DOCA App Shield](#) library.

This security capability helps identify malware detection patterns in host processes from an independent and trusted DPU. This is an innovative Intrusion Detection System (IDS) as it is designed to run independently on the DPU's Arm cores without hindering the host.

This DOCA App Shield based application provides the capability to read, analyze, and authenticate the host (bare metal/VM) memory directly from the DPU.

Using the library, this application scans host processes and looks for pre-defined YARA rules. After every scan iteration, the application indicates if any of the rules matched. Once there is a match, the application reports which rules were detected in which process. The reports are both printed to the console and exported to the [DOCA Telemetry Service](#) (DTS) using inter-process communication (IPC).

This guide describes how to build YARA inspection using the DOCA App Shield library which leverages DPU abilities such as hardware-based DMA, integrity, and more.

Note

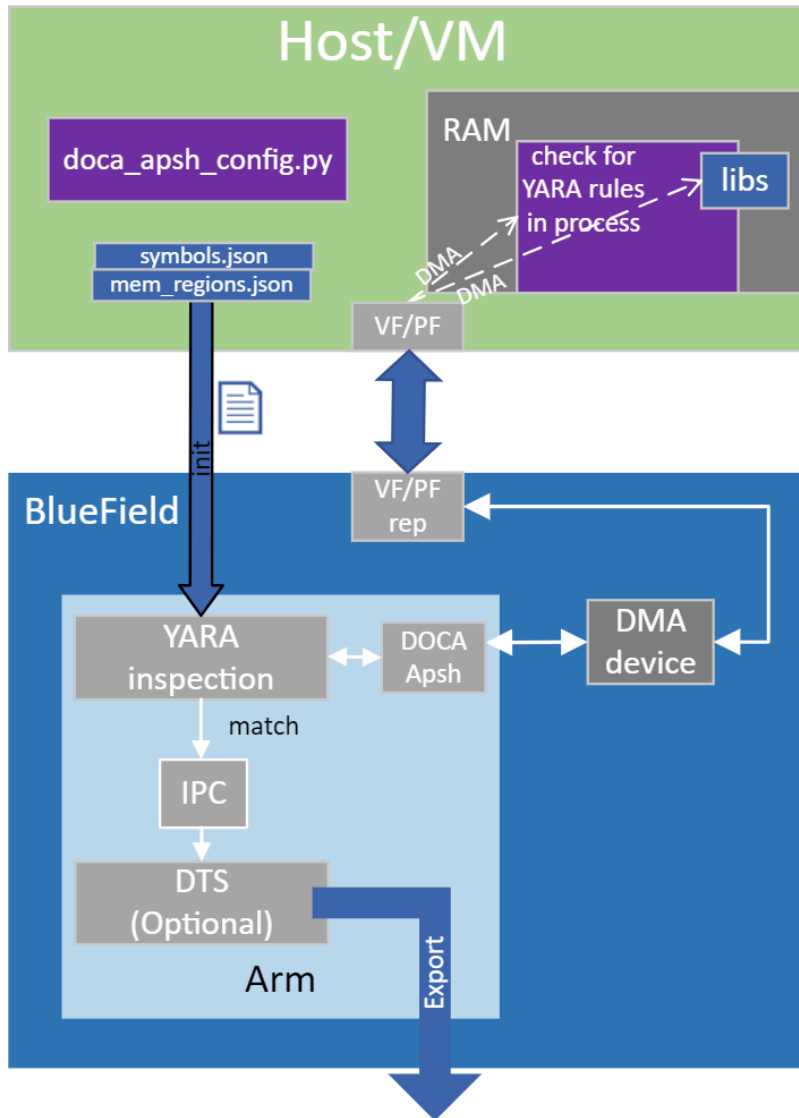
As the DOCA App Shield library only supports the YARA API for Windows hosts, this application can only be used to inspect Windows hosts.

System Design

The host's involvement is limited to generating the required ZIP and JSON files to pass to the DPU. This is done before the app is triggered, when the host is still in a "safe" state.

Generating the needed files can be done by running DOCA App Shield's `doca_apsh_config.py` tool on the host. See [DOCA App Shield](#) for more info.

-- → DMA read

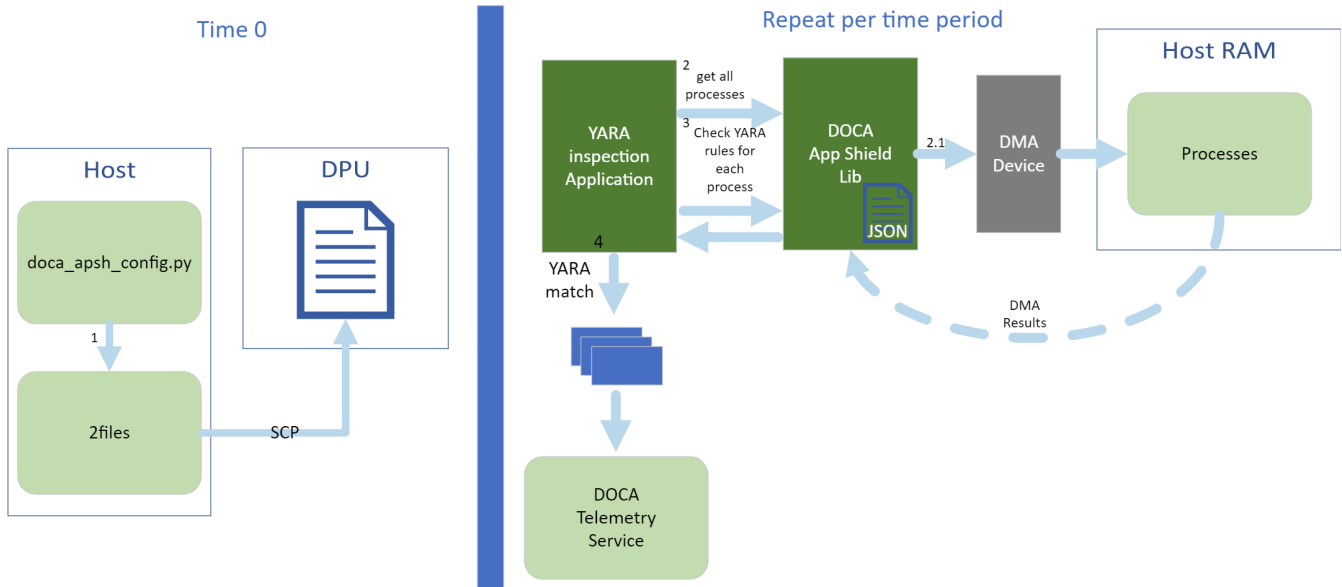


Application Architecture

The user creates the ZIP and JSON files using the DOCA tool `doca_apsh_config.py` and copies them to the DPU.

The application can report YARA rules detection to the:

- File
- Terminal
- DTS



1. The files are generated by running `doca_apsh_config.py` on the host against the process at time zero.

2. The following steps recur at regular time intervals:

1. The YARA inspection app requests a list of all apps from the DOCA App Shield library.
2. The app loops over all processes and checks for YARA rules match using the DOCA App Shield library.
3. If YARA rules are found (1 or more), the YARA attestation app reports results with a timestamp and details about the process and rules to:
 - Local telemetry files – a folder and files representing the data a real DTS would have received

Note

These files are used for the purpose of this example only as normally this data is not exported into user-readable files.

- DOCA log
- DTS IPC interface (even if no DTS is active)

3. The App Shield agent exits on first YARA rule detection.

DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA App Shield](#)
- [DOCA Telemetry](#)

Refer to their respective programming guide for more information.

Limitations

- The application is only available on Ubuntu 22.04 environments
- The application only supports the inspection of Windows hosts

Compiling the Application

Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:
`/opt/mellanox/doca/applications/yara_inspection/`.

Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

`doca_yara_inspection` is created under `/tmp/build/yara_inspection/`.

Compiling Only the Current Application

To directly build only the YARA inspection application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -Denable_yara_inspection=true  
ninja -C /tmp/build
```

Info

doca_yara_inspection is created under /tmp/build/yara_inspection/.

Alternatively, one can set the desired flags in the meson_options.txt file instead of providing them in the compilation command line:

1. Edit the following flags in /opt/mellanox/doca/applications/meson_options.txt:

- Set enable_all_applications to false
- Set enable_yara_inspection to true

2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

Info

doca_yara_inspection is created under /tmp/build/yara_inspection/.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

Running the Application

Prerequisites

1. Configure the BlueField's firmware

1. On the BlueField system, configure the PF base address register and NVME emulation. Run:

```
dpu> mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_SIZE=2  
PF_BAR2_ENABLE=1 NVME_EMULATION_ENABLE=1
```

2. Perform a [BlueField system reboot](#) for the mlxconfig settings to take effect.
3. This configuration can be verified using the following command:

```
dpu> mlxconfig -d /dev/mst/mt41686_pciconf0 q | grep -E "NVME|BAR"
```

2. Download target system (host/VM) symbols.

- o For Ubuntu:

```
host> sudo tee /etc/apt/sources.list.d/ddebs.list << EOF  
deb http://ddebs.ubuntu.com/ $(lsb_release -cs) main restricted universe multiverse  
deb http://ddebs.ubuntu.com/ $(lsb_release -cs)-updates main restricted universe  
multiverse  
deb http://ddebs.ubuntu.com/ $(lsb_release -cs)-proposed main restricted universe  
multiverse  
EOF  
host> sudo apt install ubuntu-dbgsym-keyring
```

```
host> sudo apt-get update
host> sudo apt-get install linux-image-$(uname -r)-dbg
```

- o For CentOS:

```
host> yum install --enablerepo=base-debuginfo kernel-devel-$(uname -r)
kernel-debuginfo-$(uname -r) kernel-debuginfo-common-$(uname -
m)-$(uname -r)
```

- o No action is needed for Windows

3. Perform IOMMU passthrough. This stage is only needed on some of the cases where IOMMU is not enabled by default (e.g., when the host is using an AMD CPU).

Note

Skip this step if you are not sure whether you need it. Return to it only if DMA fails with a message in `dmesg` similar to the following:

```
host> dmesg
[ 3839.822897] mlx5_core 0000:81:00.0: AMD-Vi: Event logged
[IO_PAGE_FAULT domain=0x0047 address=0x2a0aff8
flags=0x0000]
```

- o Locate your OS's grub file (most likely `/boot/grub/grub.conf`, `/boot/grub2/grub.cfg`, or `/etc/default/grub`) and open it for editing. Run:

```
host> vim /etc/default/grub
```

- o Search for the line defining `GRUB_CMDLINE_LINUX_DEFAULT` and add the argument `iommu=pt`. For example:

```
GRUB_CMDLINE_LINUX_DEFAULT="iommu=pt <intel/amd>_iommu=on"
```

- o Run:

Note

Prior to performing a power cycle, make sure to do a graceful shutdown.

- For Ubuntu:

```
host> sudo update-grub  
host> ipmitool power cycle
```

- For CentOS:

```
host> grub2-mkconfig -o /boot/grub2/grub.cfg  
host> ipmitool power cycle
```

- For Windows targets: Turn off Hyper-V capability.

4. The DOCA App Shield library uses hugepages for DMA buffers. Therefore, the user must allocate 42 huge pages.

1. Run:

```
dpu> nr_huge=$(cat  
/sys/devices/system/node/node0/hugepages/hugepages-  
2048kB/nr_hugepages)  
nr_huge=$((42+$nr_huge))  
echo $nr_huge | sudo tee -a  
/sys/devices/system/node/node0/hugepages/hugepages-
```

2048kB/nr_hugepages

2. Create the ZIP and JSON files. Run:

```
target-system> cd /opt/mellanox/doca/tools/  
target-system> python3 doca_apsh_config.py <pid-of-process-to-  
monitor> --os <windows/linux> --path <path to dwaf2json executable or  
pdbparse-to-json.py>  
target-system> cp /opt/mellanox/doca/tools/*.* <shared-folder-with-  
baremetal>  
dpu> scp <shared-folder-with-baremetal>/* <path-to-app-shield-binary>
```

If the target system does not have DOCA installed, the script can be copied from the BlueField.

The required dwaf2json and pdbparse-to-json.py are not provided with DOCA.

Note

If the kernel and process .exe have not changed, there no need to redo this step.

Application Execution

The YARA inspection application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:


```
Usage: doca_yara_inspection [DOCA Flags] [Program Flags]
```

DOCA Flags:

-h, --help Print a help synopsis
-v, --version Print program version information
-l, --log-level Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
-j, --json <path> Parse all command flags from an input json file


Program Flags:

-m, --memr <path> System memory regions map
-f, --vuid VUID of the System device
-d, --dma DMA device name
-o, --osym <path> System OS symbol map path
-t, --time <seconds> Scan time interval in seconds

 **Info**

This usage printout can be printed to the command line using the -h (or --help) options:

```
./doca_yara_inspection -h
```

 **Info**

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the BlueField:

```
./doca_yara_inspection -m mem_regions.json -o symbols.json -f
```

Note

All used identifiers (-f and -d flags) should match the identifier of the desired devices.

Command Line Flags

Flag Type	Short Flag	Long Flag	Description
General flags	h	help	Prints a help synopsis
	v	version	Prints program version information
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with TRACE log level support)

Flag Type	Short Flag	Long Flag	Description
	N/A	sdk-log-level	<p>Sets the log level for the program:</p> <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70
	j	json	Parse all command flags from an input JSON file
Program flags	m	memr	Path to the pre-generated mem_regions.json file transferred from the host
	f	pcif	<p>System PCIe function vendor unique identifier (VUID) of the VF/PF exposed to the target system. Used for DMA operations. To obtain this argument, run:</p> <pre>target-system> lspci -vv grep "[VU\] Vendor specific:"</pre> <p>Example output:</p> <pre>[VU] Vendor specific: MT2125X03335MLNXS0D0F0 [VU] Vendor specific: MT2125X03335MLNXS0D0F1</pre> <p>Two VUIDs are printed for each DPU connected to the target system. The first is of the DPU on pf0 and the second is of the DPU on port pf1.</p> <div style="background-color: #ffffcc; padding: 10px; margin-top: 10px;"> <p>Note</p> <p>Running this command on the DPU outputs VUIDs with an additional "EC" string in the middle. You must remove the "EC" to arrive at the correct VUID.</p> </div>

Flag Type	Short Flag	Long Flag	Description
			<p>The VUID of a VF allocated on PF0/1 is the VUID of the PF with an additional suffix, VF<vf-number>, where vf-number is the VF index +1. For example, for the output in the example above:</p> <ul style="list-style-type: none"> • PF0 VUID = MT2125X03335MLNXS0D0F0 • PF1 VUID = MT2125X03335MLNXS0D0F1 • VUID of VF0 on PF0 = MT2125X03335MLNXS0D0F0VF1 <p>VUIDs are persistent even on reset.</p>
	d	dma	DMA device name to use
	o	osym	Path to the pre-generated symbols.json file transferred from the host
	t	time	Number of seconds to sleep between scans

Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications .

Application Code Flow

1. Parse application argument.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register application parameters.

```
register_apsh_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

2. Initialize DOCA App Shield lib context.

1. Create lib context.

```
doca_apsh_create();
```

2. Set DMA device for lib.

```
open_doca_device_with_ibdev_name();  
doca_apsh_dma_dev_set();
```

3. Start the context

```
doca_apsh_start();  
apsh_system_init();
```

3. Initialize DOCA App Shield lib system context handler.

1. Get the representor of the remote PCIe function exposed to the system.

```
open_doca_device_rep_with_vuid();
```

2. Create and start the system context handler.

```
doca_apsh_system_create();  
doca_apsh_sys_os_symbol_map_set();  
doca_apsh_sys_mem_region_set();  
doca_apsh_sys_dev_set();  
doca_apsh_sys_os_type_set();  
doca_apsh_system_start();
```

4. Telemetry initialization.

```
telemetry_start();
```

1. Initialize a new telemetry schema.
 2. Register YARA type event.
 3. Set up output to file (in addition to default IPC).
 4. Start the telemetry schema.
 5. Initialize and start a new DTS source with the `gethostname()` name as source ID.
5. Loop until YARA rule is matched.

1. Get all processes from the host.

```
doca_apsh_processes_get();
```

2. Check for YARA rule identification and send a DTS event if there is a match.

```
doca_apsh_yara_get();  
if (yara_matches_size != 0) {  
    /* event fill logic
```

```
doca_telemetry_source_report();  
DOCA_LOG_INFO();  
sleep();
```

6. Telemetry destroy.

```
telemetry_destroy();
```

7. YARA inspection clean-up.

```
doca_apsh_system_destroy();  
doca_apsh_destroy();  
doca_dev_close();  
doca_dev_rep_close();
```

8. Arg parser destroy.

```
doca_argp_destroy();
```

References

- [/opt/mellanox/doca/applications/yara_inspection/](#)

DOCA Tools

This is an overview of the set of tools provided by DOCA and their purpose.

Introduction

DOCA tools are a set of executables/scripts that are needed to produce inputs to some of the DOCA libraries and applications.

All tools are installed with DOCA, as part of the `doca-tools` package, and can either be directly accessed from the terminal or can be found at `/opt/mellanox/doca/tools`. Refer to [NVIDIA DOCA Installation Guide for Linux](#) for more information.

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

Tools

DOCA Bench

CLI name: `doca_bench`

DOCA Bench is a tool that allows a user to evaluate the performance of DOCA applications, with reasonable accuracy for real-world applications. It provides a flexible architecture to evaluate multiple features in series with multi-core scaling to provide detailed throughput and latency analysis.

Capabilities Print Tool

CLI name: doca_caps

The caps tool is used to print the available devices and their representor devices (in the DPU), all their capabilities, and the available DOCA libraries.

DPA Tools

DOCA DPA tools are a set of executables that enable the DPA application developer and the system administrator to manage and monitor DPA resources and to debug DPA applications.

PCC Counter

CLI name: pcc_counters.sh

The PCC Counter tool is used to print PCC-related hardware counters. The output counters help debug the PCC user algorithm embedded in the DOCA PCC application.

Socket Relay

CLI name: doca_socket_relay

DOCA Socket Relay allows Unix Domain Socket (AF_UNIX family) server applications to be offloaded to Bluefield while communication between the two sides is proxied by DOCA Comm Channel.

NVIDIA DOCA Bench

Introduction

NVIDIA DOCA Bench allows users to evaluate the performance of DOCA applications, with reasonable accuracy for real-world applications. It provides a flexible architecture to evaluate multiple features in series with multi-core scaling to provide detailed throughput and latency analysis. e. The output and intermediate buffers is sized

This tool can be used to evaluate the performance of multiple DOCA operations, gain insight into each stage in complex DOCA operations and understand how items such as buffer sizing, scaling, and GGA configuration affect throughput and latency.

Feature Overview

DOCA Bench is designed as a unified testing tool for all BlueField accelerators. It, therefore, provides these major features:

- BlueField execution, utilizing the Arm cores and GGAs "locally"
- Host (x86) execution, utilizing x86 cores and the GGAs on the BlueField over PCIe
- Support for following DOCA/DPU features:
 - DOCA AES GCM
 - DOCA Comch
 - DOCA Compress
 - DOCA DMA
 - DOCA EC
 - DOCA Eth
 - DOCA RDMA
 - DOCA SHA
- Multi-core/multi-thread support
- Schedule executions based on time, job counts, etc.
- Ability to construct complex pipelines with multiple GGAs (where data moves serially through the pipeline)

- Various data sources (random data, file data, groups of files, etc.)
- Remote memory operations
 - Use data location on the host x86 platform as input to GGAs
- Comprehensive output to screen or CSV
- Query function to report supported software and hardware feature
- Sweeping of parameters between a start and end value, using a specific increment each time
- Specific attributes can be set per GGA instance, allowing fine control of GGA operation

Installation

DOCA Bench is installed and available in both DOCA-for-Host and DOCA BlueField Arm packages. It is located under the `/opt/mellanox/doca/tools` folder.

Prerequisites

DOCA 2.7.0 and higher.

Operating Modes

DOCA Bench measures performance of either throughput (bandwidth) or latency.

Throughput Measurements

In this mode, DOCA Bench measures the maximum performance of a given pipeline (see "[Core Principles](#)"). At the end of the execution, a short summary along with more detailed statistics is presented:

Aggregate stats

Duration: 3000049 micro seconds

Enqueued jobs: 17135128
Dequeued jobs: 17135128
Throughput: 005.712 MOperations/s
Ingress rate: 063.832 Gib/s
Egress rate: 063.832 Gib/s

Latency Measurements

Latency is the measurement of time taken to perform a particular operation. In this instance, DOCA Bench measures the time taken between submitting a job and receiving a response.

DOCA Bench provides two different types of latency measurement figures:

- Bulk latency mode – attempts to submit a group of jobs in parallel to gain maximum throughput, while reporting latency as the time between the first job submitted in the group and the last job received.
- Precision latency mode – used to ensure that only one job is submitted and measured before the next job is scheduled.

Bulk Latency

This latency mode effectively runs the pipelines at full rate, trying to maintain the maximum throughput of any pipeline while also recording latency figures for jobs submitted.

To record latency, while operating at the pipelines maximum throughput, users must place the latency figures inside groups or "buckets" (rather than record each individual job latency). Using this method, users can avoid the large memory and CPU overheads associated with recording millions of latency figures per second (which would otherwise significantly reduce the performance).

As each pipeline operation is different, and therefore has different latency characteristics, the user can supply the boundaries of the latency measure. DOCA Bench internally creates 100 buckets, of which the user can specify the starting value and the width or size of each bucket. The first and last bucket have significance:

- The first bucket contains all jobs that executed faster than the starting period


```
[32000ns -> 32999ns]: 0  
[33000ns -> 33999ns]: 128  
[34000ns -> 34999ns]: 0  
[35000ns -> 35999ns]: 0
```

Precision Latency

This latency mode operates on a single job at a time. At the cost of greatly reduced throughput, this allows the minimum latency to be precisely recorded. As shown below, the statistics generated are precise and include various fields such as min, max, median, and percentile values.

Aggregate stats

```
....  
min:      1878 ns  
max:      4956 ns  
median:   2134 ns  
mean:     2145 ns  
90th %ile: 2243 ns  
95th %ile: 2285 ns  
99th %ile: 2465 ns  
99.9th %ile: 3193 ns  
99.99th %ile: 4487 ns
```

Core Principles

The following subsections elaborate on principles which are essential to understand how DOCA Bench operates.

Host or BlueField Arm Execution

Whether executing DOCA Bench on an x86 host or BlueField Arm, the behavior of DOCA Bench is identical. The performance measured is be dependent on the environment.

Info

Only execution on x86 hosts is supported.

Pipelines

DOCA Bench is a highly flexible tool, providing the ability to configure how and what operations occur and in what order. To accomplish this, DOCA Bench uses a pipeline of operations, which are termed "steps". These steps can be a particular function (e.g., Ethernet receive, SHA hash generation, data compression). Therefore, a pipeline of steps can accomplish a number of sequential operations. DOCA Bench can measure the throughput performance or latency of these pipelines, whether running on single or multiple cores/threads.

Info

Currently, DOCA supports running only one pipeline at a time.

Warm-up Period

To ensure correct measurement, the pipelines must be run "hot" (i.e., any initial memory, caches, and hardware subsystems must be running prior to actual performance measurements begin). This is known as the "warm-up" period and, by default, runs approximately 100 jobs through the pipeline before starting measurements.

Defaults

DOCA Bench has a large number of parameters but, to simplify execution, only a few must be supplied to commence a performance measurement. Therefore various parameters have defaults which should be sufficient for most cases. To fine tune performance, users should pay close attention to any default parameters which may affect their pipeline's operation.

Info

When executed, DOCA Bench reports a full list of all parameters and configured values.

Optimizing Performance

To obtain maximum performance, a certain amount of tuning is required for any given environment. While outside the scope of this documentation, it is recommended for users to:

- Avoid using CPU 0 as most OS processes and interrupt request (IRQ) handlers are scheduled to execute on this core
- Enable CPU/IRQ isolation in the kernel boot parameters to remove kernel activities from any cores they wish to execute performance tests on
- On hosts, ensure to not cross any non-uniform memory access (NUMA) regions when addressing the BlueField
- Understand the memory allocation requirements of scenarios, to avoid over-allocating or running into near out-of-memory situations

Supported BlueField Feature Matrix

DOCA Bench can be executed on both host and BlueField Arm environments, and can target BlueField networking platforms.

The following table shows which operations are possible using either DOCA Bench. It also provides two columns showing whether remote memory can be used as an input or output to that operation. For example, DMA operations on the BlueField Arm can access remote memory as an input to pull memory from the host into the BlueField Arm).

	BlueField-2 Networking Platform	BlueField-3 Networking Platform	Execute on Host Side	Execute on BlueField Arm	Remote Memory as Input Allowed?	Remote Memory as Output Allowed?
doca_compress::compress						
doca_compress::decompress					<u>1</u>	
doca_dma						
doca_ec::create						
doca_ec::recover						
doca_ec::update						
doca_sha						
doca_rdma::send						
doca_rdma::receive						
doca_aes_gcm::encrypt						
doca_aes_gcm::decrypt						

	BlueField-2 Networking Platform	BlueField-3 Networking Platform	Execute on Host Side	Execute on BlueField Arm	Remote Memory as Input Allowed?	Remote Memory as Output Allowed?
pt						
doca_cc::client_producer						
doca_cc::client_consumer						
doca_eth::rx						
doca_eth::tx						

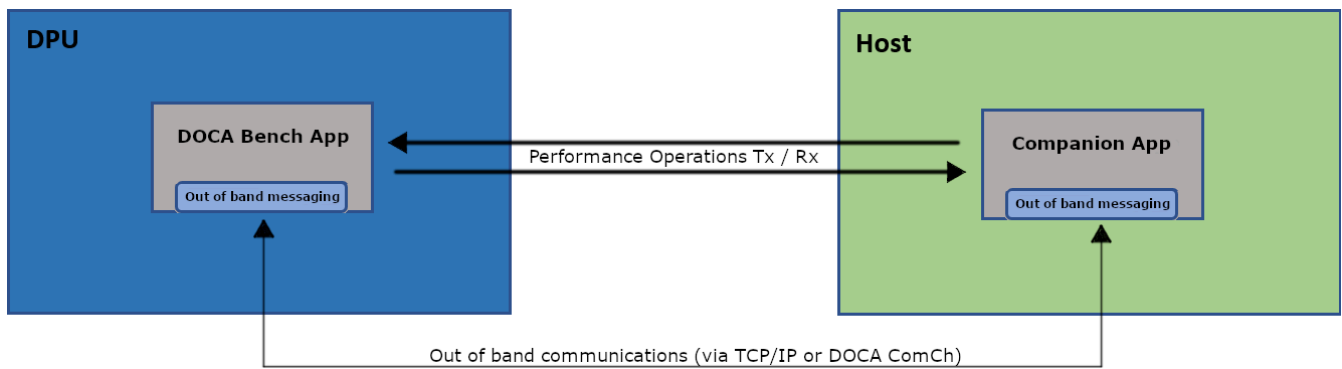
1. Input remote memory is not supported for lz4 decompression __

Remote Operations

A subset of BlueField operations have a remote element, whether this is an RDMA connection, Ethernet connectivity, or memory residing on an x86 host. All these operations require an agent to be present on the far side to facilitate the benchmarking of that particular feature.

In DOCA Bench, this agent is an additional standalone application called the "companion app". It provides the remote benchmarking facilities and is part of the standard DOCA Bench installation.

The following diagram provides an overview of the function and communications between DOCA Bench and the companion app:



In this particular setup, the BlueField executes "DOCA Bench" while the host (x86) is executing the companion App.

DOCA Bench also acts as the controller of the tests, instructing the companion app to perform the necessary operations as required. There is an out-of-band communications channel operating between the two applications that utilizes either standard TCP/IP sockets or a [DOCA ComCh](#) channel (depending on the test scenario/user preferences).

CPU Core and Thread Selection

i Note

Selection of the correct CPU cores and threads has a significant impact on the performance or latency obtained. Read this section carefully.

A key requirement to scaling any application is the number of CPU cores or threads allocated to any given activity. DOCA Bench provides the ability to specify the numbers of cores, and the number of threads to be created per core, to maximize the number of jobs submitted to a given pipeline.

The following care should be given when selecting the number of CPU's or threads:

- Threads that are on cores located on distant NUMA regions (i.e., not the same NUMA region the BlueField is connected to) will experience lower performance and higher latency
- Core 0 is often most used by the OS and should be avoided

- Standard Linux Kernel installations allow the OS to move processes on any CPU core resulting in unexpected drops in performance, or higher latency, due to process switching

The selection of CPU cores is provided through the `--core-mask`, `--core-list`, `--core-count` parameters, while thread selection is made via the `--threads-per-core` parameter.

Device Selection

When executing from a host (x86) environment DOCA Bench can target one or more BlueField devices within an installed environment. When executing from the BlueField Arm, the target is always the local BlueField.

The default method of targeting a given BlueField from either the host or the BlueField Arm is using the `--device` or `-A` parameters, which can be provided as:

- Device PCIe address (i.e., 03:00.0);
- Device IB name (mlx5_0); or
- Device interface name (ens4f0)

From the BlueField Arm environment, DOCA Bench should be targeted at the local PCIe address (i.e., `--device 03:00.0`) or the IB device name (i.e., `mlx5_0`).

Input Data Selection and Sizing of Jobs

DOCA Bench supports different methods of supplying data to jobs and providing information on the amount of data to process per job. These are referred to as "Data Providers".

Input Data Selection

The following subsections provide the modes available to provide data for input into any operation.

File

A single file is used as input to the operation. The contents of the file are not important for certain operations (e.g., DMA, SHA, etc.) but must be valid and specific for others (e.g.,

decompress, etc). The data may be used multiple times and repeated if the operations required more data than the single file contains. For more information on how file data is handled in complex operations, see section "[Command-line Parameters](#)".

File Sets

File sets are a group of files that are primarily used for structured data. The data in the file set is effectively a list of files, separated by a new line that is used sequentially as input data for jobs. Each file pointed to by the file set would have its entire contents read into a single buffer. This is useful for operations that require structured data (i.e., a complete valid block of data, such as decompression or AES).

Random Data

Random data is provided when the actual data required for the given operation is not specific (e.g., DMA).

Note

The use of random data for certain operations may reduce the maximum performance obtained. For example, compressing random data results in lower performance than compressing actual file data (due to the lack of repeating patterns in random data).

Job Sizing

Each job in DOCA Bench consists of three buffers: An original input buffer, an output, and an intermediate buffer.

The input buffer is provided by the data provider for the first step in the pipeline to use, after which the following steps use the output and intermediate buffers (can be sized by using `--job-output-buffer-size`) in a ping-pong fashion. This means, the pipeline can always start with the same deterministic data while allowing for each step to provide its newly generated output data to be used as input to the next step.

The input buffer is specified in one of two ways: using `uniform-job-size` to make every input buffer the exact same size, or using a file set to size each buffer based on the size of the selected input data file(s). Users should ensure the data generated by each step in the pipeline will fit in the provided output buffer.

Controlling Test Duration

DOCA Bench has a variety of ways to control the length of executing tests—whether based on data or time limit.

Limit to Specific Number of Seconds

Using the `--run-limit-seconds` or `-s` parameter ensures that the execution continues for a specific number of seconds.

Limited Through Total Number of Jobs

It may be desirable to measure a specific number of jobs passing through a pipeline. The `--run-limit-jobs` or `-j` parameter is used to specify the exact number of jobs submitted to the pipeline and allowed to complete before execution finishes.

GGA-specific Attributes

As DOCA Bench supports a wide range of both GGA and software based DOCA libraries, the ability to fine tune their invocation is important. Command-line parameters are generally used for configuration options that apply to all aspects of DOCA Bench, without being specific to a particular DOCA library.

Attributes are the method of providing configuration options to a particular DOCA Library, whilst some shared attributes exist the majority of libraries have specific attributes designed to control their specific behavior.

For example, the attribute `doca_ec.data_block_count` allows you to set the data block count for the DOCA EC library, whilst the attribute `doca_sha.algorithm` controls the selection of the SHA algorithm.

For a full list of support attributes, see the "[Command-line Parameters](#)" section.

Info

Due to batching it is possible that more than the supplied jobs are executed.

Command-line Parameters

DOCA Bench allows users to specify a series of operations to be performed and then scale that workload across multiple CPU cores/threads to get an estimation of how that workload performs and some insight into which stage(s), if any, cause performance problems for them. The user can then modify various configuration properties to explore how issues can be tuned to better serve their need.

When running, DOCA Bench creates a number of execution threads with affinities to the specific CPU specified by the user. Each thread creates, uniquely for themselves, a jobs pool (with job data initialized by a data provider) and a pipeline of workload steps.

CPU Core and Thread Count Configuration

There are many factors involved when carrying out performance tests, one of these is the CPU selection:

- The user should consider NUMA regions when selecting which cores to use, as using a CPU which is distant from the device under test can impact the performance achievable

- The user may also wish to avoid core 0 as this is typically the default core for kernel interrupt handlers.

Note

CPU core selection has an impact on the total memory footprint of the test. See section "[Test Memory Footprint](#)" for more details.

--core-mask

Default value: 0x02

Core mask is the simplest way to specify which cores to use but is limited in that it can only specify up to 32 CPUs (0-31). Usage example: `--core-mask 0xF001` selects CPU cores 0, 12, 13, 14, and 15.

--core-list

Core list can specify any/all CPU cores in a given system as a list, range, or combination of the two. Usage example: `--core-list 0,3,6-10` selects CPU cores 0, 3, 6, 7, 8, 9, and 10.

--core-count

The user can select the first N cores from a given core set (list or mask) if desired. Usage example: `--core-count N`.

Info

Sweep testing is supported. See section "[Sweep Tests](#)" for more details.

--threads-per-core -t

To test the impacts of contention within a single CPU core, the user can specify this value so that instead of only one thread being created per core, N threads are created with their affinity mask set to the given core for each core selected. For example, 3 cores and 2 threads per core create 6 threads total.

Info

Sweep testing is supported. See section "[Sweep Tests](#)" for more details.

Device Configuration

The test requires the use of at least one BlueField to execute. With remote system testing, a second device may be required.

--device -A

Specify the device to use from the perspective of the system under test. The value can be for any one of either the device PCIe address (e.g., 03:00.0), the device IB device name (e.g., mlx5_0), or the device interface name (e.g., ens4f0).

--representor -R

This option is used only when performing remote memory operations between a BlueField device and its host using DOCA Comch. This is typically automated by the companion connection string but exists for some developer debug use-cases.

Info

This option used to be important before the companion connection string property was introduced but now is rarely used.

Input Data and Buffer Size Configuration

DOCA Bench supports multiple methods of acquiring data to use to initialize job buffers. The user can also configure the output/intermediate buffers associated with each job.

Info

Input data and buffer size configuration has an impact on the total memory footprint of the test. See section "[Test Memory Footprint](#)" for more details.

--data-provider -I

DOCA Bench supports a number of different input data sources:

- file
- file-set
- random-data

File Data Provider

The file data provider produces uniform/non-structured data buffers by using a single input file. The input data is stripped and or repeated to fill each data buffer as required, returning back to the start of the file each time it is exhausted to collect more data. This is

desirable when the performance of the component(s) under test is meant to show different performance characteristics depending on the input data supplied.

For example, `doca_dma` and `doca_sha` would execute in constant time regardless of the input data. Whereas `doca_compress` would be faster with data with more duplication and slower for truly random data and would produce different output depending on the input data.

Example 1 – Small Input File with Large Buffers

Given a small input data (i.e., smaller than the data buffer size), the file contents are repeated until the buffer is filled and then continue onto the next buffer(s). So, if the input file contained the data `012345` and the user requested two 20-byte buffers, the buffers would appear as follows:

- `01234501234501234501`
- `23450123450123450123`

Example 2 – Large Input File with Smaller Buffers

Given a large input data (i.e., greater than the data buffer size), the file contents are distributed across the data buffers. If the the input file contained the data `0123456789abcdef` and the user requested three 12-byte buffers, the buffers would appear as follows:

- `0123456789ab`
- `cdef01234567`
- `89abcdef0123`

File Set Data Provider

The file set data provider produces structured data. The file set input file itself is a file containing one or more filenames (relative to the input "command working directory (cwd)" not relative to the file set file). Each file listed inside the file set would have its entire contents used as a job buffer. This is useful for operations where the data must be a complete

valid data block for the operation to succeed like decompression with `doca_compress` or decryption with `doca_aes`.

Example - File Set and Its Contents

Given a file set in the "command working directory (cwd)" referring to `data_1.bin` and `data_2.bin` (one file name per line), and `data_1.bin` contains 33 bytes and `data_2.bin` contains 69 bytes, then the data required by the buffers would be filled with these two files in a round-robin manner until the buffers are full . Unlike uniform (non-structured) data each task can have different lengths.

Random-data Data Provider

The random data data provider provides uniform (non-structured) data from a random data source. Each buffer will have unique (pseudo) random bytes of content.

--data-provider-job-count

Default value: 128

Each thread in DOCA Bench has its own allocation of job data buffers to avoid memory contention issues. Users may select how many jobs should be created per thread using this parameter.

Info

Sweep testing is supported. See section "[Sweep Tests](#)" for more details.

--data-provider-input-file

For data providers which use an input file, the filename can be specified here. The filename is relative to the `input_cwd`.

Info

Sweep testing is supported. See section "[Sweep Tests](#)" for more details.

--uniform-job-size

Specify the size of uniform input buffers (in bytes) that should be created.

Note

Does not apply and should not be specified when using structured data input sources.

Info

Sweep testing is supported. See section "[Sweep Tests](#)" for more details.

--job-output-buffer-size

Default value: 16384

Specify the size of output/intermediate buffers (in bytes). Each job has 3 buffers: immutable input buffer and two output/intermediate buffers. This allows for a pipeline to mutate the data an infinite number of times throughout the pipeline while allowing for it to be reset and re-used at the end, and allowing any step to use the new mutated data created by the previous step.

--input-cwd -i

To ease configuration management, the user may opt to use a separate folder for the input data for a given scenario outside of the DOCA build/install directory.

Tip

It is recommended to use relative file paths for the input files.

Example 1 – Running DOCA Bench from Current Working Directory

Considering a user executing DOCA Bench from `/home/bob/doca/build`, values specified in `--data-provider-input-file` and filenames within a file set would search relative to the shell's "command working directory (cwd)": `/home/bob/doca/build`. Their command might look something like:

```
doca_bench --data-provider file-set --data-provider-input-file my_file_set.txt
```

And assuming `my_file_set.txt` contains `data_1.bin`, the files that would be loaded by DOCA Bench after path resolution would be:

- `/home/bob/doca/build/my_file_set.txt`
- `/home/bob/doca/build/data_1.bin`

Example 2 – Running DOCA Bench from Another Directory

Considering the user executed that same test from one level up. Something like:

```
build/doca_bench --data-provider file-set --data-provider-input-file  
build/my_file_set.txt
```

The files to be loaded would be:

- /home/bob/doca/build/my_file_set.txt
- /home/bob/doca/data_1.bin

Notice how both files were loaded relative to the "command working directory (cwd)" and the data file was not loaded relative to the file set.

Example 3 – Example 2 Revisited Using input-cwd

The user can solve this easily by keeping all input files in a single directory and then referring to that directory using the parameter `input-cwd`. In this case, the command like may look something like:

```
build/doca_bench --data-provider file-set --data-provider-input-file my_file_set.txt --  
input-cwd build
```

Note that the value for `--data-provider-input-file` also changed to be relative to the new "command working directory (cwd)".

The files loaded this time are back to being what is expected:

- /home/bob/doca/build/my_file_set.txt
- /home/bob/doca/build/data_1.bin

Test Execution Control

DOCA Bench supports multiple test modes and run execution limits to allow the user to configure the test type and duration.

--mode

Default value: throughput

Select which type of test is to be performed.

Throughput Mode

Throughput mode is optimized to increase the volume of data processed in a given period with little or no regard for latency impact. Throughput mode tries to keep each component under test as busy as possible. A summary of the bandwidth and job execution rate are provided as output.

Bulk-latency Mode

Bulk latency mode strikes a balance between throughput and latency, submitting a batch of jobs and waiting for them all to complete to measure the latency of each job. This mode uses a bucketing mechanism to allow DOCA Bench to handle many millions of jobs worth of results. DOCA Bench keeps a count of the number of jobs that complete within each bucket to allow it to run for long periods of time. A summary of the distribution of results with an ASCII histogram of the results are provided as output. The latency reported is the time taken between the first job submission (for a batch of jobs) until the final job response is received (for that same batch of jobs).

Precision-latency Mode

Precision latency mode executes one job at a time to allow DOCA Bench to calculate the minimum possible latency of the jobs. This causes the components which can process many jobs in parallel to be vastly underutilized and so greatly reduces bandwidth. As this mode records every result individually, it should not be used to execute more than several thousand jobs. Precision latency mode requires 8 bytes of storage for each result, so be mindful of the memory overhead of the number of jobs to be executed.

A statistical analysis including minimum, maximum, mean, median and some percentiles of the latency value are provided as output.

--latency-bucket-range

Default value: 100ms,10ms

Only applicable to bulk-latency mode. Allows the user to specify the starting value of the buckets, and the width of each bucket. There are 100 buckets of the given size and an under flow and over flow bucket for results that fall outside of the central range.

For example:

```
--latency-bucket-range 10us,100us
```

This would start with the lowest bucket measuring $<10\mu\text{s}$ response times, then 100 buckets which are $100\mu\text{s}$ wide, and a final bucket for results taking longer than $>10010\mu\text{s}$

Execution Limits

By default, a test runs forever. This is typically undesirable so the user can specify a limit to the test.

Note

Precision-latency mode only supports job limited execution.

--run-limit-seconds -s

Runs the test for N seconds as specified by the user.

--run-limit-jobs -J

Runs the test until at least N jobs have been submitted, then allowing in-flight jobs to complete before exiting. More jobs than N may be executed based on batch size.

--run-limit-bytes -b

Runs the test until at least N bytes of data have been submitted, then allowing in-flight jobs to complete before exiting. More data may be processed than desired if the limit is not a multiple of the job input buffer size.

Gather/Scatter Support

Gather support involved breaking incoming input data from a single buffer into multiple buffers, which are "gathered" into a single gather list. Currently only gather is supported.

--gather-value

Default value: 1

Specifies the partitioning of input data from a single buffer into a gather list. The value can be specified in two flavors:

- `--gather-value 4` – splits input buffers into 4 parts as evenly as possible with odd bytes in the last segment
- `--gather-value 4KiB` – splits buffers after each 4KB of data. See `doca_bench/utility/byte_unit.hpp` for the list of possible units.

Stats Output

--rt-stats-interval

By default, DOCA Bench emits the results of an iteration once it completes. The user can ask for transient snapshots of the stats as the test progresses by providing the `--rt-stats-interval` argument with a value representing the number of milliseconds between stat prints. The end-result of the run is still displayed as normal.

Note

This may produce a large amount of console output.

--csv-output-file

DOCA Bench can produce an output file as part of its execution which can contain stats and the configuration values used to produce that stat. This is enabled by specifying the `--csv-output-file` argument with a file path as the value. Providing a value for this argument enables CSV stats output (in addition to the normal console output). When performing a sweep test, one line per iteration of the sweep test is populated.

By default, the CSV output contains every possible value. The user can tune this by applying a filter.

--csv-stats

Provide one or more filters (positive or negative) to tune which stats are displayed. The value for this argument is a comma-separated list of filter strings. Negative filters start with a minus sign ('-').

Example 1 – Emit Only Statistical Values (No Configuration Values)

```
--CSV-stats "stats.*"
```

i Note

The quotes around the * prevent the shell from interpreting it as a wild card for filenames in the command.

Example 2 – Emit Statistical Values and Some Configuration Values (Remove Attribute Values)

```
--csv-stats "stats.*",-attribute*
```

--csv-append-mode

Default: false

When enabled, DOCA Bench appends to a CSV file if it exists or creates a new one. It is assumed that all invocation uses the exact same set of output values. This is not verified by DOCA Bench. The user must ensure that all tests that append to the CSV use the same set of output values.

--csv-separate-dynamic-values

A special case which creates a non-standard CSV file. All values that are not supported by sweep tests are reported only once first, then a new line of headers for values emitted during the test, then a row for each test result. This is reserved for an internal use case and should not be relied upon by anyone else.

--enable-environment-information

Instructs DOCA Bench to collect some detailed system information as part of the test startup procedure which are then made available for output in the CSV. These also gather the same details from the companion side if the companion is in use.

Warning

This collection can take a long time (up to a few minutes in some circumstances) to complete, so it is not recommended unless you know you need it.

Remote Memory Testing

Some libraries (e.g., `doca_dma`) support the use of remote memory. To enable this, the user can specify one or both of the remote memory flags `--use-remote-input-buffers` and `--use-remote-output-buffers`. This tells DOCA Bench to use the companion to create a remote mmap. This remote mmap is then used to create buffers that are submitted to the component under test.

Note

These flags should be used with caution and an understanding that if the underlying components under test can support this scenario, there is no automated checking. It is user responsibility to ensure these are used appropriately.

--use-remote-input-buffers

Specifies that the memory used for the initial immutable job input buffers into a pipeline should be backed by an mmap on the remote side.

(i) Note

Requires the companion app to be configured.

--use-remote-output-buffers

Specifies that all output and translation buffers in use are backed by an mmap on the remote side.

(i) Note

Requires the companion app to be configured.

Network Options

--mtu-size

For use with `doca_rdma`. Value is an enum: 256B 512B 1KB 2KB 4KB or `raw_eth`.

--receive-queue-size

For use with `doca_rdma`. Configure the RDMA RQ size independently of the SQ size.

--send-queue-size

For use with `doca_rdma`. Configure the RDMA SQ size independently of the RQ size.

DOCA Lib Configuration Options

--task-pool-size

Default value: 1024

Configure the maximum task pool size used when libraries initialize task pools.

Pipeline Configuration

DOCA Bench is based on a pipeline of operations, This allow for complex test scenarios where multiple components are tested in parallel. Currently only a single chain of operations in a pipeline is supported (but scaled across multiple cores or threads), future versions will allow for varied pipeline's per CPU core.

A pipeline is described as a series of steps. All steps have a few general characteristics:

- Step type: `doca_dma`, `doca_sha`, `doca_compress`, etc.
- An operation category – transformative or non-transformative
- An input data category – structured or non structured

Individual step types may also have some additional metadata information or configuration as defined on a per step basis.

Metadata examples:

- `doca_compress` requires an operation type: `compress` OR `decompress`
- `doca_aes` requires an operation type: `encrypt` OR `decrypt`

- `doca_ec` requires an operation type: `create` , `recover` OR `update`
- `doca_rdma` requires a direction: `send` , `receive` OR `bidir`

Configuration examples:

- `--pipeline-steps doca_dma`
- `--pipeline-steps doca_compress::compress,doca_compress::decompress`

--pipeline-steps

Define the step(s) (comma-separated list) to be executed by each thread of the test.

The following is the list of supported steps:

- `doca_compress::compress`
- `doca_compress::decompress`
- `doca_dma`
- `doca_ec::create`
- `doca_ec::recover`
- `doca_ec::update`
- `doca_sha`
- `doca_rdma::send`
- `doca_rdma::receive`
- `doca_rdma::bidir`
- `doca_aes_gcm::encrypt`
- `doca_aes_gcm::decrypt`
- `doca_cc::client_producer`
- `doca_cc::client_consumer`

- `doca_eth::rx`
- `doca_eth::tx`

Info

Some modules may be unavailable if they were not compiled as part of DOCA when DOCA Bench was compiled.

--attribute

Some of the options are very niche or specific to a single step/mmo type, so they are defined simply as attributes instead of a unique command-line argument.

The following is the list of supported options:

- `doption.mmp.log_qp_depth`
- `doption.mmo.log.num_qps`
- `doption.companion_app.path`
- `doca_compress.algorithm`
- `doca_ec.matrix_count`
- `doca_ec.data_block_count`
- `doca_ec.redundancy_block_count`
- `doca_sha.algorithm`
- `doca_rdma.gid-index`
- `doca_eth.max_burst_size`

- `doca_eth.l3_chksum_offload`
- `doca_eth.l4_chksum_offload`

--warm-up-jobs

Default value: 100

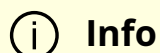
Warm-up serves two purposes:

- Firstly, it runs N tasks in a round robin fashion to get the data path code, tasks memory, and tasks data buffers memory into the CPU caches before the measurement of the test begins
- Secondly, it uses `doca_task_try_submit` instead of `doca_task_submit` to validate the jobs. This validation is not desirable during the proper hot path as it costs time revalidating the task each execution.

The user should ensure their warmup count equals or exceeds the number of tasks being used per thread (see `--data-provider-job-count`).

Companion Configuration


Some tests require a remote system to function. For this purpose, DOCA Bench comes bundled with a companion application (this application is installed as part of the DOCA-for-Host or BlueField packages). The companion is responsible for providing services to DOCA Bench such as creating a `doca_mmap` on the remote side and exporting it for use with remote operations like `doca_dma/doca_sha`, or other `doca_libs` that support remote memory input buffers. DOCA Bench can also provide remote worker processes for libraries that require them such as `doca_rdma` and `doca_cc`. The companion is enabled by providing the `--companion-connection-string` argument. Companion remote workers are enabled by providing either of the arguments `--companion-core-list` OR `--companion-core-mask`.



Info

DOCA Bench requires that an SSH key is configured to allow the user specified to SSH without a password to the remote system using the supplied address (to launch the companion). Refer to your OS's documentation for information on how to achieve this.


The companion connection may also specify the `no-launch` option.

 **Warning**

This is reserved for expert developer use.

The user may also specify a path to a specific companion binary to allow them to test companion binaries not in the default install path using the following command:

```
--attribute  
dooption.companion_app.path=/tmp/my_doca_build/tools/bench/doca_bench_compan
```

 **Warning**

This is reserved for expert developer use.

--companion-connection-string

Specifies the details required to establish a connection to and execute the companion process.

- Example of running DOCA Bench from the host side using the BlueField for the remote side using `doca_comch` as the communications method:

```
--companion-connection-string  
"proto=dcc,mode=DPU,user=bob,addr=172.17.0.1,dev=03:00.0,rep=d8:00.0"
```

- Example of running DOCA Bench from the BlueField side using the host for the remote side using `doca_comch` as the communications method:

```
--companion-connection-string  
"proto=dcc,mode=host,user=bob,addr=172.17.0.1,dev=d8:00.0"
```

- Example of running DOCA Bench on one host with the companion on another host using TCP as the communications method:

```
--companion-connection-string  
"proto=tcp,user=bob,addr=172.17.0.1,port=12345,dev=d8:00.0"
```

Note

For `doca_rdma` only.

--companion-core-list

Works the same way as `--core-list` but defines the cores to be used on the companion side.

Note

Must be at least as large as the `--core-list`.

--companion-core-mask

Works the same way as `--core-mask` but defines the cores to be used on the companion side.

Note

Must be at least as large as the `--core-mask`.

Sweep Tests

--sweep

DOCA Bench supports executing a set of tests based on a number of value ranges. For example, to understand the performance of multi-threading, the user may wish to run the same test for various CPU core counts. They may also wish to vary more than one aspect of the test. Providing one or more `--sweep` parameters activates sweep test mode where every combination of values is tested with a single invocation of DOCA Bench.

The following is a list of the supported sweep test options:

- `core-count`
- `data-provider-input-file`
- `data-provider-job-count`
- `gather-value`
- `mtu-size`
- `receive-queue-size`
- `send-queue-size`

- threads-per-core
- task-pool-size
- uniform-job-size
- doption.mmo.log_qp_depth
- doption.mmo.log_num_qps
- doca_rdma.transport-type
- doca_rdma.gid-index

Sweep test argument values take one of three forms:

- --sweep param,start_value,end_value,+N
- --sweep param,start_value,end_value,*N
- --sweep param,value1,...,valueN

Sweep core count and input file example:

```
--sweep core-count,1,8,*2 --sweep data-provider-input-file,file1.bin,file2.bin
```

This would sweep cores 1-8, inclusive, multiplying the value each time as 1,2,4,8 and two different input files resulting in a cumulative 8 test cases:

Iteration Number	Core Count	Input File
1	1	file1.bin
2	2	file1.bin
3	4	file1.bin
4	8	file1.bin
5	1	file2.bin
6	2	file2.bin
7	4	file2.bin

Iteration Number	Core Count	Input File
8	8	file2.bin

Queries

Device Capabilities

DOCA Bench allows the querying of a device to report which step types are available as well as information of valid configuration options for each step. A device must be specified:

```
tools/bench/doca_bench --device 03:00.0 --query device-capabilities
```

For each supported library, this would report:

- Capable – if that library is enabled in DOCA Bench at compile time (if not capable, installing the library would not make it become available to bench)
- Installed – if the library is installed on the machine executing the query (if not installed, installing it would make it available to bench)
- Library wide attributes
- A list of supported task types (~= step name)
 - If the task type is supported
 - Task specific attributes/capabilities

```
doca_compress:  
Capable: yes  
Installed: yes  
Tasks:  
compress::deflate:  
Supported: no  
compress::lz4:
```

```
Supported: no
compress::lz4_stream:
Supported: no
decompress::deflate:
Supported: yes
Max buffer length: 134217728
decompress::lz4:
Supported: yes
Max buffer length: 134217728
decompress::lz4_stream:
Supported: yes
Max buffer length: 134217728
```

Supported Sweep Attributes

Shows the possible parameters that can be used with the sweep test parameter

```
tools/bench/doca_bench --query sweep-properties
```

Example output:

```
Supported query properties: [
core-count
threads-per-core
uniform-job-size
task-pool-size
data-provider-job-count
gather-value
mtu-size
send-queue-size
receive-queue-size
dooption.mmo.log_qp_depth
dooption.mmo.log_num_qps
doca_rdma.transport-type
```



```
doca_rdma.gid-index
```

```
]
```

Test Memory Footprint

DOCA Bench allocates memory for all the tasks required by the test based on the input buffer size, output/intermediate buffer size, number of cores, number of threads, and number of jobs in use. All jobs contain an input buffer, an output buffer, and an intermediate buffer. The input buffer is immutable and sized based on the data provider in use. The output and intermediate buffers are sized based on the users specification or automatically calculated at the users request. For a library which produces the same amount of output as it consumes (e.g., `doca_dma`), typically the user should set the buffers all to the same size to make things as efficient as possible.

The memory footprint for job buffers can be calculated as: $(\text{number-of-tasks}) * (\text{number-of-cores}) * (\text{number-of-threads-per-core}) * (\text{input-buffer-size} + (\text{output/intermediate-buffer-size} * 2))$. For a 1KB job with the default of 32 jobs, 1 core, and 1 core per thread, the memory footprint would be 96KB.

For sweep testing and structured data input, it can be difficult to pick a suitable output buffer size so the user may choose to specify 0 and have DOCA Bench try all the tasks once to calculate the required output buffer sizes. This only has a cost in terms of time taken to perform the calculation. After this, there is no difference between auto-sizing and manually sizing the jobs output buffers.

Note

When running DOCA Bench on the BlueField and on some host OSs, it may be necessary to increase the limit of how much memory the process can acquire. Consult your OS's documentation for details of how to do this.

NVIDIA DOCA Capabilities Print Tool

This document provides instruction on the usage of the DOCA Capabilities Print Tool.

Introduction

This tool is used to print all the available DOCA libraries and devices. For each DOCA device, the tool prints its representor devices and the capabilities it supports in each DOCA library.

Prerequisites

DOCA 2.6.0 and higher.

Description

This tool can be executed on the host or Arm sides.

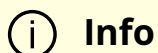
The following capabilities are supported by this tool:

- DOCA device list – print the PCIe device of every available DOCA device and its capabilities
- DOCA representor device list – for every DOCA device, print the PCIe device of every available DOCA representor device and its capabilities
- DOCA library list – print the available DOCA libraries supported by the running OS and their availability for specific OSs
- DOCA library capabilities – for every DOCA device, print the capabilities it supports in every DOCA library

Execution

- To print all the available DOCA devices and their capabilities, run:

```
/opt/mellanox/doca/tools/doca_caps --list-devs
```



Printing the capabilities of a specific DOCA device can be done using the `--pci-addr` flag.

Example output:

```
/opt/mellanox/doca/tools/doca_caps --list-devs
PCI: 0000:03:00.0
ibdev_name mlx5_0
iface_name p0
mac_addr 94:6d:ae:5c:9e:04
ipv4_addr 0.0.0.0
ipv6_addr fe80:0000:0000:0000:966d:aeff:fe5c:9e04
gid_table_size 255
GID[0] fe80:0000:0000:0000:966d:aeff:fe5c:9e04
PCI: 0000:03:00.1
ibdev_name mlx5_1
iface_name p1
mac_addr 94:6d:ae:5c:9e:05
ipv4_addr 0.0.0.0
ipv6_addr fe80:0000:0000:0000:966d:aeff:fe5c:9e05
gid_table_size 255
GID[0] fe80:0000:0000:0000:966d:aeff:fe5c:9e05
PCI: 0000:03:00.0
ibdev_name mlx5_2
iface_name enp3s0f0s0
mac_addr 02:c6:d0:fd:56:d7
ipv4_addr 0.0.0.0
ipv6_addr fe80:0000:0000:0000:00c6:d0ff:fe5d:56d7
gid_table_size 255
GID[0] fe80:0000:0000:0000:00c6:d0ff:fe5d:56d7
PCI: 0000:03:00.1
ibdev_name mlx5_3
iface_name enp3s0f1s0
mac_addr 02:b6:4f:a9:fa:9a
```

```
ipv4_addr 0.0.0.0
ipv6_addr fe80:0000:0000:0000:00b6:4fff:fea9:fa9a
gid_table_size 255
GID[0] fe80:0000:0000:0000:00b6:4fff:fea9:fa9a
```

- To print all the available DOCA representor devices and their capabilities, run:

```
/opt/mellanox/doca/tools/doca_caps --list-rep-devs
```

Info

This command is available only on the Arm side.

Info

Printing the representor list of a specific DOCA device can be done using the `--pci-addr` flag.

Example output:

```
/opt/mellanox/doca/tools/doca_caps --list-rep-devs
PCI: 0000:03:00.0
representor-PCI: 0000:3b:00.0
pci_func_type PF
hotplug no
vuid MT2308XZ0BN0MLNXS0D0F0
representor-PCI: 0000:3b:00.0
pci_func_type SF
hotplug no
vuid MT2308XZ0BN0ECMLNXS0D0F0SF32800
```

```
PCI: 0000:03:00.1
representor-PCI: 0000:3b:00.1
pci_func_type PF
hotplug no
vuid MT2308XZ0BN0MLNXS0D0F1
representor-PCI: 0000:3b:00.1
pci_func_type SF
hotplug no
vuid MT2308XZ0BN0ECMLNXS0D0F1SF32800
PCI: 0000:03:00.0
PCI: 0000:03:00.1
```

- To print all the supported DOCA libraries by the OS and their availability status, run:

```
/opt/mellanox/doca/tools/doca_caps --list-libs
```

Info

Different OSs may support different DOCA libraries.

Example output:

```
/opt/mellanox/doca/tools/doca_caps --list-libs
common installed
aes_gcm installed
apsh installed
argp installed
cc installed
comm_channel installed
compress installed
dma installed
dpa installed
```

```
dpmk_bridge installed
erasure_coding installed
eth installed
ipsec installed
flow installed
flow_ct installed
pcc installed
rdma installed
sha installed
telemetry installed
```

- To print all the capabilities for all the available libraries, that have capabilities, for every DOCA device, run:

```
/opt/mellanox/doca/tools/doca_caps
```

Info

Printing the capabilities of one specific DOCA device can be done using the `--pci-addr` flag.

Info

Printing the capabilities of one specific DOCA library can be done using the `--lib` flag.

Example output:

```
/opt/mellanox/doca/tools/doca_caps
PCI: 0000:03:00.0
```

common
mmap_export_pci supported
mmap_create_from_export_pci supported
hotplug_manager unsupported
rep_filter_all supported
rep_filter_net supported
rep_filter_emulated unsupported
aes_gcm
task_encrypt supported
task_encrypt_get_max_iv_len 12
task_encrypt_tag_96 supported
task_encrypt_tag_128 supported
task_encrypt_128b_key supported
task_encrypt_256b_key supported
task_encrypt_max_buf_size 2097152
task_encrypt_max_list_buf_num_elem 128
task_decrypt supported
task_decrypt_get_max_iv_len 12
task_decrypt_tag_96 supported
task_decrypt_tag_128 supported
task_decrypt_128b_key supported
task_decrypt_256b_key supported
task_decrypt_max_buf_size 2097152
task_decrypt_max_list_buf_num_elem 128
max_num_tasks 65536
cc
server supported
client supported
max_name_len 120
max_msg_size 4080
max_rcv_queue_size 8192
max_send_tasks 8192
max_clients 512
consumer supported
consumer_max_num_tasks 65536
consumer_max_buf_size 2097152

producer supported
producer_max_num_tasks 65536
producer_max_buf_size 2097152
comm_channel
max_service_name_len 120
max_message_size 4080
max_send_queue_size 8192
max_rcv_queue_size 8192
service_max_num_connections 512
compress
task_compress_deflate unsupported
task_compress_deflate_get_max_buf_size 0
task_compress_deflate_get_max_buf_list_len 0
task_decompress_deflate supported
task_decompress_deflate_get_max_buf_size 2097152
task_decompress_deflate_get_max_buf_list_len 128
task_decompress_lz4 supported
task_decompress_lz4_get_max_buf_size 2097152
task_decompress_lz4_get_max_buf_list_len 128
max_num_tasks 65536
dma
task_memcpy supported
max_buf_size 2097152
max_buf_list_len 64
max_num_tasks 65536
dpa
dpa supported
max_threads_per_kernel 128
kernel_max_run_time 12
erasure_coding
task_galois_mul supported
task_create supported
task_update supported
task_recover supported
max_block_size 1048576
max_buf_list_len 128

eth
rxq_cyclic_cpu unsupported
rxq_cyclic_gpu supported
rxq_managed_mempool_cpu unsupported
rxq_managed_mempool_gpu supported
rxq_regular_cpu unsupported
rxq_regular_gpu supported
rxq_max_rcv_buf_list_len 32
rxq_max_packet_size 16384
rxq_max_burst_size 32768
txq_regular_cpu unsupported
txq_regular_gpu supported
txq_max_send_buf_list_len 48
txq_max_iso_header_size 256
txq_txq_max_iso_msg_size 262144
txq_l3_chksum_offload supported
txq_l4_chksum_offload supported
txq_wait_on_time_type unsupported
flow_ct
flow_ct supported
ipsec
task_sa_create supported
task_sa_destroy supported
nvrtd_transport
task_write supported
rc_max_src_buf_list_len 0
dc_max_src_buf_list_len 0
pcc
pcc unsupported
pcc_np unsupported
min_num_threads 0
max_num_threads 0
rdma
task_send supported
task_send_imm supported
task_read supported

task_write supported
task_write_imm supported
task_atomic_cmp_swp supported
task_atomic_fetch_add supported
task_receive supported
rc_transport_type supported
dc_transport_type unsupported
rc_task_receive_get_max_dst_buf_list_len 31
dc_task_receive_get_max_dst_buf_list_len 0
task_remote_net_sync_event_get supported
task_remote_net_sync_event_notify_set supported
task_remote_net_sync_event_notify_add supported
max_send_queue_size 32768
max_rcv_queue_size 32768
max_send_buf_list_len 13
max_message_size 1073741824
sha
sha1 unsupported
sha256 unsupported
sha512 unsupported
sha1_partial unsupported
sha256_partial unsupported
sha512_partial unsupported
max_list_num_elem 0
max_src_buf_size 0
sha1_min_dst_buf_size 0
sha256_min_dst_buf_size 0
sha512_min_dst_buf_size 0
sha1_partial_hash_block_size 0
sha256_partial_hash_block_size 0
sha512_partial_hash_block_size 0
PCI: 0000:03:00.1
common
mmap_export_pci supported
mmap_create_from_export_pci supported
hotplug_manager unsupported

rep_filter_all supported
rep_filter_net supported
rep_filter_emulated unsupported
aes_gcm
task_encrypt supported
task_encrypt_get_max_iv_len 12
task_encrypt_tag_96 supported
task_encrypt_tag_128 supported
task_encrypt_128b_key supported
task_encrypt_256b_key supported
task_encrypt_max_buf_size 2097152
task_encrypt_max_list_buf_num_elem 128
task_decrypt supported
task_decrypt_get_max_iv_len 12
task_decrypt_tag_96 supported
task_decrypt_tag_128 supported
task_decrypt_128b_key supported
task_decrypt_256b_key supported
task_decrypt_max_buf_size 2097152
task_decrypt_max_list_buf_num_elem 128
max_num_tasks 65536
cc
server supported
client supported
max_name_len 120
max_msg_size 4080
max_rcv_queue_size 8192
max_send_tasks 8192
max_clients 512
consumer supported
consumer_max_num_tasks 65536
consumer_max_buf_size 2097152
producer supported
producer_max_num_tasks 65536
producer_max_buf_size 2097152
comm_channel

max_service_name_len 120
max_message_size 4080
max_send_queue_size 8192
max_rcv_queue_size 8192
service_max_num_connections 512
compress
task_compress_deflate unsupported
task_compress_deflate_get_max_buf_size 0
task_compress_deflate_get_max_buf_list_len 0
task_decompress_deflate supported
task_decompress_deflate_get_max_buf_size 2097152
task_decompress_deflate_get_max_buf_list_len 128
task_decompress_lz4 supported
task_decompress_lz4_get_max_buf_size 2097152
task_decompress_lz4_get_max_buf_list_len 128
max_num_tasks 65536
dma
task_memcpy supported
max_buf_size 2097152
max_buf_list_len 64
max_num_tasks 65536
dpa
dpa supported
max_threads_per_kernel 128
kernel_max_run_time 12
erasure_coding
task_galois_mul supported
task_create supported
task_update supported
task_recover supported
max_block_size 1048576
max_buf_list_len 128
eth
rxq_cyclic_cpu unsupported
rxq_cyclic_gpu supported
rxq_managed_mempool_cpu unsupported

rxq_managed_mempool_gpu supported
rxq_regular_cpu unsupported
rxq_regular_gpu supported
rxq_max_rcv_buf_list_len 32
rxq_max_packet_size 16384
rxq_max_burst_size 32768
txq_regular_cpu unsupported
txq_regular_gpu supported
txq_max_send_buf_list_len 48
txq_max_iso_header_size 256
txq_txq_max_iso_msg_size 262144
txq_l3_chksum_offload supported
txq_l4_chksum_offload supported
txq_wait_on_time_type unsupported
flow_ct
flow_ct supported
ipsec
task_sa_create supported
task_sa_destroy supported
nvrdr_transport
task_write supported
rc_max_src_buf_list_len 0
dc_max_src_buf_list_len 0
pcc
pcc unsupported
pcc_np unsupported
min_num_threads 0
max_num_threads 0
rdma
task_send supported
task_send_imm supported
task_read supported
task_write supported
task_write_imm supported
task_atomic_cmp_swap supported
task_atomic_fetch_add supported

task_receive supported
rc_transport_type supported
dc_transport_type unsupported
rc_task_receive_get_max_dst_buf_list_len 31
dc_task_receive_get_max_dst_buf_list_len 0
task_remote_net_sync_event_get supported
task_remote_net_sync_event_notify_set supported
task_remote_net_sync_event_notify_add supported
max_send_queue_size 32768
max_rcv_queue_size 32768
max_send_buf_list_len 13
max_message_size 1073741824
sha
sha1 unsupported
sha256 unsupported
sha512 unsupported
sha1_partial unsupported
sha256_partial unsupported
sha512_partial unsupported
max_list_num_elem 0
max_src_buf_size 0
sha1_min_dst_buf_size 0
sha256_min_dst_buf_size 0
sha512_min_dst_buf_size 0
sha1_partial_hash_block_size 0
sha256_partial_hash_block_size 0
sha512_partial_hash_block_size 0
PCI: 0000:03:00.0
common
mmap_export_pci supported
mmap_create_from_export_pci supported
hotplug_manager unsupported
rep_filter_all unsupported
rep_filter_net unsupported
rep_filter_emulated unsupported
aes_gcm

task_encrypt supported
task_encrypt_get_max_iv_len 12
task_encrypt_tag_96 supported
task_encrypt_tag_128 supported
task_encrypt_128b_key supported
task_encrypt_256b_key supported
task_encrypt_max_buf_size 2097152
task_encrypt_max_list_buf_num_elem 128
task_decrypt supported
task_decrypt_get_max_iv_len 12
task_decrypt_tag_96 supported
task_decrypt_tag_128 supported
task_decrypt_128b_key supported
task_decrypt_256b_key supported
task_decrypt_max_buf_size 2097152
task_decrypt_max_list_buf_num_elem 128
max_num_tasks 65536
cc
server unsupported
client supported
max_name_len 120
max_msg_size 4080
max_rcv_queue_size 8192
max_send_tasks 8192
max_clients 0
consumer supported
consumer_max_num_tasks 65536
consumer_max_buf_size 2097152
producer supported
producer_max_num_tasks 65536
producer_max_buf_size 2097152
comm_channel
max_service_name_len 120
max_message_size 4080
max_send_queue_size 8192
max_rcv_queue_size 8192

service_max_num_connections 0
compress
task_compress_deflate unsupported
task_compress_deflate_get_max_buf_size 0
task_compress_deflate_get_max_buf_list_len 0
task_decompress_deflate supported
task_decompress_deflate_get_max_buf_size 2097152
task_decompress_deflate_get_max_buf_list_len 128
task_decompress_lz4 supported
task_decompress_lz4_get_max_buf_size 2097152
task_decompress_lz4_get_max_buf_list_len 128
max_num_tasks 65536
dma
task_memcpy supported
max_buf_size 2097152
max_buf_list_len 64
max_num_tasks 65536
dpa
dpa supported
max_threads_per_kernel 128
kernel_max_run_time 12
erasure_coding
task_galois_mul supported
task_create supported
task_update supported
task_recover supported
max_block_size 1048576
max_buf_list_len 128
eth
rxq_cyclic_cpu supported
rxq_cyclic_gpu supported
rxq_managed_mempool_cpu supported
rxq_managed_mempool_gpu supported
rxq_regular_cpu supported
rxq_regular_gpu supported
rxq_max_recv_buf_list_len 32

rxq_max_packet_size 16384
rxq_max_burst_size 32768
txq_regular_cpu supported
txq_regular_gpu supported
txq_max_send_buf_list_len 48
txq_max_iso_header_size 256
txq_txq_max_iso_msg_size 262144
txq_l3_chksum_offload supported
txq_l4_chksum_offload supported
txq_wait_on_time_type unsupported
flow_ct
flow_ct unsupported
ipsec
task_sa_create unsupported
task_sa_destroy unsupported
nvrdr_transport
task_write supported
rc_max_src_buf_list_len 0
dc_max_src_buf_list_len 0
pcc
pcc unsupported
pcc_np unsupported
min_num_threads 0
max_num_threads 0
rdma
task_send supported
task_send_imm supported
task_read supported
task_write supported
task_write_imm supported
task_atomic_cmp_swp supported
task_atomic_fetch_add supported
task_receive supported
rc_transport_type supported
dc_transport_type unsupported
rc_task_receive_get_max_dst_buf_list_len 31

dc_task_receive_get_max_dst_buf_list_len 0
task_remote_net_sync_event_get supported
task_remote_net_sync_event_notify_set supported
task_remote_net_sync_event_notify_add supported
max_send_queue_size 32768
max_rcv_queue_size 32768
max_send_buf_list_len 13
max_message_size 1073741824
sha
sha1 unsupported
sha256 unsupported
sha512 unsupported
sha1_partial unsupported
sha256_partial unsupported
sha512_partial unsupported
max_list_num_elem 0
max_src_buf_size 0
sha1_min_dst_buf_size 0
sha256_min_dst_buf_size 0
sha512_min_dst_buf_size 0
sha1_partial_hash_block_size 0
sha256_partial_hash_block_size 0
sha512_partial_hash_block_size 0
PCI: 0000:03:00.1
common
mmap_export_pci supported
mmap_create_from_export_pci supported
hotplug_manager unsupported
rep_filter_all unsupported
rep_filter_net unsupported
rep_filter_emulated unsupported
aes_gcm
task_encrypt supported
task_encrypt_get_max_iv_len 12
task_encrypt_tag_96 supported
task_encrypt_tag_128 supported

task_encrypt_128b_key supported
task_encrypt_256b_key supported
task_encrypt_max_buf_size 2097152
task_encrypt_max_list_buf_num_elem 128
task_decrypt supported
task_decrypt_get_max_iv_len 12
task_decrypt_tag_96 supported
task_decrypt_tag_128 supported
task_decrypt_128b_key supported
task_decrypt_256b_key supported
task_decrypt_max_buf_size 2097152
task_decrypt_max_list_buf_num_elem 128
max_num_tasks 65536
cc
server unsupported
client supported
max_name_len 120
max_msg_size 4080
max_rcv_queue_size 8192
max_send_tasks 8192
max_clients 0
consumer supported
consumer_max_num_tasks 65536
consumer_max_buf_size 2097152
producer supported
producer_max_num_tasks 65536
producer_max_buf_size 2097152
comm_channel
max_service_name_len 120
max_message_size 4080
max_send_queue_size 8192
max_rcv_queue_size 8192
service_max_num_connections 0
compress
task_compress_deflate unsupported
task_compress_deflate_get_max_buf_size 0

task_compress_deflate_get_max_buf_list_len 0
task_decompress_deflate supported
task_decompress_deflate_get_max_buf_size 2097152
task_decompress_deflate_get_max_buf_list_len 128
task_decompress_lz4 supported
task_decompress_lz4_get_max_buf_size 2097152
task_decompress_lz4_get_max_buf_list_len 128
max_num_tasks 65536
dma
task_memcpy supported
max_buf_size 2097152
max_buf_list_len 64
max_num_tasks 65536
dpa
dpa supported
max_threads_per_kernel 128
kernel_max_run_time 12
erasure_coding
task_galois_mul supported
task_create supported
task_update supported
task_recover supported
max_block_size 1048576
max_buf_list_len 128
eth
rxq_cyclic_cpu supported
rxq_cyclic_gpu supported
rxq_managed_mempool_cpu supported
rxq_managed_mempool_gpu supported
rxq_regular_cpu supported
rxq_regular_gpu supported
rxq_max_rcv_buf_list_len 32
rxq_max_packet_size 16384
rxq_max_burst_size 32768
txq_regular_cpu supported
txq_regular_gpu supported

txq_max_send_buf_list_len 48
txq_max_iso_header_size 256
txq_txq_max_iso_msg_size 262144
txq_l3_chksum_offload supported
txq_l4_chksum_offload supported
txq_wait_on_time_type unsupported
flow_ct
flow_ct unsupported
ipsec
task_sa_create unsupported
task_sa_destroy unsupported
nvrtd_transport
task_write supported
rc_max_src_buf_list_len 0
dc_max_src_buf_list_len 0
pcc
pcc unsupported
pcc_np unsupported
min_num_threads 0
max_num_threads 0
rdma
task_send supported
task_send_imm supported
task_read supported
task_write supported
task_write_imm supported
task_atomic_cmp_swap supported
task_atomic_fetch_add supported
task_receive supported
rc_transport_type supported
dc_transport_type unsupported
rc_task_receive_get_max_dst_buf_list_len 31
dc_task_receive_get_max_dst_buf_list_len 0
task_remote_net_sync_event_get supported
task_remote_net_sync_event_notify_set supported
task_remote_net_sync_event_notify_add supported

```
max_send_queue_size 32768
max_rcv_queue_size 32768
max_send_buf_list_len 13
max_message_size 1073741824
sha
sha1 unsupported
sha256 unsupported
sha512 unsupported
sha1_partial unsupported
sha256_partial unsupported
sha512_partial unsupported
max_list_num_elem 0
max_src_buf_size 0
sha1_min_dst_buf_size 0
sha256_min_dst_buf_size 0
sha512_min_dst_buf_size 0
sha1_partial_hash_block_size 0
sha256_partial_hash_block_size 0
sha512_partial_hash_block_size 0
```

NVIDIA DOCA Comm Channel Admin Tool

This document provides instructions on the usage of the DOCA Comm Channel Admin Tool.

Introduction

The Comm Channel Admin Tool is used to print a snapshot of [DOCA Comch](#) (comm channel) connections:

- On the BlueField Arm side, it includes DOCA Comch servers and their current connection information

- On the host side, it includes all active client connections and the server they are connected to
- Only client-to-server control channels are reported; fast path producer/consumer channels are not.

Prerequisites

The Comm Channel Admin Tool is for Linux only and requires an up-to-date BFB bundle or DOCA host packages of at least 2.7, which include in the Resource dump binary.

Description and Execution

The Comm Channel Admin Tool can be executed on the host or Arm CPUs. By default, the tool scans all available PCIe slots to detect supported DOCA devices and reports any Comch information available.

The tool can be run on BlueField Arm or x86 host using the following command:

```
/opt/mellanox/doca/tools/doca_comm_channel_admin
```


Sample Output from BlueField Arm

On the BlueField Arm side, any active DOCA Comch servers are reported:

```
SERVERS:
+-----+-----+-----+-----+-----+
| Server name | PID | Connections | PCIe | Interface Name |
+-----+-----+-----+-----+-----+
| comch1 | 1898009 | 2/512 | 0000:03:00.0 | p0 |
+-----+-----+-----+-----+-----+
| comch3 | 1898011 | 1/512 | 0000:03:00.0 | p0 |
+-----+-----+-----+-----+-----+
| comch6 | 1898014 | 3/512 | 0000:03:00.0 | p0 |
+-----+-----+-----+-----+-----+
| comch2 | 1898010 | 1/512 | 0000:03:00.0 | p0 |
+-----+-----+-----+-----+-----+
| comch7 | 1898015 | 1/512 | 0000:03:00.0 | p0 |
+-----+-----+-----+-----+-----+
| comch5 | 1898013 | 4/512 | 0000:03:00.0 | p0 |
+-----+-----+-----+-----+-----+
| comch8 | 1898016 | 2/512 | 0000:03:00.0 | p0 |
+-----+-----+-----+-----+-----+
| comch4 | 1898012 | 0/512 | 0000:03:00.0 | p0 |
+-----+-----+-----+-----+-----+
```

The following information is available:

- Server Name – the name assigned to the server
- PID – the Linux process ID of the application which created the server
- Connections – the number of connections active on the server out of the total allowed (e.g., 2/512 means 2 active connections of a maximum of 512)
- PCIe – the PCIe address of the device which the server has been detected on
- Interface Name – the interface name associated with the PCIe address

 **Note**

Connections may also be displayed on the BlueField Arm like on x86. This occurs if SF ports are detected here. The interface name associated with the PCIe address indicates the SF port.

Sample Output from x86

The x86 host cannot run DOCA Comch servers. Therefore, individual client connections are reported:


```

CONNECTIONS:
+-----+
| Server name      | PID      | PCIe      | Interface Name |
+=====+
| comch6          | 299693  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch3          | 299688  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch2          | 299687  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch5          | 299689  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch5          | 299692  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch7          | 299696  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch5          | 299690  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch6          | 299694  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch8          | 299697  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch6          | 299695  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch8          | 299698  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch1          | 299686  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch5          | 299691  | 0000:3b:00.0 | ens1f0np0      |
+-----+
| comch1          | 299685  | 0000:3b:00.0 | ens1f0np0      |
+-----+

```

The following information is available:

- Server Name – the name of the BlueField Arm server that a client has connected to
- PID – the Linux process ID of the application running a DOCA Comch client
- PCIe – the PCIe address of the BlueField networking platform which the destination server is running on
- Interface Name – the interface name associated with the PCIe address

NVIDIA DPA Tools

Introduction

DPA tools are a set of executables that enable the DPA application developer and the system administrator to manage and monitor DPA resources and to debug DPA applications.

DPA Tools

DPACC Compiler

CLI name: dpacc

DPACC is a high-level compiler for the DPA processor. It compiles code targeted for the DPA processor into an executable and generates a DPA program.

The DPA program is a host library with interfaces encapsulating the DPA executable. This DPA program can be linked with the host application to generate a host executable where the DPA code is invoked through the FlexIO runtime API.

DPA EU Management Tool

CLI name: dpaeumgmt

[This tool](#) allows users to manage the DPA's EUs which are the basic resource of the DPA. The tool enables the resource control of EUs to optimize the usage of computation resources of the DPA. Using this tool, users may query, create, and destroy EU partitions and groups , thus ensuring proper EU allocation between devices.

FlexIO Build

CLI name: build_flexio_device.sh

The FlexIO Build tool is used to build and compile FlexIO device code into a static library.

It is designed to generate a host library that encapsulating DPA execution. This tool relies on DPACC.

DPA GDB Server Tool

CLI name: dpa-gdbserver

The DPA GDB Server tool enables debugging FlexIO DEV programs.

DPA PS Tool

CLI name: dpa-ps

This tool allows users to monitor running DPA processes and threads.

DPA Statistic Tool

CLI name: dpa-statistics

This tool allows users to monitor and obtain statistics on thread execution per running DPA process and thread.

NVIDIA DOCA PCC Counter Tool

This document provides instruction on the usage of the PCC Counter tool.

Introduction

The PCC Counter tool is used to print PCC-related hardware counters. The output counters help debug the PCC user algorithm embedded in the DOCA PCC application.

Prerequisites

DOCA 2.2.0 and higher.

Description

If NVIDIA® BlueField®-3 is operating in DPU mode, the script must be executed on the Arm side. If BlueField-3 is operating in NIC mode, the script must be executed on the host side.

Info

Refer to [NVIDIA BlueField Modes of Operation](#) for more information on the DPU's modes of operation.

The following performance counters are supported for PCC:

- MAD_RTT_PERF_CONT_REQ – the number of RTT requests received in total
- MAD_RTT_PERF_CONT_RES – the number of RTT responses received in total
- SX_EVENT_WRED_DROP – the number of TX events dropped due to the CC event queue being full
- SX_RTT_EVENT_WRED_DROP – the number of "TX event with RTT request sent indication" dropped due to the CC event queue being full
- ACK_EVENT_WRED_DROP – the number of Ack events dropped due to the CC event queue being full
- NACK_EVENT_WRED_DROP – the number of Nack events dropped due to the CC event queue being full
- CNP_EVENT_WRED_DROP – the number of CNP events dropped due to the CC event queue being full
- RTT_EVENT_WRED_DROP – the number of RTT events dropped due to the CC event queue being full

- HANDLED_SXW_EVENTS – the number of handled CC events related to SXW
- HANDLED_RXT_EVENTS – the number of handled CC events related to RXT
- DROP_RTT_PORT0_REQ – the number of RTT requests dropped in total from port 0
- DROP_RTT_PORT1_REQ – the number of RTT requests dropped in total from port 1
- DROP_RTT_PORT0_RES – the number of RTT responses dropped in total from port 0
- DROP_RTT_PORT1_RES – the number of RTT responses dropped in total from port 1
- RTT_GEN_PORT0_REQ – the number of RTT requests sent in total from port 0
- RTT_GEN_PORT1_REQ – the number of RTT requests sent in total from port 1
- RTT_GEN_PORT0_RES – the number of RTT responses sent in total from port 0
- RTT_GEN_PORT1_RES – the number of RTT responses sent in total from port 1
- PCC_CNP_COUNT – the number of CNP received in total, regardless of whether it is handled or ignored

Execution

To use the PCC Counter:

1. Initialize all supported hardware counters. Run:

```
sudo ./pcc_counters.sh set /dev/mst/mt41692_pciconf0
```

Info

Counters are zeroed after each set command.

2. Query all supported hardware counters. Run:

```
sudo ./pcc_counters.sh query /dev/mst/mt41692_pciconf0
```

Info

The output counters are counted from the time the set command is executed to the time when the query command is issued.

Example output:

```
sudo ./pcc_counters.sh query /dev/mst/mt41692_pciconf0
-----PCC Counters-----
Counter: MAD_RTT_PERF_CONT_REQ Value: 00000000028b85b
Counter: MAD_RTT_PERF_CONT_RES Value: 00000000028b85a
Counter: SX_EVENT_WRED_DROP Value: 0000000000000000
Counter: SX_RTT_EVENT_WRED_DROP Value: 0000000000000000
Counter: ACK_EVENT_WRED_DROP Value: 000000000ccdf4f
Counter: NACK_EVENT_WRED_DROP Value: 0000000000000000
Counter: CNP_EVENT_WRED_DROP Value: 0000000000000000
Counter: RTT_EVENT_WRED_DROP Value: 0000000000000000
Counter: HANDLED_SXW_EVENTS Value: 000000000932543a
Counter: HANDLED_RXT_EVENTS Value: 00000000028b85c
Counter: DROP_RTT_PORT0_REQ Value: 0000000000000000
Counter: DROP_RTT_PORT1_REQ Value: 0000000000000000
Counter: DROP_RTT_PORT0_RES Value: 0000000000000000
Counter: DROP_RTT_PORT1_RES Value: 0000000000000000
Counter: RTT_GEN_PORT0_REQ Value: 0000000000000000
Counter: RTT_GEN_PORT1_REQ Value: 00000000028b85c
Counter: RTT_GEN_PORT0_RES Value: 0000000000000000
Counter: RTT_GEN_PORT1_RES Value: 00000000028b85d
Counter: PCC_CNP_COUNT Value: 0000000000000000
```

NVIDIA DOCA Socket Relay

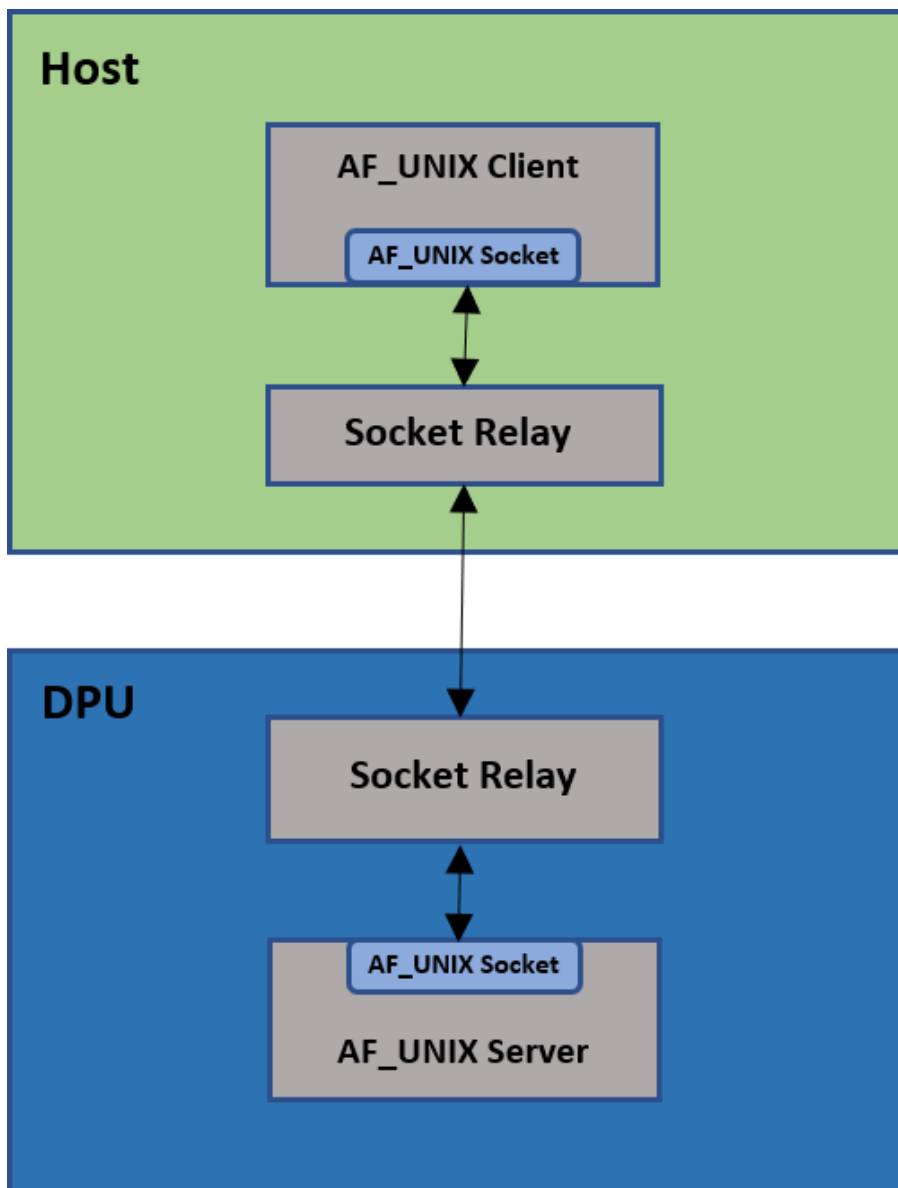
This document describes DOCA Socket Relay architecture, usage, etc.

Introduction

DOCA Socket Relay allows Unix Domain Socket (AF_UNIX family) server applications to be offloaded to the DPU while communication between the two sides is proxied by [DOCA Comm Channel](#).

Socket relay only supports SOCK_STREAM communication with a limit of 512 AF_UNIX application clients.

The tool is coupled to the client AF_UNIX server application. That is, a socket relay instance should be initiated per AF_UNIX server application.



Socket relay is transparent to the application except for the following TCP flows:

- Connection termination must be done by the host side application only
- Once a FIN packet (shutdown system call has been made) is sent by the host side application, data cannot be transferred between the DPU and the host, and the connection must be closed.

The following details the communication flow between the client and server:

- The AF_UNIX client application connects to the socket relay AF_UNIX server in the same way as in the original flow

- The AF_UNIX client application sends SOCK_STREAM packets
- The socket relay (host) AF_UNIX server receives the client application packets, and the Comm Channel client sends them on the channel
- The socket relay (DPU) Comm Channel server receives the client application packets and the AF_UNIX client sends them to the user's AF_UNIX server application

Prerequisites

Windows 10 build 17063 is the minimal Windows version to run DOCA Socket Relay on a Windows host.

Dependencies

NVIDIA® BlueField®-2 firmware version 24.35.1012 or higher.

Execution

To execute DOCA Socket Relay:

```
Usage: doca_socket_relay [DOCA Flags] [Program Flags]
```

DOCA Flags:

-h, --help Print a help synopsis

-v, --version Print program version information

-l, --log-level Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>

-j, --json <path> Parse all command flags from an input json file

Program Flags:

-s, --socket Unix domain socket path, host side will bind to and DPU connect to

-n, --cc-name Comm Channel service name

-p, --pci-addr DOCA Comm Channel device PCI address

```
-r, --rep-pci DOCA Comm Channel device representor PCI address (needed only on DPU)
```

For example (DPU side):

```
doca_socket_relay -s /tmp/sr_server.socket -n cc_channel -p 03:00.0 -r b1:00.0
```

To run `doca_socket_relay` using a JSON file:

```
doca_socket_relay --json [json_file]
```

For example:

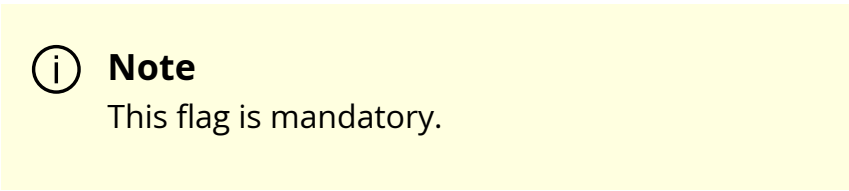
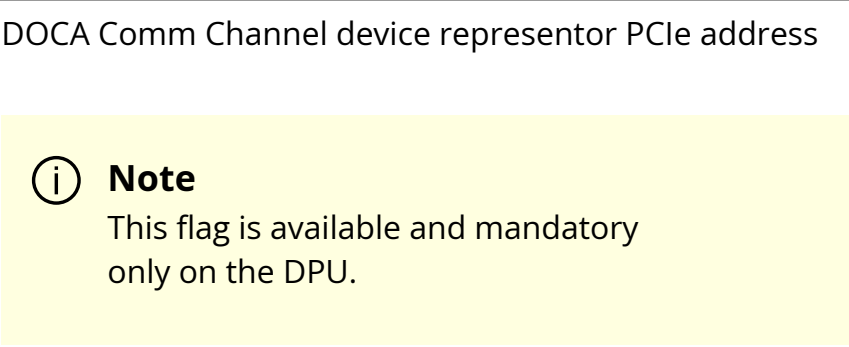
```
doca_socket_relay --json /tmp/doca_socket_relay.json
```

Arg Parser DOCA Flags

Refer to the [DOCA Arg Parser](#) for more information.

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	h	help	Prints a help synopsis	N/A
	v	version	Prints program version information	N/A

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	l	log-level	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with TRACE log level support) 	<pre>"log-level": 60</pre>
	N/A	sdk-log-level	SDK log events are currently unsupported for this tool	N/A
	j	json	Parse all command flags from an input JSON file	N/A
Program flags	s	socket	AF_UNIX (SOCK_STREAM) path. On the host, this is the path of the socket relay AF_UNIX server for the client's application to connect to. On the DPU, this is the path of the client AF_UNIX server application. <div style="background-color: #ffffcc; padding: 5px; margin-top: 10px;"> <p>Note This flag is mandatory.</p> </div>	<pre>"socket" : "/tmp/uds-server.socket"</pre>
	n	cc-name	Comm Channel service name <div style="background-color: #ffffcc; padding: 5px; margin-top: 10px;"> <p>Note This flag is mandatory.</p> </div>	<pre>"cc-name": sr_channel"</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	p	pci-addr	DOCA Comm Channel device PCIe address 	<pre>"pci-addr": b1:00.1</pre>
	r	rep-pci	DOCA Comm Channel device representor PCIe address 	<pre>"rep-pci": b1:02.2</pre>

DOCA Services

This is an overview of the set of services provided by DOCA and their purpose.

Introduction

DOCA services are DOCA-based products, wrapped in a container for fast and easy deployment on top of the NVIDIA® BlueField® DPU. DOCA services leverage DPU capabilities to offer telemetry, time synchronization, networking solutions, and more.

Services containers can be found under the official [NGC catalog](#), labeled under the "DOCA" and "DPU" NGC labels, as well as the built-in NVIDIA platform option ("DOCA") on the container catalog.

For information on the deployment of the services, refer to the [NVIDIA BlueField Container Deployment Guide](#).

Development Lifecycle

DOCA-based containers consist of two main categories:

- DOCA Base Images – containerized DOCA environments for both runtime and development. Used either by developers for their development environment or in the process of containerizing a DOCA-based solution.
- DOCA Services – containerized DOCA-based products

The process of developing and containerizing a DOCA-based product is described in the following sections.

Development

Before containerizing a product, users must first design and develop it using the same process for a bare-metal deployment on the BlueField DPU.

This process consists of the steps:

1. Identifying the requirements for the DOCA-based solution.
2. Reviewing the feature set offered by the DOCA SDK libraries, as shown in detail in their respective [programming guides](#).
3. Starting the development process by following our [Developer Guide](#) to make the best use of our provided tips and tools.
4. Testing the developed solution.

Once the developed product is mature enough, it is time to start containerizing it.

Containerization

In this process, it is recommended to make use of DOCA's provided base-images, as available on DOCA's [NGC page](#).

Three image flavors are provided:

- `base-rt` – includes the DOCA runtime, using the most basic runtime environment required by DOCA's SDK
- `full-rt` – builds on the previous image and includes the full list of runtime packages, which are all user-mode components that can be found under the `doca-runtime` package
- `devel` – builds on the previous image and adds headers and development tools for developing and debugging DOCA applications. This image is particularly useful for multi-stage builds.

All images are preconfigured to use to the DOCA repository of the matching DOCA version. This means that installing an additional DOCA package as part of a Dockerfile / within the development container can be done using the following commands:

```
apt update
apt install <package name>
```

For DOCA and CUDA environments, there are similar flavors for these images combined with [CUDA's images](#):

- base-rt (DOCA) + base (CUDA)
- full-rt (DOCA) + runtime (CUDA)
- devel (DOCA) + devel (CUDA)

Once the containerized solution is mature enough, users may start profiling it in preparation for a production-grade deployment.

Note

DOCA provides base images for both the DPU and the Host. For host-related DOCA base images, please refer to the image tag suffixed with "-host".

Profiling

As mentioned in the [NVIDIA BlueField Container Deployment Guide](#), the current deployment model of containers on top of the DPU is based on kubelet-standalone. And more specifically, this Kubernetes-based deployment makes use of YAML files to describe the resources required by the pod such as:

- CPU
- RAM
- Huge pages

It is recommended to profile your product so as to estimate the resources it requires (under regular deployments, as well as under stress testing) so that the YAML would contain an accurate "resources" section. This allows an administrator to better

understand what the requirements are for deploying your service, as well as allow the k8s infrastructure to ensure that the service is not misbehaving once deployed.

Once done, the containerized DOCA-based product is ready for the final testing rounds, after which it will be ready for deployment in production environments.

Services

Container Deployment

[This page](#) provides an overview and deployment configuration of DOCA containers for NVIDIA® BlueField® DPU.

DOCA BlueMan

DOCA BlueMan service runs in the DPU as a standalone web dashboard and consolidates all the basic information, health, and telemetry counters into a single interface. This friendly, easy-to-use web dashboard acts as a one-stop shop for all the information needed to monitor the DPU.

DOCA Firefly

DOCA Firefly service provides precision time protocol (PTP) based time syncing services to the BlueField DPU . PTP is used to synchronize clocks in a network which, when used in conjunction with hardware support, PTP is capable of sub-microsecond accuracy, which is far better than what is normally obtainable with network time protocol (NTP).

DOCA Flow Inspector

DOCA Flow Inspector service allows monitoring real-time data and extraction of telemetry components which can be utilized by various services for security, big data and more.

Specific mirrored packets can be transferred to Flow Inspector for parsing and analyzing. These packets are forwarded to DTS, which gathers predefined statistics determined by various telemetry providers.

DOCA HBN

DOCA Host-Based Networking service orchestrates network connectivity of dynamically created VMs/containers on cloud servers. HBN service is a BGP router that supports EVPN extension to enable multi-tenant clouds.

At its core, HBN is the Linux networking acceleration driver of the DPU, Netlink-to-DOCA daemon which seamlessly accelerates Linux networking using DOCA hardware programming APIs.

DOCA Management Service

DOCA Management Service (DMS) is a one-stop shop for the user to configure and operate NVIDIA BlueField Networking Platforms and NVIDIA ConnectX Adapters (NICs). DMS governs all scripts/tools of NVIDIA with an easy open API created by the OpenConfig community. The user can configure BlueField or ConnectX for any mode whether locally (ssh) or remotely (grpc). It makes it easy to migrate and bootstrap any customer for any NVIDIA network device.

DOCA Telemetry

DOCA Telemetry service (DTS) collects data from built-in providers and from external telemetry applications. Collected data is stored in binary format locally on the DPU and can be propagated onwards using Prometheus endpoint pulling, pushing to Fluent Bit, or using other supported providers. Exporting NetFlow packets collected using the DOCA Telemetry NetFlow API is a great example of DTS usage.

DOCA UROM

The DOCA UROM service provides a framework for offloading significant portions of HPC software stack directly from the host and to the BlueField networking platform.

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

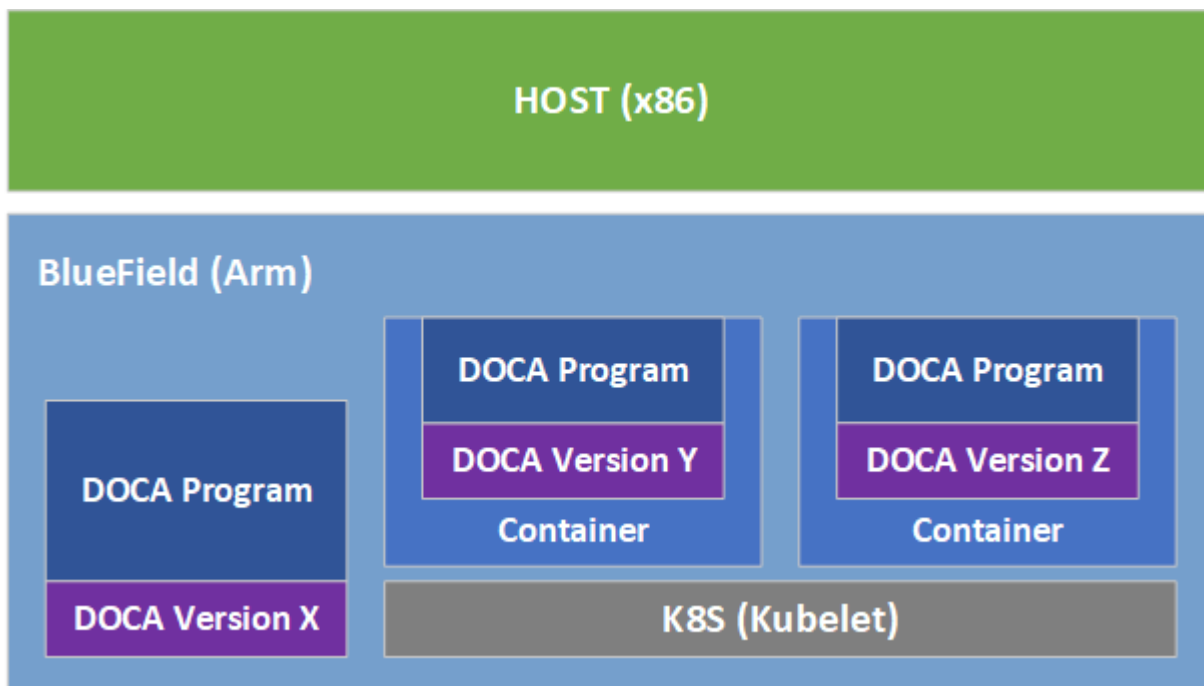
NVIDIA BlueField Container Deployment Guide

This guide provides an overview and deployment configuration of DOCA containers for NVIDIA® BlueField® DPU.

Introduction

DOCA containers allow for easy deployment of ready-made DOCA environments to the DPU, whether it is a DOCA service bundled inside a container and ready to be deployed, or a development environment already containing the desired DOCA version.

Containerized environments enable the users to decouple DOCA programs from the underlying BlueField software. Each container is pre-built with all needed libraries and configurations to match the specific DOCA version of the program at hand. One only needs to pick the desired version of the service and pull the ready-made container of that version from NVIDIA's container catalog.



The different DOCA containers are listed on [NGC](#), NVIDIA's container catalog, and can be found under both the "DOCA" and "DPU" labels.

Prerequisites

- Refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField related software
- BlueField image version required is 3.9.0 and higher

i Note

Container deployment based on **standalone** Kubelet, as presented in this guide, is currently in **alpha version** and is subject to change in future releases.

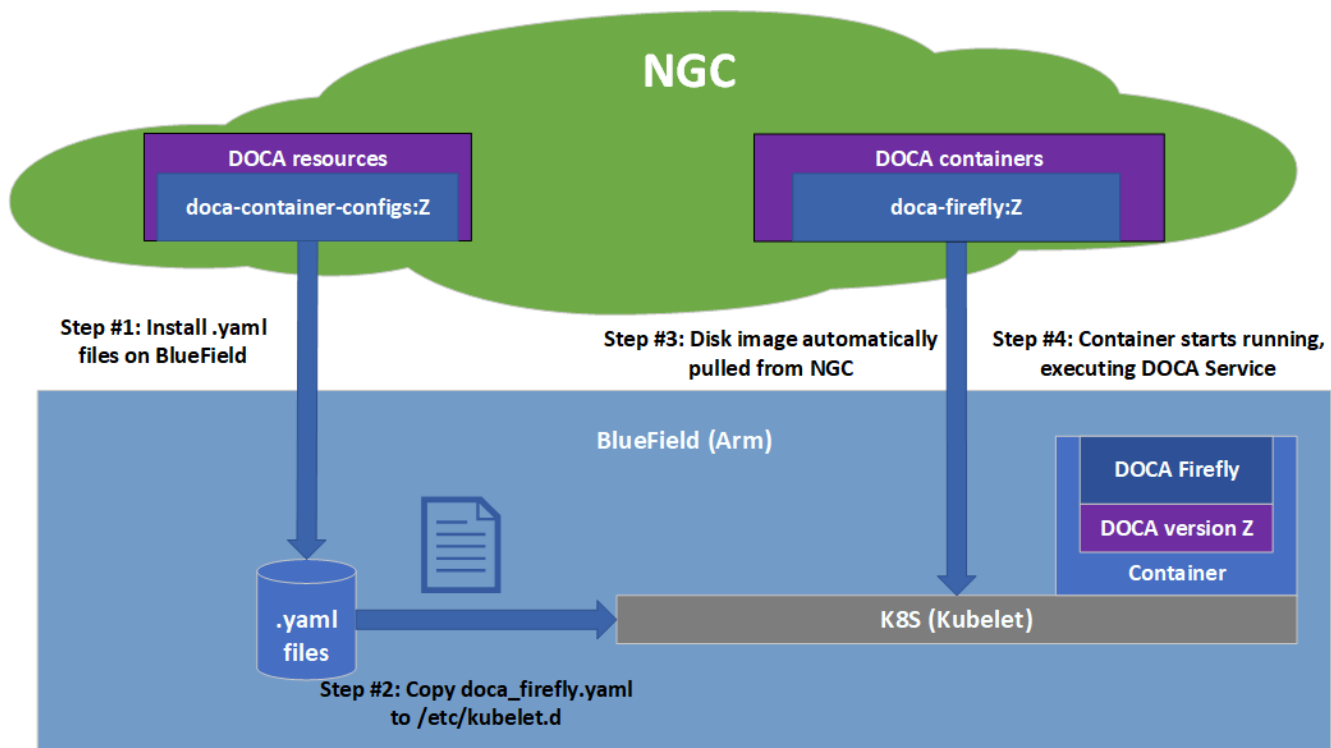
Container Deployment

Deploying containers on top of the BlueField DPU requires the following setup sequence:

1. Pull the container .yaml configuration files.
2. Modify the container's .yaml configuration file.
3. Deploy the container. The image is automatically pulled from NGC.

Some of the steps only need to be performed once, while others are required before the deployment of each container.

What follows is an example of the overall setup sequence using the DOCA Firefly container as an example.



Pull Container YAML Configurations

Note

This step pulls the .yaml configurations from NGC. If you have already performed this step for other DOCA containers you may skip to the next section.

To pull the latest resource version:

1. Pull the entire resource as a *.zip file:

```
wget  
https://api.ngc.nvidia.com/v2/resources/nvidia/doca/doca_container_configs/ver  
-O doca_container_configs_2.7.0v2.zip
```

2. Unzip the resource:

```
unzip -o doca_container_configs_2.7.0v2.zip -d doca_container_configs_2.7.0v2
```

More information about additional versions can be found in the NGC resource page.

Container-specific Instructions

Some containers require specific configuration steps for the resources used by the application running inside the container and modifications for the .yaml configuration file of the container itself.

Refer to the container-specific instructions listed under the container's relevant page on NGC.

Structure of NGC Resource

The DOCA NGC resource downloaded in section "[Pull Container YAML Configurations](#)" contains a configs directory under which a dedicated folder per DOCA version is located. For example, 2.0.2 will include all currently available .yaml configuration files for DOCA 2.0.2 containers.

```
doca_container_configs_2.0.2v1
  configs
    1.2.0
    ...
    2.0.2
    doca_application_recognition.yaml
    doca_blueman.yaml
    doca_devel.yaml
    doca_devel_cuda.yaml
    doca_firefly.yaml
    doca_flow_inspector.yaml
    doca_hbn.yaml
    doca_ips.yaml
    doca_snap.yaml
    doca_telemetry.yaml
    doca_url_filter.yaml
```

In addition, the resource also contains a scripts directory under which services may choose to provide additional helper-scripts and configuration files to use with their services.

The folder structure of the scripts directory is as follows:

```
+ doca_container_configs_2.0.2v1
  +-+ configs
  | +-+ ...
  +-+ scripts
  +-+ doca_firefly <== Name of DOCA Service
  +-+ doca_hbn <== Name of DOCA Service
  | +-+ 1.3.0
  | | +-+ ... <== Files for the DOCA HBN version "1.3.0"
  | +-+ 1.4.0
```

```
| | +-- ... <== Files for the DOCA HBN version "1.4.0"
```

A user wishing to deploy an older version of the DOCA service would still have access to the suitable YAML file (per DOCA release under configs) and scripts (under the service-specific version folder which resides under scripts).

Spawn Container

Once the desired .yaml file is updated, simply copy the configuration file to Kubelet's input folder. Here is an example using the `doca_firefly.yaml`, corresponding to the DOCA Firefly service.

```
cp doca_firefly.yaml /etc/kubelet.d
```

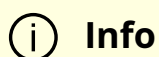
Kubelet automatically pulls the container image from NGC and spawns a pod executing the container. In this example, the DOCA Firefly service starts executing right away and its printouts would be seen via the container's logs.

Review Container Deployment

When deploying a new container, it is recommended to follow this procedure to ensure successful completion of each step in the deployment:

1. View currently active pods and their IDs:

```
sudo crictl pods
```



It may take up to 20 seconds for the pod to start.

When deploying a new container, search for a matching line in the command's output:

```
POD ID CREATED STATE NAME NAMESPACE ATTEMPT RUNTIME
06bd84c07537e 4 seconds ago Ready doca-firefly-my-dpu default 0 (default)
```

2. If a matching line fails to appear, it is recommended to view Kubelet's logs to get more information about the error:

```
sudo journalctl -u kubelet --since -5m
```

Once the issue is resolved, proceed to the next steps.

Info

For more troubleshooting information and tips, refer to the matching section in our [Troubleshooting Guide](#).

3. Verify that the container image is successfully downloaded from NGC into the DPU's container registry (download time may vary based on the size of the container image):

```
sudo crictl images
```

Example output:

```
IMAGE TAG IMAGE ID SIZE
k8s.gcr.io/pause 3.2 2a060e2e7101d 251kB
nvcv.io/nvidia/doca/doca_firefly 1.1.0-doca2.0.2 134cb22f34611 87.4MB
```


4. View currently active containers and their IDs:

```
sudo crictl ps
```

Once again, find a matching line for the deployed container (boot time may vary depending on the container's image size):

```
CONTAINER IMAGE CREATED STATE NAME ATTEMPT POD ID POD
b505a05b7dc23 134cb22f34611 4 minutes ago Running doca-firefly 0
06bd84c07537e doca-firefly-my-dpu
```

5. In case of failure, to see a line matching the container, check the list of all recent container deployments:

```
sudo crictl ps -a
```

It is possible that the container encountered an error during boot and exited right away:

```
CONTAINER IMAGE CREATED STATE NAME ATTEMPT POD ID POD
de2361ec15b61 134cb22f34611 1 second ago Exited doca-firefly 1
4aea5f5adc91d doca-firefly-my-dpu
```

6. During the container's lifetime, and for a short timespan after it exits, once can view the containers logs as were printed to the standard output:

```
sudo crictl logs <container-id>
```

In this case, the user can learn from the log that the wrong configuration was passed to the container:

```
$ sudo crictl logs de2361ec15b61
Starting DOCA Firefly - Version 1.1.0
...
Requested the following PTP interface: p10
Failed to find interface "p10". Aborting
```

Info

For additional information and guides on using `crictl`, refer to the [Kubernetes documentation](#).

Stop Container

The recommended way to stop a pod and its containers is as follows:

1. Delete the `.yaml` configuration file for Kubelet to stop the pod:

```
rm /etc/kubelet.d/<file name>.yaml
```

2. Stop the pod directly (only if it still shows "Ready"):

```
sudo crictl stopp <pod-id>
```

3. Once the pod stops, it may also be necessary to stop the container itself:

```
sudo crictl stop <container-id>
```

Troubleshooting Common Errors

This section provides a list of common errors that may be encountered when spawning a container. These account for the vast majority of deployment errors and are easy to verify first before trying to parse the Kubelet journal log.

Info

If more troubleshooting is required, refer to the matching section in the [Troubleshooting Guide](#).

Yaml Syntax

The syntax of the .yaml file is extremely sensitive and minor indentation changes may cause it to stop working. The file uses spaces (' ') for indentations (two per indent). Using any other number of spaces causes an undefined behavior.

Huge Pages

The container only spawns once all the required system resources are allocated on the DPU and can be reserved for the container. The most notable resource is huge pages.

1. Before deploying the container, make sure that:
 1. Huge pages are allocated as required per container.
 2. Both the amount and size of pages match the requirements precisely.
2. Once huge pages are allocated, it is recommended to restart the container service to apply the change:

```
sudo systemctl restart kubelet.service  
sudo systemctl restart containerd.service
```

3. Once the above operations are completed successfully, the container could be deployed (YAML can be copied to /etc/kubelet.d).

Advanced Troubleshooting

Manual Execution from Within Container - Debugging

i Note

The deployment described in this section requires an in-depth knowledge of the container's structure. As this structure might change from version to version, it is only recommended to use this deployment for debugging, and only after other debugging steps have been attempted.

Although most containers define the `entrypoint.sh` script as the container's `ENTRYPOINT`, this option is only valid for interaction-less sessions. In some debugging scenarios, it is useful to have better control of the programs executed within the container via an interactive shell session. Hence, the `.yaml` file supports an additional execution option.

Uncommenting (i.e., removing `#` from) the following 2 lines in the `.yaml` file causes the container to boot without spawning the container's `entrypoint` script.

```
# command: ["sleep"]  
# args: ["infinity"]
```

In this execution mode, users can attach a shell to the spawned container:

```
crictl exec -it <container-id> /bin/bash
```

Once attached, users get a full shell session enabling them to execute internal programs directly at the scope of the container.

Air-gapped Container Deployment

Container deployment on the BlueField DPU can be done in air-gapped networks and does not require an Internet connection. As explained previously, per DOCA service container, there are 2 required components for successful deployment:

- Container image – hosted on NVIDIA's NGC catalog

- YAML file for the container

From an infrastructure perspective, one additional module is required:

- k8s.gcr.io/pause container image

Pulling Container for Offline Deployment

When preparing an air-gapped environment, users must pull the required container images in advance so they could be imported locally to the target machine:

```
docker pull <container-image:tag>
docker save <container-image:tag> > <name>.tar
```

The following example pulls DOCA Firefly 1.1.0-doca2.0.2:

```
docker pull nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2
docker save nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2 > firefly_v1.1.0.tar
```

Note

Some of DOCA's container images support multiple architectures, causing the `docker pull` command to pull the image according to the architecture of the machine on which it is invoked. Users may force the operation to pull an Arm image by passing the `--platform` flag:

```
docker pull --platform=linux/arm64 <container-image:tag>
```

Importing Container Image

After exporting the image from the container catalog, users must place the created `*.tar` files on the target machine on which to deploy them. The import command is as follows:

```
ctr --namespace k8s.io image import <name>.tar
```

For example, to import the firefly .tar file pulled in the previous section:

```
ctr --namespace k8s.io image import firefly_v1.1.0.tar
```

Examining the status of the operation can be done using the image inspection command:

```
cricctl images
```

Built-in Infrastructure Support

The DOCA image comes pre-shipped with the `k8s.gcr.io/pause` image:

```
/opt/mellanox/doca/services/infrastructure/  
docker_pause_3_2.tar  
enable_offline_containers.sh
```

This image is imported by default during boot as part of the automatic activation of DOCA Telemetry Service (DTS).

Note

Importing the image independently of DTS can be done using the `enable_offline_container.sh` script located under the same directory as the image's *.tar file.

In versions prior to DOCA 4.2.0, this image can be pulled and imported as follows:

- Exporting the image:

```
docker pull k8s.gcr.io/pause:3.2
docker save k8s.gcr.io/pause:3.2 > docker_pause_3_2.tar
```

- Importing the image:

```
ctr --namespace k8s.io image import docker_pause_3_2.tar
cricctl images
IMAGE TAG IMAGE ID SIZE
k8s.gcr.io/pause 3.2 2a060e2e7101d 487kB
```

DOCA Services for Host

A subset of the DOCA services are available for host-based deployment as well. This is indicated in those services' deployment and can also be identified by having container tags on NGC with the *-host suffix.

In contrast to the managed DPU environment, the deployment of DOCA services on the host is based on docker. This deployment can be extended further based on the user's own container runtime solution.

Docker Deployment

DOCA services for the host are deployed directly using Docker.

1. Make sure Docker is installed on your host. Run:

```
docker version
```

If it is not installed, visit the official [Install Docker Engine](#) webpage for installation instructions.

2. Make sure the Docker service is started. Run:

```
sudo systemctl daemon-reload
```

```
sudo systemctl start docker
```

3. Pull the container image directly from NGC (can also be done using the `docker run` command):

1. Visit the NGC page of the desired container.
2. Under the "Tags" menu, select the desired tag and click the paste icon so it is copied to the clipboard.
3. The docker pull command will be as follows:

```
sudo docker pull <NGC container tag here>
```

For example:

```
sudo docker pull nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2-host
```

Note

For DOCA services with deployments on both DPU and host, make sure to select the tag ending with `-host`.

4. Deploy the DOCA service using Docker:

1. The deployment is performed using the following command:

```
sudo docker run --privileged --net=host -v <host directory>:<container directory> -e <env variables> -it <container tag> /entrypoint.sh
```

Info

For more information, refer to [Docker's official documentation](#).

2. The specific deployment command for each DOCA service is listed in their respective deployment guide.

NVIDIA DOCA BlueMan Service Guide

This guide provides instructions on how to use the DOCA BlueMan service on top of NVIDIA® BlueField® DPU.

Introduction

DOCA BlueMan runs in the DPU as a standalone web dashboard and consolidates all the basic information, health, and telemetry counters into a single interface.

All the information that BlueMan provides is gathered from the DOCA Telemetry Service (DTS), starting from DTS version 1.11.1-doca1.5.1.

The screenshot displays the NVIDIA BlueMan web dashboard. The main content area is titled 'System Services' and shows a table of active services. The table has columns for Name, Description, Active, Load, Sub, and Reason. The right sidebar contains system health metrics: CPU Cores Usage (%), Memory Usage (KBytes), Disk Usage (M), and DPU Temperature (°C).

Name	Description	Active	Load	Sub	Reason
accounts-daemon.service	Accounts Service	active	loaded	running	
acpid.service	ACPI event daemon	active	loaded	running	
apparmor.service	Load AppArmor profiles	active	loaded	exited	
apport.service	LSB: automatic crash report generation	active	loaded	exited	
atd.service	Deferred execution scheduler	active	loaded	running	
autofs.service	Automounts filesystems on demand	active	loaded	running	
blk-availability.service	Availability of block devices	active	loaded	exited	
cloud-config.service	Apply the settings specified in cloud-config	active	loaded	exited	
cloud-final.service	Execute cloud userfinal scripts	active	loaded	exited	
cloud-init-local.service	Initial cloud-init job (pre-networking)	active	loaded	exited	
cloud-init.service	Initial cloud-init job (metadata service crawler)	active	loaded	exited	
console-setup.service	Set console font and keymap	active	loaded	exited	
containerd.service	containerd container runtime	active	loaded	running	
cron.service	Regular background program processing daemon	active	loaded	running	
dbus.service	D-Bus System Message Bus	active	loaded	running	
docker.service	Docker Application Container Engine	active	loaded	running	
dpe.service	Nvidia DOCA privileged executor for telemetry service	active	loaded	running	
finalrd.service	Create final runtime dir for shutdown pivot root	active	loaded	exited	
getty@tty1.service	Getty on tty1	active	loaded	running	
gillab-runner.service	Gillab Runner	active	loaded	running	
ifupdown-pre.service	Helper to synchronize boot up for ifupdown	active	loaded	exited	
irqbalance.service	irqbalance daemon	active	loaded	running	
kernel-load.service	LSB: Load kernel image with kexec	active	loaded	exited	
kexec.service	LSB: Execute the kexec -e command to reboot system	active	loaded	exited	
keyboard-setup.service	Set the console keyboard layout	active	loaded	exited	

CPU Cores Usage (%)
Last updated on: 12/21/2022 17:03:49

Core	Usage (%)
core0	~5
core1	~5
core2	~5
core3	~5
core4	~5
core6	~5
core7	~5

Memory Usage (KBytes)
Last updated on: 12/21/2022 17:03:48

Category	Value
Total	16330356
Free	13745000
Used	2291080
Usage	14%

Disk Usage (M)
Last updated on: 12/21/2022 17:03:48

Category	Value
Total	14563
Free	5999
Used	7884
Usage	57%

DPU Temperature (°C)
Last updated on: 12/21/2022 17:03:48

58

Requirements

- BlueField image version 3.9.3.1 or higher
- DTS and the [DOCA Privileged Executer](#) (DPE) daemon must be up and running

Verifying DTS Status

All the information that BlueMan provides is gathered from DTS .

Verify that the state of the DTS pod is ready:

```
$ crictl pods --name doca-telemetry-service
```

Verify that the state of the DTS container is running:

```
$ crictl ps --name doca-telemetry-service
```

Verifying DPE Status

All the information that DTS gathers for BlueMan is from the the DPE daemon .

Verify that the DPE daemon is active:

```
$ systemctl is-active dpe.service  
active
```

If the daemon is inactive, activate it by starting the dpe.service:

```
$ systemctl start dpe.service
```

Service Deployment

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to the [NVIDIA DOCA Container Deployment Guide](#).

DOCA Service on NGC

BlueMan is available on NGC, NVIDIA's container catalog. Service-specific configuration steps and deployment instructions can be found under the service's [container page](#).

Default Deployment – BlueField BSP

BlueMan service is located under `/opt/mellanox/doca/services/blueman /`.

The following is a list of the files under the BlueMan directory:

```
doca_blueman_fe_service_<version>-doca<version>_arm64.tar
doca_blueman_conv_service_<version>-doca<version>_arm64.tar
doca_blueman_standalone.yaml
bring_up_doca_blueman_service.sh
```

Enabling BlueMan Service

Using Script

Run `bring_up_doca_blueman_service.sh`:

```
$ chmod +x
/opt/mellanox/doca/services/blueman/bring_up_doca_blueman_service.sh
$ /opt/mellanox/doca/services/blueman/bring_up_doca_blueman_service.sh
```

Manual Procedure

1. Import images to crictl images:

```
$ cd /opt/mellanox/doca/services/blueman/
```

```
$ ctr --namespace k8s.io image import doca_blueman_fe_service_<version>-  
doca<version>_arm64.tar  
$ ctr --namespace k8s.io image import doca_blueman_conv_service_<version>-  
doca<version>_arm64.tar
```

2. Verify that the DPE daemon is active:

```
$ systemctl is-active dpe.service  
active
```

If the daemon is inactive, activate it by starting the `dpe.service`:

```
$ systemctl start dpe.service
```

3. Copy `blueman_standalone.yaml` to `/etc/kubelet.d/`:

```
$ cp doca_blueman_standalone.yaml /etc/kubelet.d/
```

Verifying Deployment Success

1. Verify that the DPE daemon is active:

```
$ systemctl is-active dpe.service
```

2. Verify that the state of the DTS container is running:

```
$ crictl ps --name doca-telemetry-service
```

3. Verify that the state of the BlueMan service container is running:

```
$ crictl ps --name doca-blue-man-fe
$ crictl ps --name doca-blue-man-conv
```

Configuration

The configuration of the BlueMan back end is located under `/opt/mellanox/doca/services/telemetry/config/blueman_config.ini`. Users can interact with the `blueman_config.ini` file which contains the default range values of the Pass, Warning, and Failed categories which are used in the health page. Changing these values gets reflected in the BlueMan webpage within 60 seconds.

Example of `blueman_config.ini`:

```
;Health Cpu usages Pass, warning, Failed
[Health:CPU_Usages:Pass]
range = 0,80
[Health:CPU_Usages:Warning]
range = 80,90
[Health:CPU_Usages:Failed]
range = 90,100
```

Collected Data

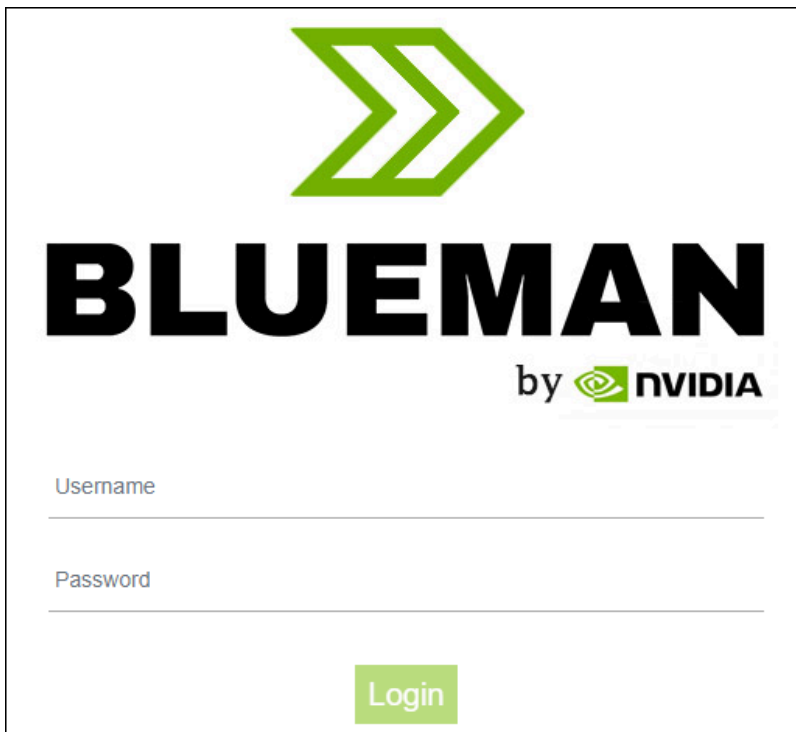
- Info
 - General info – OS name, kernel, part number, serial number, DOCA version, driver, board ID, etc.
 - Installed packages – list of all installed packages on the DPU including their version
 - CPU info – vendor, cores, model, etc.
 - FW info – all the `mlxconfig` parameters with default/current/next boot data
 - DPU operation mode

- Health
 - System service
 - Kernel modules
 - Dmesg
 - DOCA services
 - Port status of the PF and OOB
 - Core usage and processes running on each core
 - Memory usage
 - Disk usage
 - Temperature
- Telemetry – all telemetry counters that come from DTS according to the enabled providers displayed on tables
 - Users have the ability to build graphs of specific counters

Connecting to BlueMan Web Interface

To log into BlueMan, enter the IP address of the DPU's OOB interface (http://<DPU_OOB_IP>) to a web browser located in the same network as the DPU .

The login credentials to use are the same pair used for the SSH connection to the DPU.



Troubleshooting

For general troubleshooting, refer to the [NVIDIA DOCA Troubleshooting Guide](#).

For container-related troubleshooting, refer to the "Troubleshooting" section in the [NVIDIA DOCA Container Deployment Guide](#).

The following are additional troubleshooting tips for DOCA BlueMan:

- The following error message in the login page signifies a failure to connect to the DPE daemon: "The service is currently unavailable. Please check server up and running."

1. Restart the DPE daemon:

```
$ systemctl restart dpe.service
```

2. Verify that DTS is up and running by following the instructions in section "[Verifying DTS Status](#)".

- If the message "Invalid Credentials" appears in the login page, verify that the username and password are the same ones used to SSH to the DPU.

- If all of the above is configured as expected and there is still some failure to log in, it is recommended to check if there are any firewall rules that block the connection.
- For other issues, check the `/var/log/syslog` and `/var/log/doca/telemetry/blueman_service.log` log file.

NVIDIA DOCA Firefly Service Guide

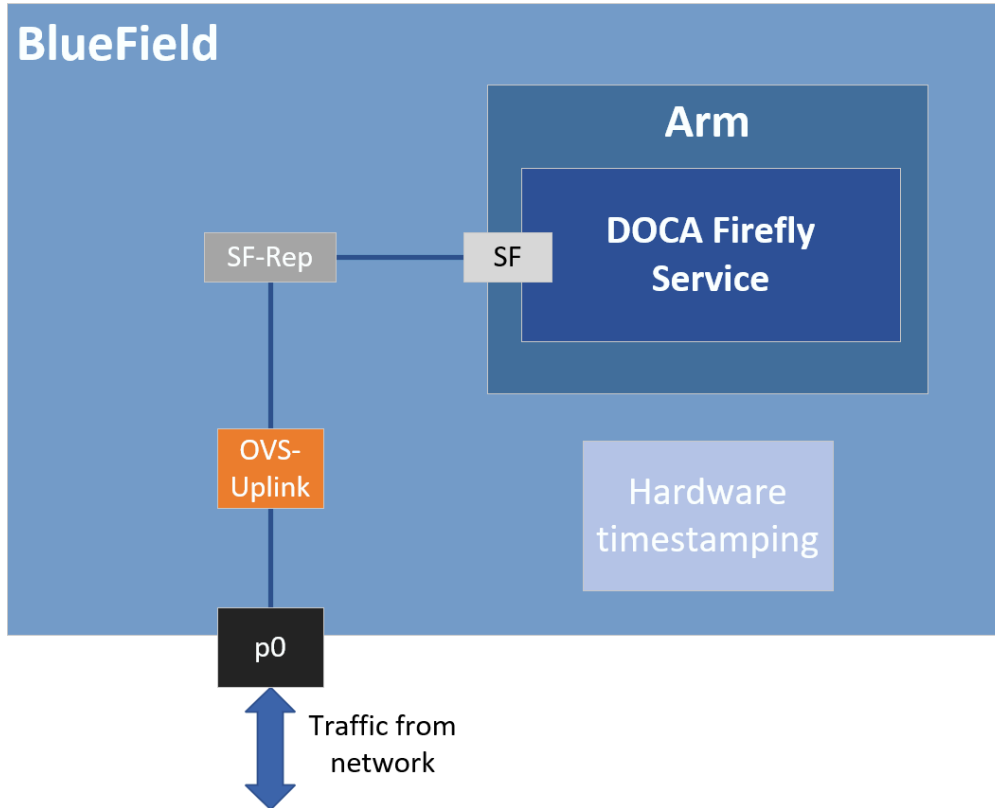
This guide provides instructions on how to use the DOCA Firefly service container on top of NVIDIA® BlueField® DPU.

Introduction

DOCA Firefly Service provides precision time protocol (PTP) based time syncing services to the BlueField DPU .

PTP is a protocol used to synchronize clocks in a network. When used in conjunction with hardware support, PTP is capable of sub-microsecond accuracy, which is far better than is what is normally obtainable with network time protocol (NTP). PTP support is divided between the kernel and user space. The `ptp4l` program implements the PTP boundary clock and ordinary clock. With hardware time stamping, it is used to synchronize the PTP hardware clock to the master clock.

Host (x86)



Requirements

Some of the features provided by Firefly require specific BlueField DPU hardware capabilities:

- PTP – Supported by all BlueField DPUs
- PPS – Requires BlueField DPU with PPS capabilities
- SyncE - Requires converged card BlueField DPUs

Failure to run PPS due to missing hardware support will be noted in the service's output. However, the service will continue to run the timing services it can provide on the provided hardware.

Firmware Version

Firmware version must be 24.34.1002 or higher.

BlueField BSP Version

Supported BlueField image versions are 3.9.0 and higher.

Embedded Mode

Configuring Firmware Settings on DPU for Embedded Mode

1. Set the DPU to embedded mode (default mode):

```
sudo mlxconfig -y -d 03:00.0 s INTERNAL_CPU_MODEL=1
```

2. Enable the real time clock (RTC):

```
sudo mlxconfig -d 03:00.0 set REAL_TIME_CLOCK_ENABLE=1
```

3. [Graceful shutdown](#) and power cycle the DPU to apply the configuration.

4. You may check the DPU mode using the following command:

```
sudo mlxconfig -d 03:00.0 q | grep INTERNAL_CPU_MODEL
# Example output
INTERNAL_CPU_MODEL          EMBEDDED_CPU(1)
```

Ensuring OVS Hardware Offload

DOCA Firefly requires that hardware offload is activated in Open vSwitch (OVS). This is enabled by default as part of the BFB image installed on the DPU.

To verify the hardware offload configuration in OVS:

```
sudo ovs-vsctl get Open_vSwitch . other_config | grep hw-offload
# Example output
    {hw-offload="true"}
```

If inactive:

1. Activate hardware offloading by running:

```
sudo ovs-vsctl set Open_vSwitch . other_config:hw-offload=true;
```

2. Restart the OVS service:

```
sudo /etc/init.d/openvswitch-switch restart
```

3. [Graceful shutdown](#) and power cycle the DPU to apply the configuration.

Helper Scripts

Firefly's deployment contains a script to help with the configuration steps required for the network interface in embedded mode:

- `scripts/doca_firefly/<firefly-version>/prepare_for_embedded_mode.sh`
- `scripts/doca_firefly/<firefly-version>/set_new_sf.sh`

The latest DOCA Firefly version is 1.4.0.

Both scripts are included as part of DOCA's container resource which can be downloaded according to the instructions in the [NVIDIA DOCA Container Deployment Guide](#). For more information about the structure of the DOCA container resource, refer to section "[Structure of NGC Resource](#)" in the deployment guide.

Note

Due to technical limitations of the NGC resource, both scripts are provided without execute (+x) permissions. This could be resolved by running the following command:

```
chmod +x scripts/doca_firefly/<firefly-version>/*.sh
```

prepare_for_embedded_mode.sh

This script automates all the steps mentioned in section "[Setting Up Network Interfaces for Embedded Mode](#)" and configures a freshly installed BFB image to the settings required by DOCA Firefly.

Notes:

- The script deletes all previous OVS settings and creates a single OVS bridge that matches the definitions in section "[Setting Up Network Interfaces for Embedded Mode](#)"
- The script should only be run once when connecting to the DPU for the first time or after a power cycle
- The only manual step required after using this script is configuring the IP address for the created network interface (step 5 in section "[Setting Up Network Interfaces for Embedded Mode](#)")

Script arguments:

- SF number (checks if already exists)

Examples:

- Prepare OVS settings using an SF indexed 4:

```
chmod +x ./*.sh
./prepare_for_embedded_mode.sh 4
```

The script makes use of `set_new_sf.sh` as a helper script.

set_new_sf.sh

Creates a new trusted SF and marks it as "trusted".

Script arguments:

- PCIe address
- SF number (checks if already exists)
- MAC address (if absent, a random address is generated)

Examples:

- Create SF with number "4" over port 0 of the DPU:

```
./set_new_sf.sh 0000:03:00.0 4
```

- Create SF with number "5" over port 0 of the DPU and a specific MAC address:

```
./set_new_sf.sh 0000:03:00.0 5 aa:bb:cc:dd:ee:ff
```

- Create SF with number "4" over port 1 of the DPU:

```
./set_new_sf.sh 0000:03:00.1 4
```

The first two examples should work out of the box for a BlueField-2 device and create SF4 and SF5 respectively.

Setting Up Network Interfaces for DPU Mode

1. Create a trusted SF to be used by the service according to the [Scalable Function Setup Guide](#) .

(i) Note

The following instructions assume that the SF has been created using index 4.

2. Create the required OVS setting as is shown in the [architecture diagram](#):

```
$ sudo ovs-vsctl add-br uplink
$ sudo ovs-vsctl add-port uplink p0
$ sudo ovs-vsctl add-port uplink en3f0pf0sf4
# This port is needed to ensure we have traffic host<->network as well
$ sudo ovs-vsctl add-port uplink pf0hpf
```

3. Verify the OVS settings:

```
sudo ovs-vsctl show
Bridge uplink
Port pf0hpf
Interface pf0hpf
Port en3f0pf0sf4
Interface en3f0pf0sf4
Port p0
Interface p0
Port uplink
Interface uplink
type: internal
```

4. Enable TX timestamping on the SF interface (not the representor):

```
# tx port timestamp offloading
sudo ethtool --set-priv-flags enp3s0f0s4 tx_port_ts on
```

5. Enable the interface and set an IP address for it:

```
# configure ip for the interface:
sudo ifconfig enp3s0f0s4 <ip-addr> up
```

6. Configure OVS to support TX timestamping over this SF and multicast traffic in general:

```
# Multicast-related definitions
$ sudo ovs-vsctl set Bridge uplink mcast_snooping_enable=true
$ sudo ovs-vsctl set Bridge uplink other_config:mcast-snooping-disable-flood-unregistered=true
$ sudo ovs-vsctl set Port p0 other_config:mcast-snooping-flood=true
$ sudo ovs-vsctl set Port p0 other_config:mcast-snooping-flood-reports=true
# PTP-related definitions
$ sudo ovs-ofctl add-flow uplink
in_port=en3f0pf0sf4,udp,tp_src=319,actions=output:p0
$ sudo ovs-ofctl add-flow uplink
in_port=p0,udp,tp_src=319,actions=output:en3f0pf0sf4
$ sudo ovs-ofctl add-flow uplink
in_port=en3f0pf0sf4,udp,tp_src=320,actions=output:p0
$ sudo ovs-ofctl add-flow uplink
in_port=p0,udp,tp_src=320,actions=output:en3f0pf0sf4
```

Note

If your OVS bridge uses a name other than `uplink`, make sure that the used name is reflected in the `ovs-vsctl` and `ovs-ofctl` commands. For instance:

```
$ sudo ovs-vsctl set Bridge <bridge-name>
mcast_snooping_enable=true
```

Separated Mode

Configuring Firmware Settings on DPU for Separated Mode

1. Set the BlueField mode of operation to "Separated":

```
sudo mlxconfig -y -d 03:00.0 s INTERNAL_CPU_MODEL=0
```

2. Enable RTC:

```
sudo mlxconfig -d 03:00.0 set REAL_TIME_CLOCK_ENABLE=1
```

3. [Graceful shutdown](#) and power cycle the DPU to apply the configuration.

4. You may check the BlueField's operation mode using the following command:

```
sudo mlxconfig -d 03:00.0 q | grep INTERNAL_CPU_MODEL  
# Example output  
INTERNAL_CPU_MODEL          SEPARATED_HOST(0)
```

Setting Up Network Interfaces for Separated Mode

1. Make sure that that p0 is not connected to an OVS bridge:

```
sudo ovs-vsctl show
```

2. Enable TX timestamping on the p0 interface:

```
# TX port timestamp offloading (assuming PTP interface is p0)  
sudo ethtool --set-priv-flags p0 tx_port_ts on
```

3. Enable the interface and set an IP address for it:

```
# Configure IP for the interface
```



```
sudo ifconfig p0 <ip-addr> up
```

Host-based Deployment

Host-based deployment requires the same configuration described under section "[Separated Mode](#)".

Service Deployment

DPU Deployment

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to [NVIDIA DOCA Container Deployment Guide](#).

Service-specific configuration steps and deployment instructions can be found under the service's [container page](#).

Note

DOCA Firefly can also be deployed on DPUs not connected to the Internet. For instructions, refer to the relevant section in the [NVIDIA DOCA Container Deployment Guide](#).

Host Deployment

DOCA Firefly has a version adapted for host-based deployments. For more information about the deployment of DOCA containers on top of a host, refer to the [NVIDIA BlueField DPU Container Deployment Guide](#).

The following is the docker command for deploying DOCA Firefly on the host:

```
sudo docker run --privileged --net=host -v /var/log/doca/firefly:/var/log/firefly -v /etc/firefly:/etc/firefly -e PTP_INTERFACE='eth2' -it nvcr.io/nvidia/doca/doca_firefly:1.4.0-doca2.7.0-host /entrypoint.sh
```

Where:

- Additional YAML configs may be passed as environment variables as additional `-e` key-value pairs as done with `PTP_INTERFACE` above
- The exact container tag should be the desired tag as chosen on DOCA Firefly's [NGC page](#)

Configuration

All modules within the service have configuration files that allow customizing various settings, both general and PTP-related.

Built-In Config File

Each profile has its own base PTP configuration file for `ptp4l`. For example, the Media profile PTP configuration file is `ptp4l-media.conf`.

The built-in PTP configuration files can be found in section "[PTP Profile Default Config Files](#)". For ease-of-use, those files are provided as part of DOCA's container resource as downloaded from NGC and are placed under Firefly's `configs` directory (`scripts/doca_firefly/<firefly version>/configs`).

Note

When using a built-in configuration file, Firefly uses the files as stored within the container itself in the `/etc/linuxptp` directory. The configuration files included in the NGC resource are only provided for ease of access. Modifying them does **not** impact the configuration

used in practice by the container. Instead, updates to the configuration should be done as described in the following sections.

Custom Config File

Instead of using a profile's base config file, users can create a file of their own, for each of the modules.

To set a custom config file, users should locate their config file in the directory `/etc/firefly` and set the config file name in DOCA Firefly's YAML file.

For example, to set a custom `linuxptp` config file, the user can set the parameter `PTP_CONFIG_FILE` in the YAML file:

```
- name: PTP_CONFIG_FILE
  value: my_custom_ptp.conf
```

In this example, `my_custom_ptp.conf` should be placed at `/etc/firefly/my_custom_ptp.conf`.

Note

A config file must not define values for the UDS-related ports (`/var/run/ptp4l` and `/var/run/ptp4lro`), as those will impact internal container behavior. Such settings will prompt a warning and will be ignored when preparing the finalized configuration (See more in the next sections).

Overriding Specific Config File Parameters

Instead of replacing the entire config file, users may opt to override specific parameters. This can be done using the following variable syntax in the YAML file:

`CONF_<TYPE>_<SECTION>_<PARAMETER_NAME>`.

- TYPE – either PTP, MONITOR, PHC2SYS, SYNCE, or SERVO
- SECTION – the section in the config file that the parameter should be placed in

Note

If the specified section does not already exist in the config file, a new section is created unless it refers to a PTP network interface that has not been included in the PTP_INTERFACE YAML field.

- PARAMETER_NAME – the config parameter name as should be placed in the config file

Note

If the parameter name already exists in the config file, then the value is changed according to the value provided in the .yaml file. If the parameter name does not already exist in the config file, then it is added.

For example, the following variable in the YAML file definition changes the value of the parameter `priority1` under section `global` in the PTP config file to 64.

```
- name: CONF_PTP_global_priority1
  value: "64"
```

Note

Configuring `unicast_master_table` through the YAML file is not supported due to the structure of the table (i.e., multiple entries sharing the same key).

Ensuring and Debugging Correctness of Config Files

The previous sections describe 2 layers for the configuration file definitions:

- Basic configuration file – either a built-in config file or a custom config file
- Adding/overriding values to/from the YAML file

In practice, there are slightly more layers in place, and the precedence is as follows (presented in increasing order):

- Default configuration values of the PTP program (`ptp4l` for instance) – holds values of all available configuration options
- Your chosen configuration file – contains a subset of options
- Definitions from the YAML file – narrower subset
- Firefly mandatory values

When combining the supplied configuration file with the definitions from the YAML file, Firefly goes over those definitions and checks them against a predefined set of configuration options:

- Warning only – warns if a certain value leads to known issues in a supported deployment scenario
- Override – container-internal definitions that should not be set by the user and will be overridden by Firefly

Suitable log messages are provided in either case:

```

# Example for a warning
2023-01-31 11:55:13 - Firefly - Config - INFO - Missing explicit definition
"fault_reset_interval", verifying default value instead: "4"
2023-01-31 11:55:13 - Firefly - Config - WARNING - Value "4" for definition
"fault_reset_interval" will be invalid in Embedded Mode, expected a value lesser or
equal to "1"
2023-01-31 11:55:13 - Firefly - Config - WARNING - Continuing with invalid value
# Example for an override
2023-01-31 11:21:00 - Firefly - Config - WARNING - Invalid value "/var/run/ptp4l2" for
definition "uds_address", expected "/var/run/ptp4l"
2023-01-31 11:21:00 - Firefly - Config - INFO - Setting definition "uds_address" value
to the following: "/var/run/ptp4l"

```

At the end of this process, an updated configuration file is generated by Firefly to be used later by the various time providers (as mentioned below). To avoid accidental modification of a user-supplied configuration file or permission issues, the finalized file is generated within the container under the `/tmp` directory.

For instance, if using a custom configuration file named `my_custom_ptp.conf` under the `/etc/firefly` directory on the DPU, the updated file will reside within the container at the following path: `/tmp/my_custom_ptp.conf`.

For troubleshooting possible issues with the configuration file, one can do one of the following:

- Connect to the container directly as is explained in the [debugging finalized configuration file](#) bullet under "[Troubleshooting](#)".
- Map the container's `/tmp` directory to the DPU using the built-in support in the YAML file:
 - Before the change:

```

# Uncomment when debugging the finalized configuration files used -
Part #1
#- name: debug-firefly-volume
# hostPath:

```

```
# path: /tmp/firefly
# type: DirectoryOrCreate
containers:
...
volumeMounts:
- name: logs-firefly-volume
  mountPath: /var/log/firefly
- name: conf-firefly-volume
  mountPath: /etc/firefly
# Uncomment when debugging the finalized configuration files used -
Part #2
#- name: debug-firefly-volume
# mountPath: /tmp
```

- o After the change:

```
# Uncomment when debugging the finalized configuration files used -
Part #1
- name: debug-firefly-volume
  hostPath:
  path: /tmp/firefly
  type: DirectoryOrCreate
containers:
...
volumeMounts:
- name: logs-firefly-volume
  mountPath: /var/log/firefly
- name: conf-firefly-volume
  mountPath: /etc/firefly
# Uncomment when debugging the finalized configuration files used -
Part #2
- name: debug-firefly-volume
  mountPath: /tmp
```

i Note

The finalized configuration file keeps the sections and config options in the same order as they appear in the original file, yet the file is stripped from spare new lines or comment lines. This should be taken into considerations when directly accessing it during a debugging session.

Description

Providers

DOCA Firefly Service uses the following third-party providers to provide time syncing services:

- Linuxptp - Version v4.2
 - PTP – PTP service, provided by the PTP4L program
 - PHC2SYS – OS time calibration, provided by the PHC2SYS program
- Testptp
 - PPS - PPS settings service

In addition, DOCA Firefly Service also makes use of the following NVIDIA modules:

- SyncE
 - SYNCE – Synchronous Ethernet Daemon (synced)
- Firefly
 - MONITOR - Firefly PTP Monitor
- Firefly

- SERVO - Firefly PTP Servo

Each of the providers can be enabled, disabled, or set to use the setting defined by the configuration profile:

- YAML setting – <provider name>_STATE
- Supported values – enable, disable, defined_by_profile

Note

For the default profile settings per provider, refer to the table under section "[Profiles](#)".

An example YAML setting for specifically disabling the phc2sys provider is the following:

```
- name: PHC2SYS_STATE
  value: "disable"
```

Note

The defined_by_profile setting is only available for well-defined profiles. As such, it cannot be used when the custom profile is selected. For more information about the profile settings, refer to the table under section "[Profiles](#)".

Profiles

DOCA Firefly Service includes profiles which represent common use cases for the Firefly service that provide a different default configuration per profile:

	Default	Media	Telco (L2)	Custom
Purpose	Any user that requires PTP	Media productions	Telco networks	Custom configuration for a dedicated user scenario
PTP	Enabled	Enabled	Enabled	No default. Enable/disable should be set by the user.
PTP profile	PTP default profile	SMPTE 2059-2	G.8275.1	Set by the user
PTP Client/Server ¹	Both	Client-only	Both	Set by the user
PHC2SYS	Enabled	Enabled	Enabled	No default. Enable/disable should be set by the user.
PPS (in/out)	Enabled	Enabled	Enabled	No default. Enable/disable should be set by the user.
PTP Monitor	Disabled	Disabled	Disabled	No default. Enable/disable should be set by the user.
SyncE	Disabled	Disabled	Enabled	No default. Enable/disable should be set by the user.
Servo	Disabled	Disabled	Disabled	No default. Enable/disable should be set by the user.

1. Client-only is only relevant to a single PTP interface. If more than one PTP interface is provided in the YAML file, both modes are enabled. [___](#)

Outputs

Container Output

While running, the full output of the DOCA Firefly Service container can be viewed using the following command:

```
sudo crictl logs <CONTAINER-ID>
```

Where CONTAINER-ID can be retrieved using the following command:

```
sudo crictl ps
```

For example, in the following output, the container ID is 8f368b98d025b.

```
$ sudo crictl ps
CONTAINER      IMAGE          CREATED        STATE         NAME
ATTEMPT       POD ID        POD
8f368b98d025b 289809f312b4c 2 seconds ago Running       doca-firefly
0              5af59511b4be4 doca-firefly-some-computer-name
```

The output of the container depends on the services supported by the hardware and enabled by configuration and the selected profile. However, note that any of the configurations runs PTP, so when DOCA FireFly is running successfully expect to see the line "Running ptp4l".

The following is an example of the expected container output when running the default profile on a DPU that supports PPS:

```
2023-09-07 14:04:23 - Firefly - Init - INFO - Starting DOCA Firefly - Version 1.4.0
2023-09-07 14:04:23 - Firefly - Init - INFO - Selected features:
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] PTP - Enabled - ptp4l will be used
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] MONITOR - Enabled - PTP Monitor will
be used
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] PHC2SYS - Enabled - phc2sys will be
used
2023-09-07 14:04:23 - Firefly - Init - INFO - [-] SyncE - Disabled
2023-09-07 14:04:23 - Firefly - Init - INFO - [-] SERVO - Disabled
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] PPS - Enabled - testptp will be used (if
supported by hardware)
2023-09-07 14:04:23 - Firefly - Init - INFO - Going to analyze the configuration files
2023-09-07 14:04:23 - Firefly - Init - INFO - Requested the following PTP interface: p0
```

```
2023-09-07 14:04:23 - Firefly
2023-09-07 14:04:23 - Firefly - Init - INFO - Starting PPS configuration
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] PPS is supported by hardware
2023-09-07 14:04:23 - Firefly - Init - INFO - set pin function okay
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] PPS in - Activated
2023-09-07 14:04:23 - Firefly - Init - INFO - set pin function okay
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] PPS out - Activated
2023-09-07 14:04:23 - Firefly - Init - INFO - name mlx5_pps0 index 0 func 1 chan 0
2023-09-07 14:04:23 - Firefly - Init - INFO - name mlx5_pps1 index 1 func 2 chan 0
2023-09-07 14:04:23 - Firefly - Init - INFO - periodic output request okay
2023-09-07 14:04:23 - Firefly
2023-09-07 14:04:23 - Firefly - Init - INFO - Running ptp4l
2023-09-07 14:04:23 - Firefly - Init - INFO - Running Firefly PTP Monitor
2023-09-07 14:04:23 - Firefly - Init - INFO - Running phc2sys
```

The following is an example of the expected container output when running the default profile on a DPU that does not support PPS:

```
2023-09-07 14:04:23 - Firefly - Init - INFO - Starting DOCA Firefly - Version 1.3.0
2023-09-07 14:04:23 - Firefly - Init - INFO - Selected features:
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] PTP - Enabled - ptp4l will be used
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] MONITOR - Enabled - PTP Monitor will
be used
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] PHC2SYS - Enabled - phc2sys will be
used
2023-09-07 14:04:23 - Firefly - Init - INFO - [-] SyncE - Disabled
2023-09-07 14:04:23 - Firefly - Init - INFO - [-] SERVO - Disabled
2023-09-07 14:04:23 - Firefly - Init - INFO - [+] PPS - Enabled - testptp will be used (if
supported by hardware)
2023-09-07 14:04:23 - Firefly - Init - INFO - Going to analyze the configuration files
2023-09-07 14:04:23 - Firefly - Init - INFO - Requested the following PTP interface: p0
2023-09-07 14:04:23 - Firefly
2023-09-07 14:04:23 - Firefly - Init - INFO - Starting PPS configuration
2023-09-07 14:04:23 - Firefly - Init - WARNING - [-] PPS capability is missing, seems
that the card doesn't support PPS
```

```
2023-09-07 14:04:23 - Firefly - Init - INFO - capabilities:
2023-09-07 14:04:23 - Firefly - Init - INFO - 50000000 maximum frequency
adjustment (ppb)
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 programmable alarms
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 external time stamp channels
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 programmable periodic signals
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 pulse per second
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 programmable pins
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 cross timestamping
2023-09-07 14:04:23 - Firefly
2023-09-07 14:04:23 - Firefly - Init - INFO - Running ptp4l
2023-09-07 14:04:23 - Firefly - Init - INFO - Running Firefly PTP Monitor
2023-09-07 14:04:23 - Firefly - Init - INFO - Running phc2sys
```

Firefly Output

On top of the container's log, Firefly defines an additional, non-volatile log that can be found in `/var/log/doca/firefly/firefly.log`.

This file contains the same output described in section "[Container Output](#)" and is useful for debugging deployment errors should the container stop its execution.

Note

To avoid disk space issues, the `/var/log/doca/firefly/firefly.log` file only contains the log from Firefly's initialization, and not the logs of the rest of the modules (ptp4l, phc2sys, etc.) or that of the PTP monitor. The latter is still included in the container log and can be inspected using the command `sudo crictl logs <CONTAINER-ID>`.

ptp4l Output

The ptp4l output can be found in the file `/var/log/doca/firefly/ptp4l.log`.

Example output:

```
ptp4l[192710.691]: rms 1 max 1 freq -114506 +/- 0 delay -15 +/- 0
ptp4l[192712.692]: rms 6 max 9 freq -114501 +/- 3 delay -15 +/- 0
ptp4l[192714.692]: rms 7 max 9 freq -114511 +/- 3 delay -13 +/- 0
ptp4l[192716.692]: rms 5 max 7 freq -114502 +/- 1 delay -13 +/- 0
ptp4l[192718.693]: rms 4 max 6 freq -114509 +/- 2 delay -13 +/- 0
ptp4l[192720.693]: rms 3 max 3 freq -114506 +/- 2 delay -13 +/- 0
ptp4l[192722.694]: rms 4 max 6 freq -114510 +/- 3 delay -12 +/- 0
ptp4l[192724.694]: rms 5 max 7 freq -114510 +/- 5 delay -12 +/- 1
ptp4l[192726.695]: rms 4 max 5 freq -114508 +/- 3 delay -11 +/- 0
ptp4l[192728.695]: rms 6 max 9 freq -114504 +/- 4 delay -11 +/- 0
```

phc2sys Output

The phc2sys output can be found in the file `/var/log/doca/firefly/phc2sys.log`.

Example output:

```
phc2sys[1873325.928]: reconfiguring after port state change
phc2sys[1873325.928]: selecting CLOCK_REALTIME for synchronization
phc2sys[1873325.928]: selecting enp3s0f0s4 as the master clock
phc2sys[1873325.928]: CLOCK_REALTIME phc offset 1378 s2 freq -165051 delay 255
phc2sys[1873326.928]: CLOCK_REALTIME phc offset 1378 s2 freq -163673 delay 240
phc2sys[1873327.928]: port 62b785.ffe.0c9369-1 changed state
phc2sys[1873327.929]: CLOCK_REALTIME phc offset 14 s2 freq -164624 delay 255
phc2sys[1873328.936]: CLOCK_REALTIME phc offset 89 s2 freq -164545 delay 240
```

SyncE Output

The SyncE output can be found in the file `/var/log/doca/firefly/synced.log`.

Example output:

```
INFO [05/09/2023 05:11:01.493414]: SyncE Group #0: is in TRACKING holdover
acquired mode on p0, frequency_diff: 0 (ppb)
INFO [05/09/2023 05:11:02.502963]: SyncE Group #0: is in TRACKING holdover
acquired mode on p0, frequency_diff: -113 (ppb)
INFO [05/09/2023 05:11:03.512491]: SyncE Group #0: is in TRACKING holdover
acquired mode on p0, frequency_diff: 37 (ppb)
```

Note

The verbosity of the output from the SYNCE module is limited by default. To set the output to be more verbose, set the `verbose` option to 1 (True).

Before:

```
# Example #4 - Overwrite the value of verbose in the [global]
section of the SyncE configuration file.
#- name: CONF_SYNCCE_global_verbose
# value: "1"
```

After:

```
# Example #4 - Overwrite the value of verbose in the [global]
section of the SyncE configuration file.
- name: CONF_SYNCCE_global_verbose
value: "1"
```

Firefly Servo Output

The Firefly servo output can be found in the file `/var/log/doca/firefly/servo.log`.

Example output:

```
2024-03-18 09:04:22 - Firefly - SERVO - INFO - offset +8 +/- 2 freq -5.66 +/- 0.41 delay
-48 +/- 2
2024-03-18 09:04:24 - Firefly - SERVO - INFO - offset +4 +/- 2 freq -6.35 +/- 0.36 delay
-47 +/- 2
2024-03-18 09:04:26 - Firefly - SERVO - INFO - offset +2 +/- 2 freq -6.75 +/- 0.41 delay
-47 +/- 1
2024-03-18 09:04:28 - Firefly - SERVO - INFO - offset +0 +/- 2 freq -6.97 +/- 0.35 delay
-47 +/- 1
2024-03-18 09:04:30 - Firefly - SERVO - INFO - offset +0 +/- 3 freq -7.30 +/- 0.60 delay
-47 +/- 1
2024-03-18 09:04:33 - Firefly - SERVO - INFO - offset +1 +/- 2 freq -6.93 +/- 0.41 delay
-47 +/- 1
2024-03-18 09:04:35 - Firefly - SERVO - INFO - offset +1 +/- 2 freq -6.81 +/- 0.48 delay
-47 +/- 1
2024-03-18 09:04:37 - Firefly - SERVO - INFO - offset +2 +/- 2 freq -6.76 +/- 0.52 delay
-48 +/- 2
```

Tx Timestamping Support on DPU Mode

When the BlueField is operating in DPU mode, additional OVS configuration is required as mentioned in [step 6](#) of section "[Setting Up Network Interfaces for DPU Mode](#)". This configuration achieves the following:

- Proper support for incoming/outgoing multicast traffic
- Enabling Tx timestamping

Firefly only gets the packet timestamping for outgoing PTP messages (Tx timestamping) when they are offloaded to the hardware. As such, when working with OVS, users must ensure this traffic flow is properly recognized and offloaded. If offloading does not take place, Firefly gets stuck in a fault loop while waiting to receive the Tx timestamp events:

```
ptp4l[2912.797]: timed out while polling for tx timestamp
```



```
ptp4l[2912.797]: increasing tx_timestamp_timeout may correct this issue, but it is
likely caused by a driver bug
ptp4l[2912.797]: port 1 (enp3s0f0s4): send sync failed
ptp4l[2923.528]: timed out while polling for tx timestamp
ptp4l[2923.528]: increasing tx_timestamp_timeout may correct this issue, but it is
likely caused by a driver bug
ptp4l[2923.528]: port 1 (enp3s0f0s4): send sync failed
```

The solution to this issue:

- Activation of hardware offloading in OVS
- OpenFlow rules that ensure OVS properly recognizes the traffic and offloads it to the hardware
- Modification to the `fault_reset_interval` configuration value to ensure timely recovery from the fault induced by the first packet being always treated by software (until the rule is offloaded to hardware). As such, Firefly requires that the `fault_reset_interval` value is 1 or less. Proper warnings are raised if an improper value is detected. The value is updated accordingly in the built-in profiles.

When these configurations are in order, Firefly includes a report for a single fault during boot, but recovers from it and continues as usual:

```
ptp4l[3715.687]: timed out while polling for tx timestamp
ptp4l[3715.687]: increasing tx_timestamp_timeout may correct this issue, but it is
likely caused by a driver bug
ptp4l[3715.687]: port 1 (enp3s0f0s4): send delay request failed
```

Troubleshooting Tx Timestamp Issues

As explained earlier, there are several layers required to ensure Tx timestamping works as necessary by Firefly. The following is a list of commands to debug the state of each layer:

1. Inspect the OpenFlow rules:

```
$ sudo ovs-ofctl dump-flows uplink
```

```
cookie=0x0, duration=4075.576s, table=0, n_packets=2437, n_bytes=209582,  
udp,in_port=en3f0pf0sf4,tp_src=319 actions=output:p0  
cookie=0x0, duration=4075.549s, table=0, n_packets=1216, n_bytes=109420,  
udp,in_port=p0,tp_src=319 actions=output:en3f0pf0sf4  
cookie=0x0, duration=4075.521s, table=0, n_packets=13, n_bytes=1242,  
udp,in_port=en3f0pf0sf4,tp_src=320 actions=output:p0  
cookie=0x0, duration=4074.604s, table=0, n_packets=3034, n_bytes=297376,  
udp,in_port=p0,tp_src=320 actions=output:en3f0pf0sf4  
cookie=0x0, duration=4075.856s, table=0, n_packets=184, n_bytes=12901,  
priority=0 actions=NORMAL
```

2. Inspect hardware TC rules while DOCA Firefly is deployed (the rules age out after 10 seconds without traffic):

```
$ sudo tc -s -d filter show dev en3f0pf0sf4 egress  
filter ingress protocol ip pref 4 flower chain 0  
filter ingress protocol ip pref 4 flower chain 0 handle 0x1  
eth_type ipv4  
ip_proto udp  
src_port 320  
ip_flags nofrag  
in_hw in_hw_count 1  
action order 1: mirred (Egress Redirect to device p0) stolen  
index 3 ref 1 bind 1 installed 7 sec used 7 sec  
Action statistics:  
Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)  
backlog 0b 0p requeues 0  
cookie bec8bd6ede4e86341e9045a6edb58ca2  
no_percpu  
  
filter ingress protocol ip pref 4 flower chain 0 handle 0x2  
eth_type ipv4  
ip_proto udp  
src_port 319  
ip_flags nofrag
```

```
in_hw in_hw_count 1
action order 1: mirrored (Egress Redirect to device p0) stolen
index 4 ref 1 bind 1 installed 6 sec used 6 sec
Action statistics:
Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
cookie c568d97efd400de98608fbbf86ccdf3c
no_percpu
```

Note

If no TC rules are present when Firefly is running, this usually indicates that hardware offloading is disabled at the OVS level, in which case it should be activated as explained under "[Ensuring OVS Hardware Offload](#)".

PTP

Firefly uses the `ptp4l` utility to handle the Precision Time Protocol (IEEE 1588).

Through the YAML file, users can configure the network interfaces used for the protocol:

```
# Network interfaces to be used (For multiple interfaces use a space (" ") separated
list)
- name: PTP_INTERFACE
# Set according to used interfaces on the local setup
value: "p0"
```

Before the deployment of the container, users should configure this field to point at the desired network interface(s) configured in the previous steps.

PHC2SYS

Firefly uses the `phc2sys` utility to synchronize the OS's clock to the accurate time stamps received by `ptp4l`.

Through the YAML file, users can configure the command-line arguments used by the `phc2sys` program:

```
- name: PHC2SYS_ARGS  
value: "-a -r"
```

Firefly adds the following command-line arguments on top of the user-selected flags:

- Use of chosen configuration file (empty configuration file by default, or user-supplied file if specified in the YAML file)
- Redirection of output to a log file using the `-m` command line option

Note

`phc2sys` must use the same `domainNumber` setting used by `ptp4l`. If the same `domainNumber` is not set by the user, Firefly does that automatically.

Note

`phc2sys` is only able to accurately sync the clock of the hosting environment (usually the DPU, but may also be the host if deployed there) if other timing services, such as NTP, are disabled.

So, for instance, on Ubuntu 22.04, users must ensure that the NTP timing service is disabled by running:

```
systemctl stop systemd-timesyncd
```

SYNCE

Note

This feature is supported at beta level.

Firefly uses the proprietary *synced* utility to implement the [Synchronous Ethernet](#) protocol, aimed at ensuring synchronization of the clock's frequency with the reference clock. Once achieved, both clocks are declared as "syntonized".

Through the YAML file, users can configure the network interfaces used for the protocol:

```
# Network interfaces to be used (For multiple interfaces use a space (" ") separated list)
- name: SYNCE_INTERFACE
# Set according to used interfaces on the local setup
value: "p0"
```

Before the deployment of the container, one should configure this field to point at the desired network interface(s) configured in the previous steps.

Linux kernel 6.8 and above include *synced* support for the "dp11" backend (default) which adds support for SFs and VFs. Prior to Linux kernel 6.8, only PFs were supported with the "mft" backend.

The "dp11" backend is the default backend used. If DOCA detects the system does not support it, it will automatically falls back to the "mft" backend. To explicitly set the backend option, one can set it through the YAML file by uncommenting the following lines:

Before	<pre># Example #5 - Explicitly specify the used backend in the [global] section of the SyncE configuration file. #- name: CONF_SYNCE_global_backend # # Options are "mft"/"dpll". If nothing is specified in YAML, "dpll" is taken as the default # value: "mft"</pre>
After	<pre># Example #5 - Explicitly specify the used backend in the [global] section of the SyncE configuration file. - name: CONF_SYNCE_global_backend # Options are "mft"/"dpll". If nothing is specified in YAML, "dpll" is taken as the default value: "mft"</pre>

Note

DOCA Firefly 1.4.0 YAML file explicitly specifies the use of the "mft" backend for SyncE so as to work around a known issue in the BlueField image.

The following is an example for the OVS commands required to route the SyncE-related traffic when using a SF on top of the "dpll" backend:

```
$ sudo ovs-ofctl add-flow uplink
dl_dst=01:80:c2:00:00:02,in_port=en3f0pf0sf4,actions=p0
$ sudo ovs-ofctl add-flow uplink
dl_dst=01:80:c2:00:00:02,in_port=p0,actions=en3f0pf0sf4
$ sudo ovs-ofctl add-flow uplink dl_dst=01:80:c2:00:00:02,actions=controller
```

Info

This example uses the same OVS settings used earlier in the guide:

- uplink – bridge name
- en3f0pf0sf4 – SF representor
- p0 – PF interface we are working (port 0)

If your deployment uses different values make sure to adjust the above commands accordingly.

If the kernel version does not yet support this feature, and SF/VF are used, the following error is printed:

```
...  
mlx5 DPLL kernel support appears to be missing  
Falling back to MFT tools backend  
...
```

If this error is shown, only PFs can be used, and `syncd` falls back to using the "mft" backend.

PTP Monitor

PTP monitor periodically queries for various PTP-related information and prints it to the container's log.

The following is a sample output of this tool:

```
gmIdentity: 48:B0:2D:FF:FE:5C:4D:24 (48b02d.ffe.5c4d24)  
portIdentity: 48:B0:2D:FF:FE:5C:53:44 (48b02d.ffe.5c5344-1)  
port_state: Active  
domainNumber: 2  
master_offset: avg: 1 max: -8 rms: 3
```

```
gmPresent: true
ptp_stable: Recovered
UtcOffset: 37
timeTraceable: 0
frequencyTraceable: 0
grandmasterPriority1: 128
gmClockClass: 248
gmClockAccuracy: 0x6
grandmasterPriority2: 128
gmOffsetScaledLogVariance: 0xffff
ptp_time (TAI): Thu Sep 7 11:22:50 2023
ptp_time (UTC adjusted): Thu Sep 7 11:22:13 2023
system_time (UTC): Thu Sep 7 11:22:13 2023
error_count: 1
last_err_time (UTC): Thu Sep 7 09:55:48 2023
```

Among others, this monitoring provides the following information:

- Details about the Grandmaster the DPU is syncing with
- Current PTP timestamp
- Health information such as connection errors during execution and whether they have been recovered from

PTP monitoring is disabled by default and can be activated by replacing the `disable` value with the IP address for the monitor server to use:

```
- name: MONITOR_STATE
Value: "<IP address for the monitoring server>"
```

Once activated, the information can be viewed from the container using the following command:

```
sudo crictl logs --tail=20 <CONTAINER-ID>
```

It is recommended to use the following `watch` command to actively monitor the PTP state:


```
sudo watch -n 1 crictl logs --tail=20 <CONTAINER-ID>
```

When triaging deployment issues, additional logging information can be found in the monitor's developer logs: `/var/log/doca/firefly/firefly_monitor_dev.log`.

Note

The monitoring feature connects to ptp4l's local UDS server to query the necessary information. This is why the configuration manager prevents users from modifying the `uds_address` and `uds_ro_address` fields used by ptp4l within the container.

Configuration

The PTP monitor supports configuration options which are passed through a dedicated configuration file like the rest of DOCA Firefly's modules. The built-in monitor configuration file can be found in the section "[PTP Monitor](#)". For ease of use, the file is also provided as part of DOCA's container resource as downloaded from NGC.

"[Firefly Modules Configuration Options](#)" contains a complete explanation of each of the configuration options alongside their default values.

To set a custom config file, users should locate their config file in the directory `/etc/firefly` and set the config file name in DOCA Firefly's YAML file.

```
- name: MONITOR_CONFIG_FILE  
  value: my_custom_monitor.conf
```

In this example, `my_custom_monitor.conf` should be placed at `/etc/firefly/my_custom_monitor.conf`.

Time Representations (PTP Time vs System Time)

Under most deployment scenarios, the PTP time shown by the monitor is presented according to the International Atomic Time (TAI) standard, while the system time would most commonly use the Coordinated Universal Time (UTC). Due to the differences between these time representation models, the monitor provides 2 different time readings (each marked accordingly):

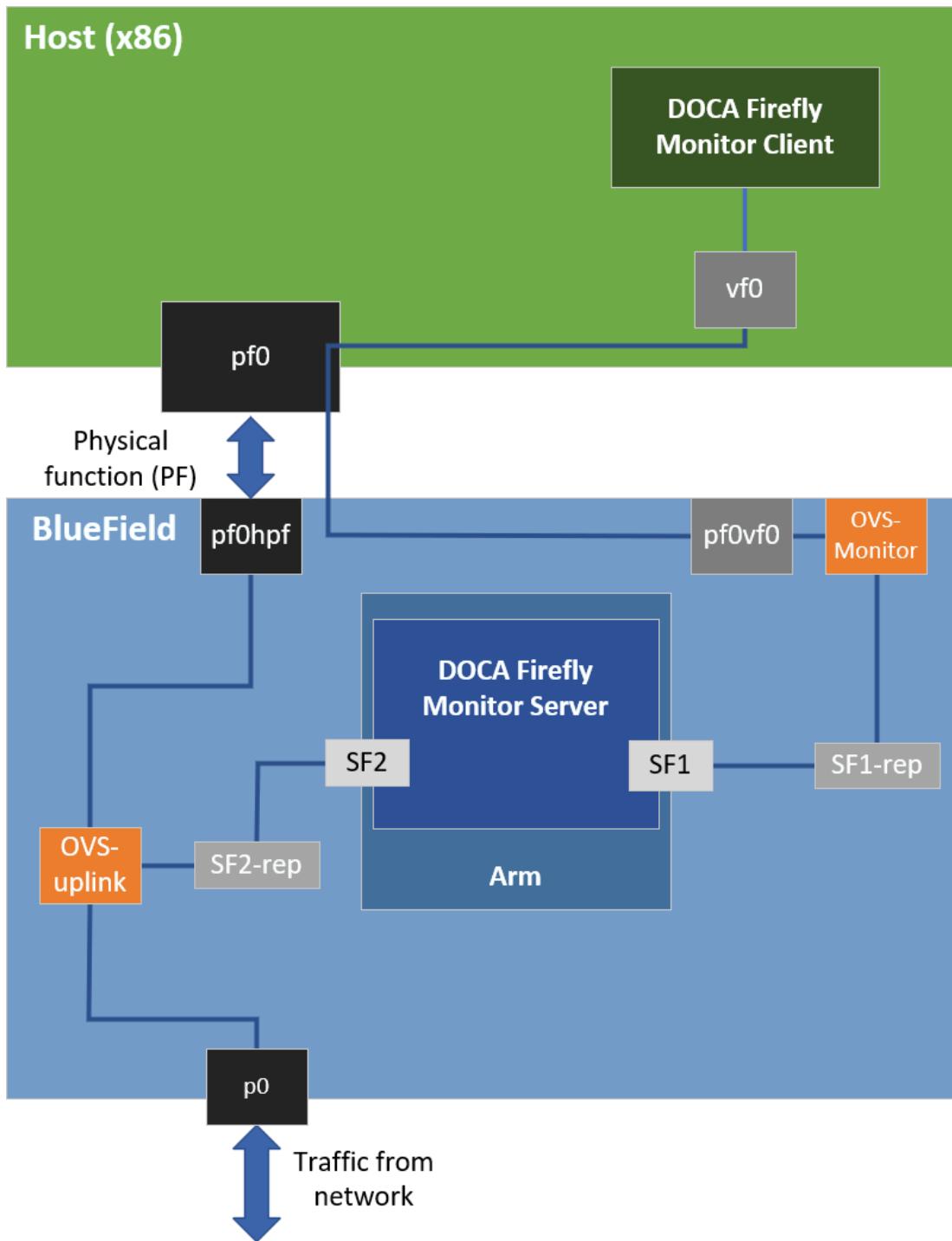
```
...
UtcOffset: 37
...
ptp_time (TAI): Thu Sep 7 11:22:50 2023
ptp_time (UTC adjusted): Thu Sep 7 11:22:13 2023
system_time (UTC): Thu Sep 7 11:22:13 2023
```

This difference (37 seconds in the above example) is intentional and stems from the amount of leap seconds since epoch. This is indicated by the `UtcOffset` field that is also included in the monitor's report.

Monitor Server

In addition to printing the monitoring data to the container's standard output available through the container logs, the monitoring data is also exposed through a gRPC server that clients can subscribe to. This allows a monitoring client on the host to subscribe to monitor events from the service running on top of the DPU, thus providing better visibility.

The following diagram presents the recommended deployment architecture for connecting the monitoring client (on the host) to the monitor server (on the DPU).



Based on the above, when activating the monitor feature, the user must provide the IP address to be used by the monitor server:

```
- name: MONITOR_STATE
value: "<IP address for the monitoring server>"
```

Users can choose to only view the monitoring events through the container logs without connecting to the monitoring server. In this case, it is recommended to configure the local host IP address (127.0.0.1) in the YAML file to avoid exposing it to an unwanted network.

Monitor Client

The required files for the monitor client are available under the service's dedicated NGC resource "scripts" directory.

Example command line for executing the python-based monitor client from a Linux host:

```
$ export PYTHONPATH=${PYTHONPATH}:/opt/mellanox/grpc/python3/lib
$ ./doca_firefly_monitor_client.py <ip-address-for-the-monitoring-server>
```

Note

Reference source files and the .proto file used for Firefly's monitor are placed under the `src/` within the NGC resource.

Firefly Servo

Firefly's Servo module can be seen as an extension to the built-in set of servos offered by `linuxptp`. When active, `linuxptp` is automatically set to "free running" and the control over the physical hardware clock (PHC) is handed over to Firefly's own servo.

The following is a sample output of this tool when using the `I2-telco` profile (16 messages per seconds):

2024-03-18 07:46:45 - Firefly - SERVO - INFO - Detected new master clock:
48b02d.ffffe.5c4d24-1

2024-03-18 07:46:45 - Firefly - SERVO - INFO - Transition from servo state IDLE to
FREE_RUNNING

2024-03-18 07:46:47 - Firefly - SERVO - INFO - Estimated a logSyncInterval of: -4

2024-03-18 07:46:47 - Firefly - SERVO - INFO - Measured offset 18691 delay -47

2024-03-18 07:46:48 - Firefly - SERVO - INFO - Transition from servo state
FREE_RUNNING to LOCKED

2024-03-18 07:46:50 - Firefly - SERVO - INFO - offset +164 +/- 164 freq -1.50 +/- 0.00 delay
-48 +/- 1

2024-03-18 07:46:52 - Firefly - SERVO - INFO - Transition from servo state LOCKED to
LOCKED_STABLE

2024-03-18 07:46:52 - Firefly - SERVO - INFO - offset +0 +/- 1 freq -1.41 +/- 0.47 delay -48
+/- 1

2024-03-18 07:46:54 - Firefly - SERVO - INFO - offset -8 +/- 4 freq -4.21 +/- 1.40 delay -47 +/-
1

2024-03-18 07:46:57 - Firefly - SERVO - INFO - offset -12 +/- 2 freq -5.46 +/- 0.73 delay -47
+/- 1

2024-03-18 07:46:59 - Firefly - SERVO - INFO - offset -13 +/- 2 freq -6.13 +/- 0.65 delay -47
+/- 1

2024-03-18 07:47:01 - Firefly - SERVO - INFO - offset -13 +/- 3 freq -6.19 +/- 1.23 delay -47
+/- 2

2024-03-18 07:47:03 - Firefly - SERVO - INFO - offset -19 +/- 2 freq -8.04 +/- 0.96 delay -47
+/- 1

2024-03-18 07:47:06 - Firefly - SERVO - INFO - offset -14 +/- 3 freq -6.46 +/- 1.11 delay -47
+/- 1

2024-03-18 07:47:08 - Firefly - SERVO - INFO - offset -16 +/- 2 freq -7.32 +/- 0.78 delay -48
+/- 2

2024-03-18 07:47:10 - Firefly - SERVO - INFO - offset -15 +/- 2 freq -7.11 +/- 0.87 delay -47
+/- 2

2024-03-18 07:47:12 - Firefly - SERVO - INFO - offset -14 +/- 1 freq -6.74 +/- 0.57 delay -47
+/- 2

2024-03-18 07:47:15 - Firefly - SERVO - INFO - offset -12 +/- 3 freq -6.20 +/- 1.01 delay -48
+/- 1

```
2024-03-18 07:47:17 - Firefly - SERVO - INFO - offset -13 +/- 2 freq -6.40 +/- 0.89 delay -47 +/- 1
```

```
2024-03-18 07:47:19 - Firefly - SERVO - INFO - offset -11 +/- 2 freq -5.98 +/- 0.86 delay -48 +/- 1
```

```
2024-03-18 07:47:21 - Firefly - SERVO - INFO - offset -10 +/- 2 freq -5.75 +/- 0.87 delay -46 +/- 1
```

```
2024-03-18 07:47:24 - Firefly - SERVO - INFO - offset -8 +/- 1 freq -5.15 +/- 0.42 delay -47 +/- 1
```

As can be seen, the servo's behavior is similar to that of linuxptp's ptp4l and consists of a state machine that tracks the state of the active PTP port (FREE_RUNNING, LOCKED, LOCKED_STABLE, etc).

Firefly's Servo is disabled by default (in all profiles) and can be activated by replacing the `define_by_profile` value with `enable`:

```
# Activation status
- name: SERVO_STATE
# Options are "enable"/"disable"/"defined_by_profile"
value: "enable"
```

Once activated, the information can be viewed from the module's log file `/var/log/doca/firefly/servo.log`.

Firefly Servo Configuration

Firefly's Servo is currently aimed for telco-related deployments, using the `l2-telco` profile including the use of SyncE. As such, the default values in the built-in configuration file are optimized for those scenarios.

The servo supports configuration options which are passed through a dedicated configuration file like the rest of DOCA Firefly's modules. The built-in servo configuration file can be found in the section "[Firefly Servo](#)". For ease of use, the file is also provided as part of DOCA's container resource as downloaded from NGC.

"[Firefly Modules Configuration Options](#)" contains a complete explanation of each of the configuration options alongside their default values.

To set a custom config file, users should locate their config file in the directory `/etc/firefly` and set the config file name in DOCA Firefly's YAML file.

```
- name: SERVO_CONFIG_FILE
  value: my_custom_servo.conf
```

In this example, `my_custom_servo.conf` should be placed at `/etc/firefly/my_custom_servo.conf`.

Dynamic Packet Rate Support

The servo has the ability to dynamically detect the packet rate used by the PTP grandmaster clock, so to calibrate itself accordingly incase it differs from the recommended 16 packets per seconds.

```
2024-03-18 07:46:45 - Firefly - SERVO - INFO - Transition from servo state IDLE to
FREE_RUNNING
2024-03-18 07:46:47 - Firefly - SERVO - INFO - Estimated a logSyncInterval of: -4
2024-03-18 07:46:47 - Firefly - SERVO - INFO - Measured offset 18691 delay -47
```

In a case the message rate is constant and known in advance, the dynamic estimation can be disabled, in favour of a provided message rate:

```
- name: CONF_SERVO_global_servo_const_log_sync_interval
  value: "-2"
```

In the above example, a fixed message rate of 4 packets per seconds will be used (logSyncInterval of "-2").

Note

While the servo was tested to produce stable results with various packets rates (2, 4, 8, 16, 32, 64, 128), it is only officially

recommended for use in deployments using a packet rate of 16 packets per second.

VLAN Tagging

DOCA Firefly natively supports VLAN-tagging-enabled network interfaces.

Separated Mode

The name of the VLAN-enabled network interface should be the one passed through the YAML file in the `PTP_INTERFACE` field.

Embedded Mode

In addition to passing on the VLAN-enabled interface through the YAML as listed in the previous section, the user is also required to configure the network routing within the DPU to support the VLAN tagging:

1. The following example configures a VLAN tag of 10 to the `enp3s0f0s4` interface:

```
$ sudo ip link add link enp3s0f0s4 name enp3s0f0s4.10 type vlan id 10
$ sudo ip link set up enp3s0f0s4.10
$ sudo ifconfig enp3s0f0s4.10 192.168.104.1 up
```

In this example, `enp3s0f0s4.10` is the interface to be passed to DOCA Firefly.

2. Additional commands to route the traffic within the DPU:

```
$ sudo ovs-ofctl add-flow uplink
in_port=en3f0pf0sf4,dl_vlan=10,actions=output:p0
$ sudo ovs-ofctl add-flow uplink
in_port=p0,dl_vlan=10,actions=output:en3f0pf0sf4
```


Multiple Interfaces

DOCA Firefly can support multiple network interfaces through the following YAML file syntax:

```
- name: PTP_INTERFACE  
value: "<space ( ' ') separated list of interface names>"
```

For example:

```
- name: PTP_INTERFACE  
value: "p0 p1"
```

Note

The monitoring feature is supported for multiple interfaces only when the `clientOnly` configuration is enabled.

Note

Automatic mode (-a) for `phc2sys` is not supported when working with multiple interfaces. It is recommended to disable `phc2sys` in this mode.

Troubleshooting

When troubleshooting container deployment issues, it is highly recommended to follow the deployment steps and tips in the "Review Container Deployment" section of the [NVIDIA DOCA Container Deployment Guide](#).

To debug the finalized configuration file used by Firefly, users can connect to the container as follows:

1. Open a shell session on the running container using the container ID:

```
sudo crictl exec -it <container-id> /bin/bash
```

2. Once connected to the container, the finalized configuration file can be found under the `/tmp` directory using the same filename as the original configuration file.

Info

More information regarding the configuration files can be found under section "[Ensuring and Debugging Correctness of Config File](#)".

Pod is Marked as "Ready" and No Container is Listed

Error

When deploying the container, the pod's STATE is marked as Ready, an image is listed, however no container can be seen running:

```
$ sudo crictl pods
POD ID CREATED STATE NAME NAMESPACE ATTEMPT RUNTIME
06bd84c07537e 4 seconds ago Ready doca-firefly-my-dpu default 0 (default)

$ sudo crictl images
IMAGE TAG IMAGE ID SIZE
k8s.gcr.io/pause 3.2 2a060e2e7101d 251kB
```

```
nvcr.io/nvidia/doca/doca_firefly 1.1.0-doca2.0.2 134cb22f34611 87.4MB
```

```
$ sudo crictl ps
```

```
CONTAINER IMAGE CREATED STATE NAME ATTEMPT POD ID POD
```

Solution

In most cases, the container did start, but immediately exited. This could be checked using the following command:

```
$ sudo crictl ps -a
```

```
CONTAINER IMAGE CREATED STATE NAME ATTEMPT POD ID POD
```

```
556bb78281e1d 134cb22f34611 7 seconds ago Exited doca-firefly 1
```

```
06bd84c07537e doca-firefly-my-dpu
```

Should the container fail (i.e., state of Exited) it is recommended to examine Firefly's main log at `/var/log/doca/firefly/firefly.log`.

In addition, for a short period of time after termination, the container logs could also be viewed using the the container's ID:

```
$ sudo crictl logs 556bb78281e1d
```

```
Starting DOCA Firefly - Version 1.1.0
```

```
...
```

```
Requested the following PTP interface: p10
```

```
Failed to find interface "p10". Aborting
```

Custom Config File is Not Found

Error

When DOCA Firefly is deployed using a custom configuration file, a deployment error occurs and the following log message appears:

```
...
2023-09-07 14:04:23 - Firefly - Init - ERROR - Custom config file not found:
my_file.conf. Aborting
...
```

Solution

Check the custom file name written in the YAML file and make sure that you properly placed the file with that name under the `/etc/firefly/` directory of the DPU.

Profile is Not Supported

Error

When DOCA Firefly is deployed, a deployment error occurs and the following log message appears:

```
...
2023-09-07 14:04:23 - Firefly - Init - ERROR - profile <name> is not supported.
Aborting
...
```

Solution

Verify that the profile selected in the YAML file matches one of the supported profiles as listed in the [profiles table](#).

Note

The profile name is case sensitive. The name must be specified in lower-case letters.

PPS Capability is Missing

Error

When DOCA Firefly is deployed and configured to use the PPS module, a deployment error occurs and the following log message appears:

```
...
2023-09-07 14:04:23 - Firefly - Init - INFO - Starting PPS configuration
2023-09-07 14:04:23 - Firefly - Init - WARNING - [-] PPS capability is missing, seems
that the card doesn't support PPS
2023-09-07 14:04:23 - Firefly - Init - INFO - capabilities:
2023-09-07 14:04:23 - Firefly - Init - INFO - 50000000 maximum frequency
adjustment (ppb)
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 programmable alarms
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 external time stamp channels
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 programmable periodic signals
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 pulse per second
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 programmable pins
2023-09-07 14:04:23 - Firefly - Init - INFO - 0 cross timestamping
...
```

Solution

This log indicates that the DPU hardware does not support PPS. However, PTP can still run on this hardware and you should see the line `Running ptp4l` in the container log, indicating that PTP is running successfully.

Timed Out While Polling for Tx Timestamp

Error

When the BlueField is operating in DPU mode, DOCA Firefly gets stuck in a fault loop while waiting to receive the Tx timestamp events:

```
ptp4l[2912.797]: timed out while polling for tx timestamp
ptp4l[2912.797]: increasing tx_timestamp_timeout may correct this issue, but it is
likely caused by a driver bug
ptp4l[2912.797]: port 1 (enp3s0f0s4): send sync failed
ptp4l[2923.528]: timed out while polling for tx timestamp
ptp4l[2923.528]: increasing tx_timestamp_timeout may correct this issue, but it is
likely caused by a driver bug
ptp4l[2923.528]: port 1 (enp3s0f0s4): send sync failed
```

Info

DOCA Firefly has a known gap leading to this error appearing once, after which ptp4l recovers from it. This section only covers the case in which there is a fault loop and no recovery occurs.

Solution

DOCA Firefly's configurations were already adjusted to accommodate for Tx port timestamping. For more information about the reason for this error and for the designed recovery mechanism from it, refer to section "[Tx Timestamping Support on DPU Mode](#)".

Warning – Time Jumped Backwards

Error

When using Firefly's Servo module, the following warning log message is encountered on start:

```
2024-01-01 14:04:23 - Firefly - SERVO - WARNING - Clock is going to jump backwards in time - this might have a system-wide impact
```

Solution

This warning message indicates that the system's time jumped backwards with a value of at least one minute. This event is logged by Firefly given that such jumps might have system-wide implications. For more information, refer to section "[Failed to Reserve Sandbox Name](#)" in the [NVIDIA DOCA Troubleshooting Guide](#).

Such jumps can only happen during Firefly's boot, before the Servo achieves initial time synchronization with the reference clock.

PTP Profile Default Config Files

Media Profile

```
#  
# This config file contains configurations for media & entertainment alongside  
# DOCA Firefly specific adjustments.  
#  
  
[global]  
domainNumber 127  
priority1 128  
priority2 127
```

```
use_syslog 1
logging_level 6
tx_timestamp_timeout 30
hybrid_e2e 1
dscp_event 46
dscp_general 46
logAnnounceInterval -2
announceReceiptTimeout 3
logSyncInterval -3
logMinDelayReqInterval -3
delay_mechanism E2E
network_transport UDPv4
# Value lesser or equal to 1 is required for Embedded Mode
fault_reset_interval 1
# Required for multiple interfaces support
boundary_clock_jbod 1
```

Default Profile

```
#
# This config file extends linuxptp default.cfg config file with DOCA Firefly
# specific adjustments.
#

[global]
# Value lesser or equal to 1 is required for Embedded Mode
fault_reset_interval 1
# Required for multiple interfaces support
boundary_clock_jbod 1
```

Telco (L2) Profile


```

#
# This config file extends linuxptp G.8275.1.cfg config file with DOCA Firefly
# specific adjustments.
#

[global]
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
maxStepsRemoved 255
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
G.8275.portDS.localPriority 128
ptp_dst_mac 01:80:C2:00:00:0E
network_transport L2
domainNumber 24
# Value lesser or equal to 1 is required for Embedded Mode
fault_reset_interval 1
# Required for multiple interfaces support
boundary_clock_jbod 1

```

Firefly Modules Configuration Options

PTP Monitor

monitor-default.conf

```

#
# Default values for all of Firefly's PTP monitor configuration values.
#

[global]
# General
report_interval 1000

```

```
# Debugging & Logging
doca_logging_level 50
```

Configuration Options

- `report_interval` – the time interval (in milliseconds) for when the monitor should publish a report to all defined output providers (standard output, gRPC clients, etc). Default: 1000 (1 second).
- `doca_logging_level` – Logging level for the module, based on DOCA's logging levels. Default is 50 (INFO). Valid options:
 - 10=DISABLE
 - 20=CRITICAL
 - 30=ERROR
 - 40=WARNING
 - 50=INFO
 - 60=DEBUG

Firefly Servo

`servo-default.conf`

```
#
# Default values for all of Firefly's servo configuration values
#

[global]
# Time thresholds
offset_from_master_min_threshold -1500
```

```

offset_from_master_max_threshold 1500
init_max_time_adjustment 0
max_time_adjustment 1500
step_adjustment_threshold 0
hold_over_timer 0
# Sampling Window & servo logic
warmup_period 1500
sync_filter_length 6
delay_request_filter_length 6
servo_adjustment_interval 4
servo_init_adjustment_interval 24
servo_const_log_sync_interval 0xFF
servo_window_min_samples 2
servo_num_offset_values 5
servo_pi_cutoff_frequency 0.0159
servo_pi_dumping_factor 7.85

# Debugging & Logging
summary_interval 2000
doca_logging_level 50
free_running 0

```

Configuration Options

- `offset_from_master_min_threshold` – Minimal threshold (in nanoseconds) for declaring time offset from the master clock as "stable". Default is -1500 (-1.5 microseconds).
- `offset_from_master_max_threshold` – Maximal threshold (in nanoseconds) for declaring time offset from the master clock as "stable". Default is +1500 (+1.5 microseconds).
- `init_max_time_adjustment` – When active, defines the maximal allowed time (step) adjustment (in nanoseconds) before the servo reaches the "locked" state. Default is 0 (disabled).
- `max_time_adjustment` – When active, defines the maximal allowed reference time adjustment (in nanoseconds) after the servo has reached the "locked" state. Default is 1500 (1.5 microseconds).

- `step_adjustment_threshold` – When active, defines the thresholds above which a time (step) adjustment (in nanoseconds) would be allowed, even after the servo has reached the "locked" state. Default is 0 (disabled).
- `hold_over_timer` – When active, defines the time duration (in seconds) in which the servo stays in "hold over" mode, until reverting back to "free running". Default is 0 ("hold over" state is disabled).
- `warmup_period` – Time span (in milliseconds) during which samples are collected to estimate the `logSyncInterval` value (packet rate). Default is 1500 (1.5 seconds).
- `sync_filter_length` – Number of SYNC messages in the servo's history buffer. Default is 6.
- `delay_request_filter_length` – Number of DELAY_REQUEST messages in the servo's history buffer. Default is 6 messages.
- `servo_adjustment_interval` – Number of SYNC messages after which the PHC is updated once the servo has reached the "locked" state at least once. Default is 4 messages.
- `servo_init_adjustment_interval` – Number of SYNC messages after which the PHC is updated before the servo has ever reached the "locked" state. Default is 24 messages.
- `servo_const_log_sync_interval` – Known fixed value to be used as the `logSyncInterval` instead of trying to estimate it at runtime. Default is 0xFF (disabled).
- `servo_window_min_samples` – Minimal number of samples needed for a servo calculation. Default is 2 messages.
- `servo_num_offset_values` – Number of consecutive timestamps within the "offset from master" threshold that are required so to transition from the "locked" state and to the "locked stable" state. Default is 5 offset values.
- `servo_pi_cutoff_frequency` – The PI servo's cutoff frequency value. Default is 0.0159.
- `servo_pi_dumping_factor` – The PI servo's dumping factor value. Default is 7.85.
- `summary_interval` – The time interval (in milliseconds) for when the servo should publish a report log event. Default is 2000 (2 seconds).
- `doca_logging_level` – Logging level for the module, based on DOCA's logging levels. Default is 50 (INFO). Valid options:

- 10=DISABLE
 - 20=CRITICAL
 - 30=ERROR
 - 40=WARNING
 - 50=INFO
 - 60=DEBUG
- `free_running` – Tell the servo to only log the operations, without actually adjusting the PHC. Default is 0 (disabled).

NVIDIA DOCA Flow Inspector Service Guide

This guide provides instructions on how to use the DOCA Flow Inspector service container on top of NVIDIA® BlueField® DPU.

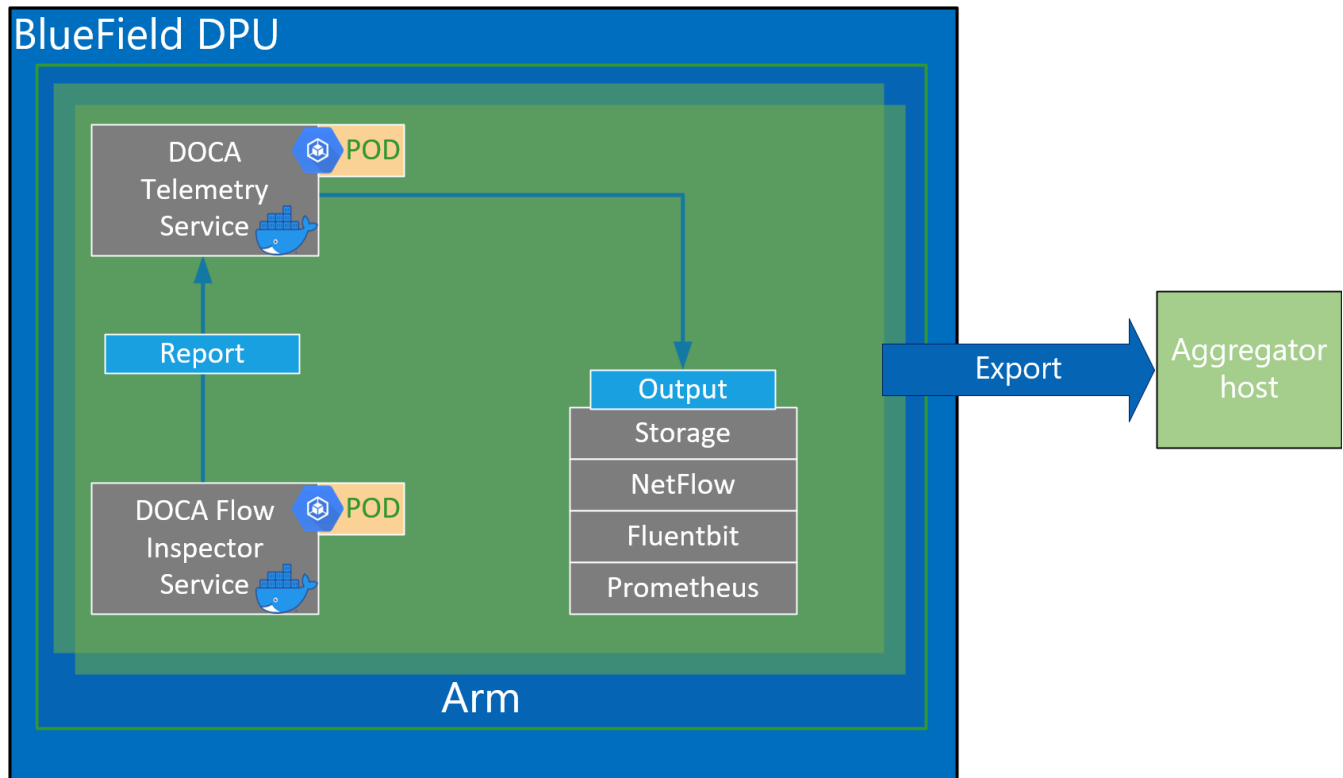
Introduction

DOCA Flow Inspector service enables real-time data monitoring and extraction of telemetry components. These components can be leveraged by various services, including those focused on security, big data, and other purposes.

DOCA Flow Inspector service is linked to DOCA Telemetry Service (DTS). It receives mirrored packets from the user parses the data, and forwards it to the DTS, which aggregates predefined statistics from various providers and sources. The service utilizes the DOCA Telemetry API to communicate with the DTS, while the DPDK infrastructure facilitates packet acquisition at a user-space layer.

DOCA Flow Inspector operates within its dedicated Kubernetes pod on BlueField, aimed at receiving mirrored packets for analysis. The received packets are parsed and

transmitted, in a predefined structure, to a telemetry collector that manages the remaining telemetry aspects.



Service Flow

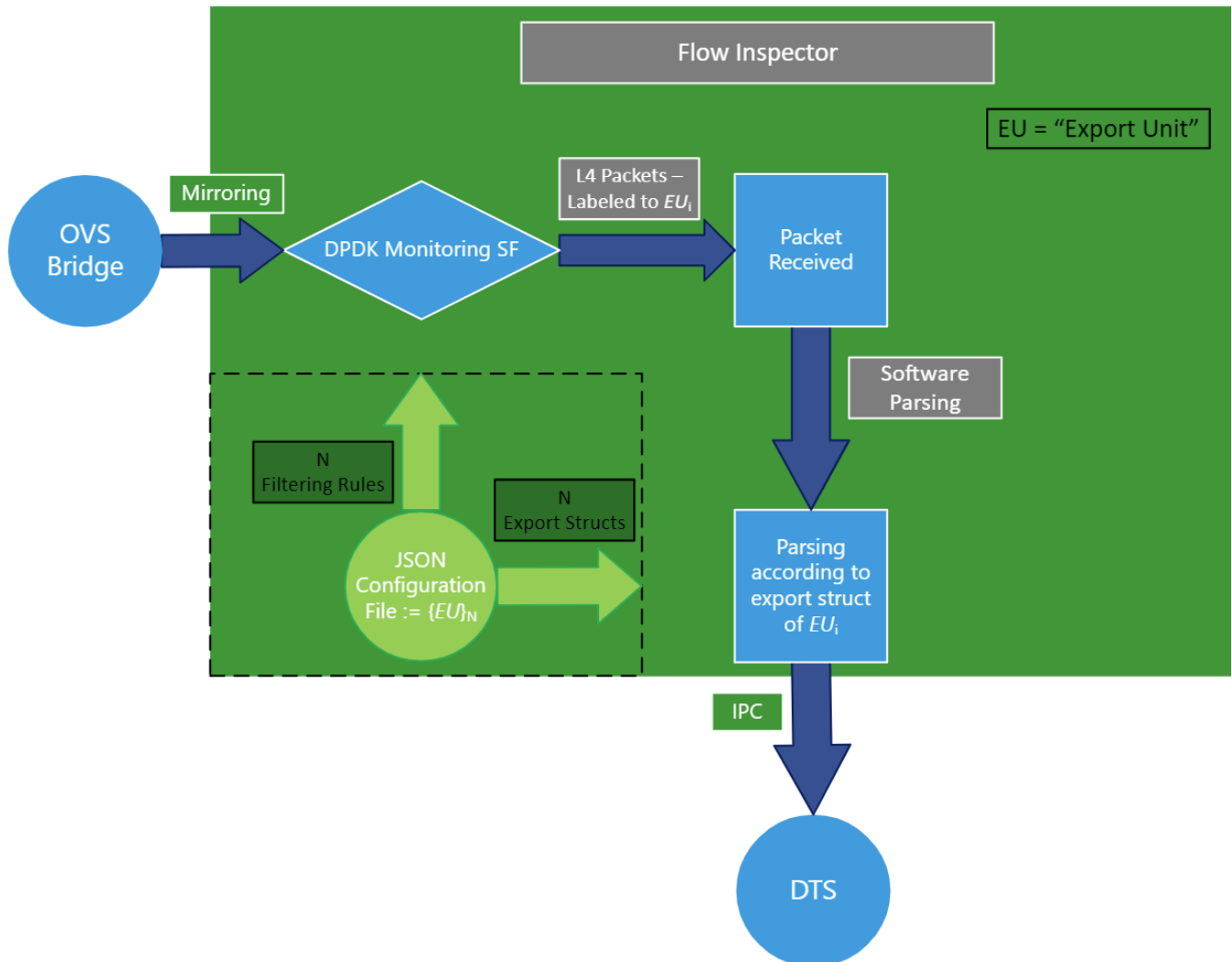
The DOCA Flow Inspector receives a configuration file in a JSON format which includes which of the mirrored packets should be filtered and which information should be sent to DTS for inspection.

The configuration file can include several export units under the "export-units" attribute. Each one is comprised of a "filter" and an "export". Each packet that matches one filter (based on the protocol and ports in the L4 header) is then parsed to the corresponding requested struct defined in the export. That information only is sent for inspection. A packet that does not match any filter is dropped.

In addition, the configuration file could contain FI optional configuration flags, see JSON format and example in the [Configuration](#) section.

The service watches for changes in the JSON configuration file in runtime and for any change that reconfigures the service.

The DOCA Flow Inspector runs on top of DPDK to acquire L4. The packets are then filtered and HW-marked with their export unit index. The packets are then parsed according to their export unit and export struct, and then forwarded to the telemetry collector using IPC.



Configuration phase:

1. A JSON file is used as input to configure the export units (i.e., filters and corresponding export structs).
2. The filters are translated to HW rules on the SF (scalable function port) using the DOCA Flow library.
3. The connection to the telemetry collector is initialized and all export structures are registered to DTS.

Inspection phase:

1. Traffic is mirrored to the relevant SF.
2. Ingress traffic is received through the configured SF.
3. Non-L4 traffic and packets that do not match any filter are dropped using hardware rules.
4. Packets matching a filter are marked with the export unit index they match and are passed to the software layer in the Arm cores.
5. Packets are parsed to the desired struct by the index of export unit.
6. The telemetry information is forwarded to the telemetry agent using IPC.
7. Mirrored packets are freed.
8. If the JSON file is changed, run the configuration phase with the updated file.

Requirements

Before deploying the flow inspector container, ensure that the following prerequisites are satisfied:

1. Create the needed files and directories. Folders should be created automatically. Make sure the `.json` file resides inside the folder:

```
$ touch  
/opt/mellanox/doca/services/flow_inspector/bin/flow_inspector_cfg.json
```

Validate that DTS's configuration folders exist. They should be created automatically when DTS is deployed.

```
$ sudo mkdir -p /opt/mellanox/doca/services/telemetry/config  
$ sudo mkdir -p /opt/mellanox/doca/services/telemetry/ipc_sockets  
$ sudo mkdir -p /opt/mellanox/doca/services/telemetry/data
```


2. Allocate huge pages as needed by DPDK. This requires root privileges.

```
$ sudo echo 2048 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

Or alternatively:

```
$ sudo echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
$ sudo mkdir /mnt/huge
$ sudo mount -t hugetlbfs nodev /mnt/huge
```

Deploy a scalable function according to [NVIDIA BlueField DPU Scalable Function User Guide](#) and mirror packets accordingly using the Open vSwitch command. For example:

1. Mirror packets from p0 to sf4:

```
$ ovs-vsctl add-br ovsbr1
$ ovs-vsctl add-port ovsbr1 p0
$ ovs-vsctl add-port ovsbr1 en3f0pf0sf4
$ ovs-vsctl -- --id=@p1 get port en3f0pf0sf4 \
-- --id=@p2 get port p0 \
-- --id=@m create mirror name=m0 select-dst-port=@p2 select-src-
port=@p2 output-port=@p1 \
-- set bridge ovsbr1 mirrors=@m
```

2. Mirror packets from pf0hpf or p0 that pass through sf4:

```
$ ovs-vsctl add-br ovsbr1
$ ovs-vsctl add-port ovsbr1 pf0hpf
$ ovs-vsctl add-port ovsbr1 p0
$ ovs-vsctl add-port ovsbr1 en3f0pf0sf4
$ ovs-vsctl -- --id=@p1 get port en3f0pf0sf4 \
-- --id=@p2 get port pf0hpf \
```

```
-- --id=@m create mirror name=m0 select-dst-port=@p2 select-src-  
port=@p2 output-port=@p1 \  
-- set bridge ovsbr1 mirrors=@m  
$ ovs-vsctl -- --id=@p1 get port en3f0pf0sf4 \  
-- --id=@p2 get port p0 \  
-- --id=@m create mirror name=m0 select-dst-port=@p2 select-src-  
port=@p2 output-port=@p1 \  
-- set bridge ovsbr1 mirrors=@m
```

The output of last command (creating the mirror) should output a sequence of letters and numbers similar to the following:

```
0d248ca8-66af-427c-b600-af1e286056e1
```

Note

The designated SF must be created as a trusted function. Additional details can be found in the [NVIDIA BlueField DPU Scalable Function User Guide](#).

Service Deployment

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to [NVIDIA DOCA Container Deployment Guide](#).

DTS is available on NGC, NVIDIA's container catalog. Service-specific configuration steps and deployment instructions can be found under the service's [container page](#).

Note

The order of running DTS and DOCA Flow Inspector is important. You must launch DTS, wait a few seconds, and then launch DOCA Flow Inspector.

Configuration

JSON Input

The DOCA Flow Inspector configuration file should be placed under `/opt/mellanox/doca/services/flow_inspector/bin/<json_file_name>.json` and be built in the following format:

```
{
/* Optional param, time period to check for changes in JSON config file (in seconds) and flush telemetry
buffer if enabled (default is 60 seconds) */
"config-sample-rate": <time>,

/* Optional param, telemetry buffer size in bytes (default is 60KB) */
"telemetry-buffer-size": <size>,

/* Optional param, enable periodic telemetry buffer flush and defining the period time (in seconds) */
"telemetry-flush-rate": <numeric value in seconds>,

/* Mandatory param, Flow Inspector export units */
"export-units":
[

/* Export Unit 0 */
{
"filter":
{ "protocols": [<L4 protocols separated by comma>], # What L4 protocols are
allowed
"ports":
[
<source port>, <destination port>],
```

```

[<source ports range>, <destination ports range>],
<... more pairs of source, dest ports>
  ]
},
"export":
{
"fields": [<fields to be part of export struct, separated by comma>] # the Telemetry
event will contain these fields.

}
},
  <... More Export Units>
]
}

```

Export Unit Attributes

Allowed protocols:

- "TCP"
- "UDP"

Port range:

- It is possible to insert a range of ports for both source and destination
- Range should include borders [start_port-end_port]

Allowed ports:

- All ports in range 0-65535 as a string
- Or * to indicate any ports

Allowed fields in export struct:

- timestamp – timestamp indicating when it was received by the service

- host_ip – the IP of the host running the service
- src_mac – source MAC address
- dst_mac – destination MAC address
- src_ip – source IP
- dst_ip – destination IP
- protocol – L4 protocol
- src_port – source port
- dst_port – destination port
- flags – additional flags (relevant to TCP only)
- data_len – data payload length
- data_short – short version of data (payload sliced to first 64 bytes)
- data_medium – medium version of data (payload sliced to first 1500 bytes)
- data_long – long version of data (payload sliced to first 9*1024 bytes)

JSON example:

```
{
  /* Optional param, time period to check for changes in JSON config file (in seconds)
  and flush telemetry buffer if enabled (default is 60 seconds) */
  "config-sample-rate": 30,

  /* Optional param, telemetry maximum buffer size in bytes */
  "telemetry-buffer-size": 70000,

  /* Optional param, enable periodic telemetry buffer flush and defining the period
  time (in seconds) */
  "telemetry-flush-rate": 1.5,
```

```

/* Mandatory param, Flow Inspector export units */
"export-units":
[

/* Export Unit 0 */
{
"filter":
{
"protocols": ["tcp", "udp"],
"ports":
[
["*", "433-460"],
["20480", "28341"],
["28341", "20480"],
["68", "67"],
["67", "68"]
]
},
"export":
{
"fields": ["timestamp", "host_ip", "src_mac", "dst_mac", "src_ip", "dst_ip", "protocol", "src_port",
"dst_port", "flags", "data_len", "data_long"]
}
},

/* Export Unit 1 */
{
"filter":
{
"protocols": ["tcp"],
"ports":
[
["5-10", "422"],
["80", "80"]
]
},

```

```
"export":
{
"fields": ["timestamp","dst_ip", "host_ip", "data_len", "flags", "data_medium"]
}
}
]
}
```

Note

If a packet header contains L4 ports or L4 protocol which are not specified in any filter, they are filtered out.

Yaml File

The .yaml file downloaded from NGC can be easily edited according to your needs.

```
env:
# Set according to the local setup
- name: SF_NUM_1
value: "2" # Additional EAL flags, if needed
- name: EAL_FLAGS
value: "" # Service-Specific command line arguments
- name: SERVICE_ARGS
value: "--policy /flow_inspector/flow_inspector_cfg.json -l 60"
```

- The SF_NUM_1 value can be changed according to the SF used in the OVS configuration and can be found using the command in [NVIDIA BlueField DPU Scalable Function User Guide](#).

- The EAL_FLAGS value must be changed according to the DPDK flags required when running the container.
- The SERVICE_ARGS are the runtime arguments received by the service:
 - -l, --log-level <value> – sets the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
 - -p, --policy <json_path> – sets the JSON path inside the container

Verifying Output

Enabling write to data in the DTS allows debugging the validity of the DOCA Flow Inspector.

To allow DTS to write locally, uncomment the following line in `/opt/mellanox/doca/services/telemetry/config/dts_config.ini`:

```
#output=/data
```

Note

Any changes in `dts_config.ini` necessitate restarting the pod for the new settings to apply.

The schema folder contains JSON-formatted metadata files which allow reading the binary files containing the actual data. The binary files are written according to the naming convention shown in the following example:

Note

Requires installing the tree runtime utility (apt install tree).

```
$ tree /opt/mellanox/doca/services/telemetry/data/  
/opt/mellanox/doca/services/telemetry/data/  
  {year}  
    {mmd}  
    {hash}  
    {source_id}  
      {source_tag}{timestamp}.bin  
    {another_source_id}  
      {another_source_tag}{timestamp}.bin  
  schema  
  schema_{MD5_digest}.json
```

New binary files appear when:

- The service starts
- When the binary file's max age/size restriction is reached
- When JSON file is changed and new schemas of telemetry are created
- An hour passes

If no schema or no data folders are present, refer to the Troubleshooting section in [NVIDIA DOCA Telemetry Service Guide](#).

Note

source_id is usually set to the machine hostname. source_tag is a line describing the collected counters, and it is often set as the provider's name or name of user-counters.

Reading the binary data can be done from within the DTS container using the following command:

```
cricctl exec -it <Container-ID> /opt/mellanox/collectx/bin/clx_read -s /data/schema
/data/path/to/datafile.bin
```

The data written locally should be shown in the following format assuming a packet matching Export Unit 1 from the example has arrived:

```
{
  "timestamp": 1656427771076130,
  "host_ip": "10.237.69.238",
  "src_ip": "11.7.62.4",
  "dst_ip": "11.7.62.5",
  "data_len": 1152,
  "data_short": "Hello World"
}
```

Troubleshooting

When troubleshooting container deployment issues, it is highly recommended to follow the deployment steps and tips in the "Review Container Deployment" section of the [NVIDIA DOCA Container Deployment Guide](#).

Pod is Marked as "Ready" and No Container is Listed

Error

When deploying the container, the pod's STATE is marked as Ready, an image is listed, however no container can be seen running:

```
$ sudo cricctl pods
POD ID CREATED STATE NAME NAMESPACE ATTEMPT RUNTIME
3162b71e67677 4 seconds ago Ready doca-flow-inspector-my-dpu default 0
(default)
```

```
$ sudo crictl images
IMAGE TAG IMAGE ID SIZE
k8s.gcr.io/pause 3.2          2a060e2e7101d 487kB
nvcr.io/nvidia/doca/doca_flow_inspector 1.1.0-doca2.0.2 2af1e539eb7ab 86.8MB

$ sudo crictl ps
CONTAINER IMAGE CREATED STATE NAME ATTEMPT POD ID POD
```

Solution

In most cases, the container did start, but immediately exited. This could be checked using the following command:

```
$ sudo crictl ps -a
CONTAINER IMAGE CREATED STATE NAME ATTEMPT POD ID POD
556bb78281e1d 2af1e539eb7ab 6 seconds ago Exited doca-flow-inspector 1
3162b71e67677 doca-flow-inspector-my-dpu
```

Should the container fail (i.e., state of Exited), it is recommended to examine the Flow Inspector's main log at `/var/log/doca/flow_inspector/flow_inspector_fi_dev.log`.

In addition, for a short period of time after termination, the container logs could also be viewed using the container's ID:

```
$ sudo crictl logs 556bb78281e1d
...
2023-10-04 11:42:55 - flow_inspector - FI - ERROR - JSON file was not found <config-
file-path>.
```

Pod is Not Listed

Error

When placing the container's YAML file in the Kubelet's input folder, the service pod is not listed in the list of pods:

```
$ sudo crictl pods
POD ID CREATED STATE NAME NAMESPACE ATTEMPT RUNTIME
```

Solution

In most cases, the pod does not start due to the absence of the requested hugepages. This can be verified using the following command:

```
$ sudo journalctl -u kubelet -e. . .
Oct 04 12:12:19 <my-dpu> kubelet[2442376]: I1004 12:12:19.905064 2442376
predicate.go:103] "Failed to admit pod, unexpected error while attempting to recover from
admission failure" pod="default/doca-flow-inspector-<my-dpu>" err="preemption: error finding a set
of pods to preempt: no set of running pods found to reclaim resources: [(res: hugepages-2Mi, q:
104563999874), ]"
```

NVIDIA DOCA HBN Service Guide

This guide provides instructions on how to use the DOCA HBN Service container on top of NVIDIA® BlueField® networking platform .

Introduction

Release Notes

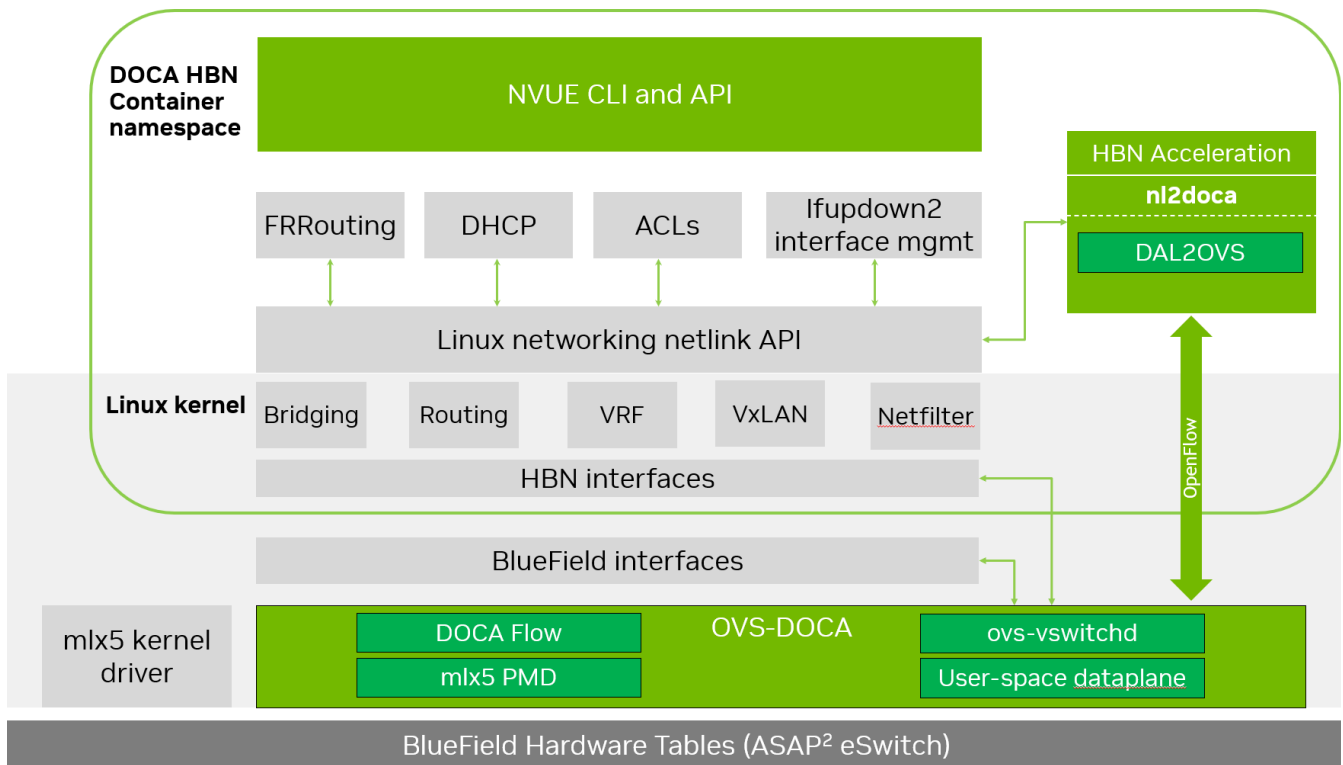
For the release notes of HBN 2.2.0, please refer to "[HBN Service Release Notes](#)".

HBN Overview

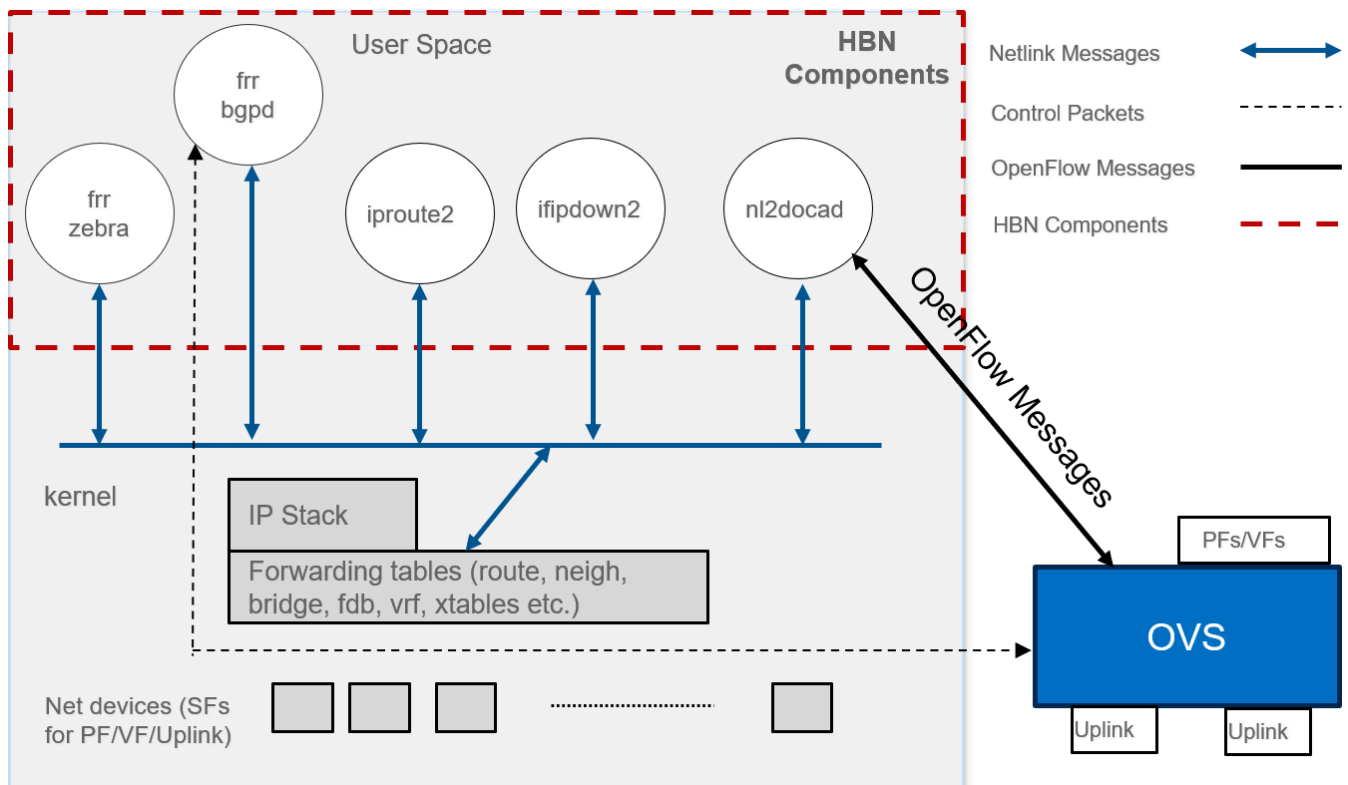
Host-based Networking (HBN) is a DOCA service that enables the network architect to design a network purely on L3 protocols, enabling routing to run on the server-side of the network by using the BlueField as a BGP router. The EVPN extension of BGP, supported by HBN, extends the L3 underlay network to multi-tenant environments with overlay L2 and L3 isolated networks.

The HBN solution packages a set of network functions inside a container which, itself, is packaged as a service pod to be run on BlueField Arm. At the core of HBN is the Linux networking BlueField acceleration driver Netlink-to-DOCA, or nl2docad. This daemon seamlessly accelerates Linux networking using DOCA APIs to program specific packet processing rules in BlueField hardware.

The driver mirrors the Linux kernel routing and bridging tables into the BlueField hardware tables by discovering the configured Linux networking objects using the Linux Netlink API. Dynamic network flows, as learned by the Linux kernel networking stack, are also programmed by the driver into BlueField hardware by listening to Linux kernel networking events.



The following diagram captures an overview of HBN and the interactions between various components of HBN.



- ifupdown2 is the interface manager which pushes all the interface related states to kernel
- The routing stack is implemented in FRR and pushes all the control states (EVPN MACs and routes) to kernel via netlink
- Kernel maintains the whole network state and relays the information using netlink. The kernel is also involved in the punt path and handling traffic that does not match any rules in the eSwitch.
- nl2docad listens for the network state via netlink and invokes the DOCA interface to accelerate the flows in BlueField hardware tables. nl2docad also offloads these flows to eSwitch.

Service Function Chaining

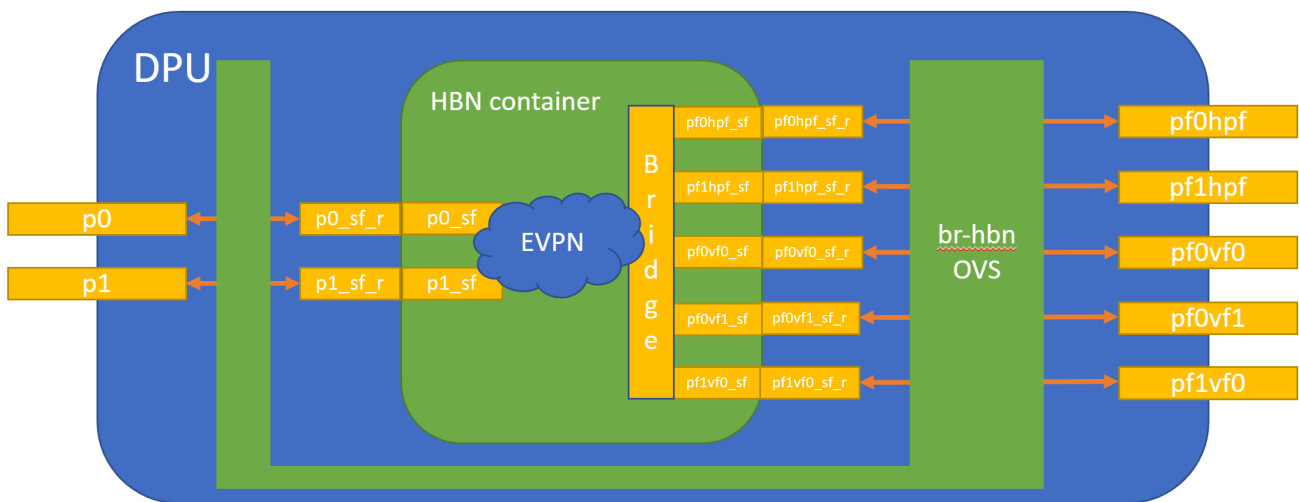
HBN is a "bump-in-the-wire" service and requires specific network configuration on BlueField called service function chaining (SFC). SFC configuration is used to redirect network traffic, which is originated from or forwarded to the host or BlueField itself via the HBN data plane.

The diagram below shows the fully detailed default configuration for HBN with SFC.

In this setup, the HBN container is configured to use sub-function ports (SFs) instead of the actual uplinks, PFs and VFs. To illustrate, for example:

- Uplinks – use p0_sf instead of p0
- PF – use pf0hpf_sf instead of pf0hpf
- VF – use pf0vf0_sf instead of pf0vf0

The indirection layer between the SF and the actual ports is managed via a br-hbn OVS bridge automatically configured when the BFB image is installed on BlueField with HBN enabled. This indirection layer allows other services to be chained to existing SFs and provide additional functionality to transit traffic.



Requirements

i Info

Refer to the "[HBN Service Release Notes](#)" page for information on the specific hardware and software requirements for HBN.

The following subsections describe specific prerequisites for the BlueField before deploying the DOCA HBN Service.

Enabling BlueField DPU Mode

HBN requires BlueField to work in either DPU mode or zero-trust mode of operation. Information about configuring BlueField modes of operation can be found under "[NVIDIA BlueField Modes of Operation](#)".

Enabling SFC

HBN requires SFC configuration to be activated on the BlueField before running the HBN service container. SFC allows for additional services/containers to be chained to HBN and provides additional data manipulation capabilities.

The following subsections provide additional information about SFC and instructions on enabling it during BlueField DOCA image installation.

Deploying BlueField DOCA Image with SFC from Host

For DOCA image installation on BlueField, the user should follow the instructions under [NVIDIA DOCA Installation Guide for Linux](#) with the following extra notes to enable BlueField for HBN setup:

1. Make sure link type is set to ETH under the "Installing Software on Host" section.
2. Add the following parameters to the `bf.cfg` configuration file:
 1. Enable HBN specific OVS bridge on BlueField Arm by setting `ENABLE_BR_HBN=yes`.
 2. Define the uplink ports to be used by HBN `BR_HBN_UPLINKS='<port>'`.

Note

Must include both ports (i.e., p0,p1) for dual-port BlueField devices and only p0 for single-port BlueField devices.

3. Include PF and VF ports to be used by HBN. The following example sets both PFs and 8 VFs on each uplink: `BR_HBN_REPS='pf0hpf,pf1hpf,pf0vf0-pf0vf7,pf1vf0-pf1vf7'`.
4. (Optional) Include SF devices to be created and connected to HBN bridge on the BlueField Arm side by setting `BR_HBN_SFS='pf0dpu1,pf0dpu3'`.

Info

If nothing is provided, `pf0dpu1` and `pf0dpu3` are created by default.

Warning

While older formats of `bf.cfg` still work in this release, they will be deprecated over the next 2 releases. So, its advisable to move to the new format to avoid any upgrade issues in future releases. The following is an example for the old `bf.cfg` format:

```
ENABLE_SFC_HBN=yes
NUM_VFs_PHYS_PORT0=12 # <num VFs supported by HBN
on Physical Port 0> (valid range: 0-127) Default 14
NUM_VFs_PHYS_PORT1=2 # <num VFs supported by HBN
on Physical Port 1> (valid range: 0-127) Default 0
```

3. Then run:

```
bfb-install -c bf.cfg -r rshim0 -b <BFB-image>
```

Deploying BlueField DOCA Image with SFC Using PXE Boot

To enable HBN SFC using a PXE installation environment with BFB content, use the following configuration for PXE:

```
bfnet=<IFNAME>:<IPADDR>:<NETMASK> or <IFNAME>:dhcp  
bfks=<URL of the kickstart script>
```

The kickstart script (bash) should include the following lines:

```
cat >> /etc/bf.cfg << EOF  
  
ENABLE_BR_HBN=yes  
BR_HBN_UPLINKS='p0,p1'  
BR_HBN_REPS='pf0hpf,pf1hpf,pf0vf0-pf0vf7,pf1vf0-pf1vf7'  
BR_HBN_SFS='pf0dpu1,pf0dpu3'  
EOF
```

The `/etc/bf.cfg` generated above is sourced by the BFB `install.sh` script.

Note

It is recommended to verify the accuracy of the BlueField's clock post-installation. This can be done using the following command:

```
$ date
```

Please refer to the known issues listed in the "[NVIDIA DOCA Release Notes](#)" for more information.

Deploying HBN with Other Services

When the HBN container is deployed by itself, BlueField Arm is configured with 3k huge pages. If it is deployed with other services, the actual number of huge-pages must be adjusted based on the requirements of those services. For example, SNAP or NVMesh need approximately 1k huge pages. So if HBN is running with either of these services on the same BlueField, the total number of huge pages must be set to 4k (3k for HBN and 1k for SNAP or NVMesh).

To do that, add the following parameters to the `bf.cfg` configuration file alongside other desired parameters.

```
HUGEPAGE_COUNT=4096
```

Warning

This should be performed only on a BlueField-3 running with 32G of memory. Doing this on 16G system may cause memory issues for various applications on BlueField Arm.

Service Deployment

HBN Service Container Deployment

HBN service is available on [NGC](#), NVIDIA's container catalog. For information about the deployment of DOCA containers on top of the BlueField, refer to [NVIDIA DOCA Container Deployment Guide](#).

Downloading DOCA Container Resource File

Pull the latest DOCA container resource as a *.zip file from NGC and extract it to the <resource> folder (doca_container_configs_2.7.0v1 in this example):

```
wget
https://api.ngc.nvidia.com/v2/resources/nvidia/doca/doca_container_configs/versions/
-O doca_container_configs_2.7.0v1.zip
unzip -o doca_container_configs_2.7.0v1.zip -d doca_container_configs_2.7.0v1
```

Running HBN Preparation Script

The HBN script (hbn-dpu-setup.sh) performs the following steps on BlueField Arm which are required for HBN service to run:

1. Sets the BlueField to DPU mode if needed.
2. Enables IPv4/IPv6 kernel forwarding.
3. Sets up interface MTU if needed.
4. Sets up mount points between BlueField Arm and HBN container for logs and configuration persistency.
5. Sets up various paths as needed by supervisord and other services inside container.

The script is located in <resource>/scripts/doca_hbn/<hbn_version>/ folder, which is downloaded as part of the [DOCA Container Resource](#).

Note

To achieve the desired configuration on HBN's first boot, before running preparation script, users can update default NVUE or flat (network interfaces and FRR) configuration files, which are located in <resource>/scripts/doca_hbn/<hbn_version>/.

- For NVUE-based configuration:

- etc/nvue.d/startup.yaml

- For flat-files based configuration:
 - etc/network/interfaces
 - etc/frr/frr.conf
 - etc/frr/daemons

Run the following commands to execute the hbn-dpu-setup.sh script:

```
cd <resource>/scripts/doca_hbn/2.2.0/  
chmod +x hbn-dpu-setup.sh  
sudo ./hbn-dpu-setup.sh
```

Note

After running the script, perform [BlueField system-level reset](#).

Spawning HBN Container

HBN container .yaml configuration is called doca_hbn.yaml and it is located in <resource>/configs/<doca_version>/ directory. To spawn the HBN container, simply copy the doca_hbn.yaml file to the /etc/kubelet.d directory:

```
cd <resource>/configs/2.7.0/  
sudo cp doca_hbn.yaml /etc/kubelet.d/
```

Kubelet automatically pulls the container image from NGC and spawns a pod executing the container. The DOCA HBN Service starts executing right away.

Verifying HBN Container is Running

To inspect the HBN container and verify if it is running correctly:

1. Check HBN pod and container status and logs:

1. Examine the currently active pods and their IDs (it may take up to 20 seconds for the pod to start):

```
sudo crictl pods
```

2. View currently active containers and their IDs:

```
sudo crictl ps
```

3. Examine logs of a given container:

```
sudo crictl logs
```

4. Examine kubelet logs if something did not work as expected:

```
sudo journalctl -u kubelet@mgmt
```

2. Log into the HBN container:

```
sudo crictl exec -it $(crictl ps | grep hbn | awk '{print $1;}') bash
```

3. While logged into HBN container, verify that the `frr`, `nl2doca`, and `neighmgr` services are running:

```
(hbn-container)$ supervisorctl status frr  
(hbn-container)$ supervisorctl status nl2doca  
(hbn-container)$ supervisorctl status neighmgr
```

4. Users may also examine various logs under `/var/log` inside the HBN container.

HBN Default Deployment Configuration

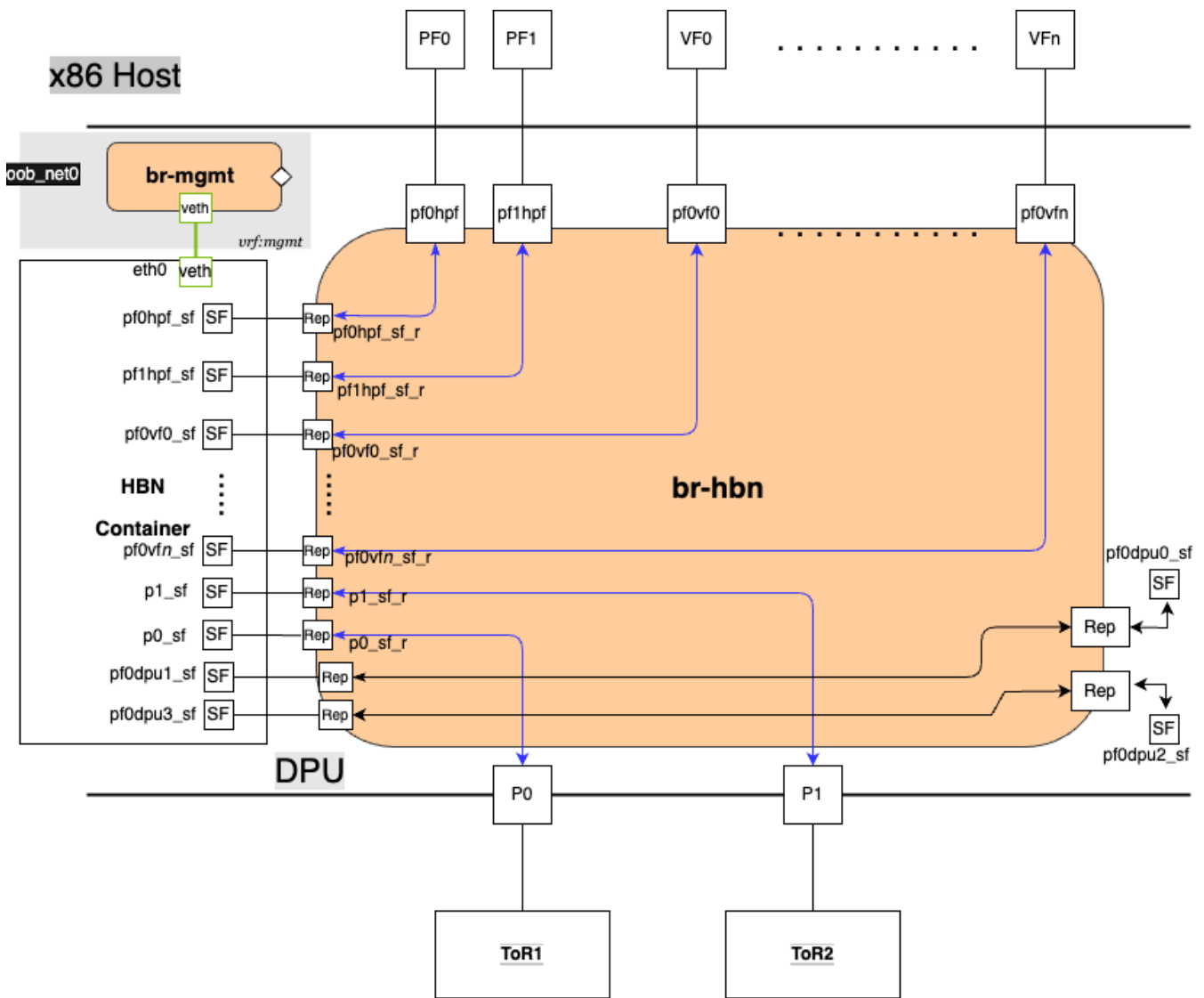
The HBN service comes with four types of configurable interfaces:

- Two uplinks (p0_sf, p1_sf)
- Two PF port representors (pf0hpf_sf, pf1hpf_sf)
- User-defined number of VFs (i.e., pf0vf0_sf, pf0vf1_sf, ..., pf1vf0_sf, pf1vf1_sf, ...)
- Two interfaces to connect to services running on BlueField, outside of the HBN container (pf0dpu1_sf and pf0dpu3_sf)

The *_sf suffix indicates that these are sub-functions and are different from the physical uplinks (i.e., PFs, VFs). They can be viewed as virtual interfaces from a virtualized BlueField.

Each of these interfaces is connected outside the HBN container to the corresponding physical interface, see section "[Service Function Chaining](#)" (SFC) for more details.

The HBN container runs as an isolated namespace and does not see any interfaces outside the container (oob_net0, real uplinks and PFs, *_sf_r representors).



pf0dpu1_sf and pf0dpu3_sf are special interfaces for HBN to connect to services running on BlueField. Their counterparts pf0dpu0_sf and pf0dpu2_sf respectively are located outside the HBN container. See section "[Connecting to DOCA Services to HBN on BlueField Arm](#)" for deployment considerations when using the pf0dpu1_sf or pf0dpu3_sf interface in HBN.

eth0 is equivalent to the oob_net0 interface in the HBN container. It is part of the management VRF of the container. It is not configurable via NVUE and does not need any configuration from the user. See section "[MGMT VRF Inside HBN Container](#)" for more details on this interface and the management VRF.

HBN Deployment Considerations

SF Interface State Tracking

When HBN is deployed with SFC, the interface state of the following network devices is propagated to their corresponding SFs:

- Uplinks – p0, p1
- PFs – pf0hpf, pf1hpf
- VFs – pf0vfX, pf1vfX where X is the VF number

For example, if the p0 uplink cable gets disconnected:

- p0 transitions to DOWN state with NO-CARRIER (default behavior on Linux); and
- p0 state is propagated to p0_sf whose state also becomes DOWN with NO-CARRIER

After p0 connection is reestablished:

- p0 transitions to UP state; and
- p0 state is propagated to p0_sf whose state becomes UP

Interface state propagation only happens in the uplink/PF/VF-to-SF direction.

A daemon called `sfc-state-propagation` runs on BlueField, outside of the HBN container, to sync the state. The daemon listens to netlink notifications for interfaces and transfers the state to SFs.

SF Interface MTU

In the HBN container, all the interfaces MTU are set to 9216 by default. MTU of specific interfaces can be overwritten using flat-files configuration or NVUE.

On BlueField side (i.e., outside of the HBN container), the MTU of the uplinks, PFs and VFs interfaces are also set to 9216. This can be changed by modifying `/etc/systemd/network/30-hbn-mtu.network` or by adding a new configuration file in the `/etc/systemd/network` for specific directories.

To reload this configuration, execute `systemctl restart systemd-networkd`.

Connecting to DOCA Services to HBN on BlueField Arm

There are various SF ports (named pf0dpuX_sf, where X is [0..n]) on BlueField Arm, which can be used to run any services on BlueField and use HBN to provide network connectivity. These ports are always created and connected in pairs of even and odd numbered ports, where even numbered ports are on BlueField side and odd numbered port are on the HBN side. For example, pf0dpu0_sf can be used by another service running on BlueField Arm to connect to HBN port pf0dpu1_sf.

Traffic between BlueField and the outside world is hardware-accelerated when the HBN side port is an L3 interface or access-port using switch virtual interface (SVI). So, it is treated the same way as PF or VF ports from a traffic handling standpoint.

Info

There are 2 SF port pairs created by default on BlueField Arm side so there can be 2 separate DOCA services running at same time.

Disabling BlueField Uplinks

The uplink ports must be always kept administratively up for proper operation of HBN. Otherwise, the NVIDIA® ConnectX® firmware would bring down the corresponding representor port which would cause data forwarding to stop.

Note

Change in operational status of uplink (e.g., carrier down) would result in traffic being switched to the other uplink.

When using ECMP failover on the two uplink SFs, locally disabling one uplink does not result in traffic switching to the second uplink. Disabling local link in this case means to set one uplink admin DOWN directly on BlueField.

To test ECMP failover scenarios correctly, the uplink must be disabled from its remote counterpart (i.e., execute admin DOWN on the remote system's link which is connected to the uplink).

HBN NVUE User Credentials

The preconfigured default user credentials are as follows:

Username	nvidia
Password	nvidia

NVUE user credentials can be added post installation:

1. This can be done by specifying additional `--username` and `--password` to the HBN startup script (refer to "[Running HBN Preparation Script](#)"). For example:

```
sudo ./hbn-dpu-setup.sh -u newuser -p newpassword
```

2. After executing this script, respawn the container or start the `decrypt-user-add` script inside running HBN container:

```
supervisorctl start decrypt-user-add  
decrypt-user-add: started
```

The script creates a new user in the HBN container:

```
cat /etc/passwd | grep newuser  
newuser:x:1001:1001::/home/newuser:/bin/bash
```

HBN NVUE Interface Classification

Interface	Interface Type	NVUE Type
p0_sf	Uplink representor	swp
p1_sf	Uplink representor	swp
lo	Loopback	loopback
pf0hpf_sf	Host representor	swp
pf1hpf_sf	Host representor	swp
pf0vfx_sf (where x is 0-255)	VF representor	swp
pf1vfx_sf (where x is 0-255)	VF representor	swp

HBN Files Persistence

The following directories are mounted from BlueField Arm to the HBN container namespace and are persistent across HBN service restarts and BlueField reboots:

	BlueField Arm Mount Point	HBN Container Mount Point
Configuration file mount points	/var/lib/hbn/etc/network/	/etc/network/
	/var/lib/hbn/etc/frr/	/etc/frr/
	/var/lib/hbn/etc/nvue.d/	/etc/nvue.d/
	/var/lib/hbn/etc/supervisor/conf.d/	/etc/supervisor/conf.d/
	/var/lib/hbn/var/lib/nvue/	/var/lib/nvue/
Support and log file mount points	/var/lib/hbn/var/support/	/var/support/
	/var/log/doca/hbn/	/var/log/hbn/

SR-IOV Support in HBN

Creating SR-IOV VFs on Host

The first step to use SR-IOV is to create Virtual Functions (VFs) on the host server.

VFs can be created using the following command:

```
sudo echo N > /sys/class/net/<host-rep>/device/sriov_numvfs
```

Where:

- <host-rep> is one of the two host representors (e.g., ens1f0 or ens1f1)
- $0 \leq N \leq 16$ is the desired total number of VFs
 - Set $N=0$ to delete all the VFs on $0 \leq N \leq 16$
 - $N=16$ is the maximum number of VFs supported on HBN across all representors

Automatic Creation of VF Representors and SF Devices on BlueField

VFs created on the host must have corresponding VF representor devices and SF devices for HBN on BlueField side. For example:

- ens1f0vf0 is the first SR-IOV VF device from the first host representor; this interface is created on the host server
- pf0vf0 is the corresponding VF representor device to ens1f0vf0; this device is present on the BlueField Arm side and automatically created at the same time as ens1f0vf0 is created by the user on the host side
- pf0vf0_sf is the corresponding SF device for pf0vf0 which is used to connect the VF to HBN pipeline

The creation of the SF device for VFs is done ahead of time when provisioning the BlueField and installing the DOCA image on it, see section "[Enabling SFC](#)" to see how to select how many SFs to create ahead of time.

The SF devices for VFs (i.e., pfXvfY) are pre-mapped to work with the corresponding VF representors when these are created with the command from the previous step.

Management VRF

Two management VRFs are automatically configured for HBN when BlueField is deployed with SFC:

- The first management VRF is outside the HBN container on BlueField. This VRF provides separation between out-of-band (OOB) traffic (via `oob_net0` or `tmfifo_net0`) and data-plane traffic via uplinks and PFs.
- The second management VRF is inside the HBN container and provides similar separation. The OOB traffic (via `eth0`) is isolated from the traffic via the `*_sf` interfaces.

MGMT VRF on BlueField Arm

The management (`mgmt`) VRF is enabled by default when the BlueField is deployed with SFC (see section "[Enabling SFC](#)"). The `mgmt` VRF provides separation between the OOB management network and the in-band data plane network.

The uplinks and PFs/VFs use the default routing table while the `oob_net0` (OOB Ethernet port) and the `tmfifo_net0` netdevices use the `mgmt` VRF to route their packets.

When logging in either via SSH or the console, the shell is by default in `mgmt` VRF context. This is indicated by a `mgmt` added to the shell prompt:

```
root@bf2:mgmt:/home/ubuntu#  
root@bf2:mgmt:/home/ubuntu# ip vrf identify  
mgmt.
```

When logging into the HBN container with `crictl`, the HBN shell will be in the default VRF. Users must switch to MGMT VRF manually if OOB access is required. Use `ip vrf exec` to do so.

```
root@bf2:mgmt:/home/ubuntu# ip vrf exec mgmt bash
```

The user must run `ip vrf exec mgmt` to perform operations requiring OOB access (e.g., apt-get update).

Network devices belonging to the mgmt VRF can be listed with the `vrf` utility:

```
root@bf2:mgmt:/home/ubuntu# vrf link list
```

```
VRF: mgmt
```

```
-----
```

```
tmfifo_net0 UP 00:1a:ca:ff:ff:03 <BROADCAST,MULTICAST,UP,LOWER_UP>
```

```
oob_net0 UP 08:c0:eb:c0:5a:32 <BROADCAST,MULTICAST,UP,LOWER_UP>
```

```
root@bf2:mgmt:/home/ubuntu# vrf help
```

```
vrf <OPTS>
```

```
VRF domains:
```

```
vrf list
```

```
Links associated with VRF domains:
```

```
vrf link list [<vrf-name>]
```

```
Tasks and VRF domain asociation:
```

```
vrf task exec <vrf-name> <command>
```

```
vrf task list [<vrf-name>]
```

```
vrf task identify <pid>
```

NOTE: This command affects only AF_INET and AF_INET6 sockets opened by the command that gets exec'ed. Specifically, it has **no** impact on netlink sockets (e.g., ip command).

To show the routing table for the default VRF, run:

```
root@bf2:mgmt:/home/ubuntu# ip route show
```

To show the routing table for the mgmt VRF, run:

```
root@bf2:mgmt:/home/ubuntu# ip route show vrf mgmt
```

MGMT VRF Inside HBN Container

Inside the HBN container, a separate mgmt VRF is present. Similar commands as those listed under section "[MGMT VRF on BlueField Arm](#)" can be used to query management routes.

The *_sf interfaces use the default routing table while the eth0 (OOB) uses the mgmt VRF to route out-of-band packets out of the container. The OOB traffic gets NATed through the oob_net0 interface on BlueField Arm, ultimately using the BlueField OOB's IP address.

When logging into the HBN container via `crictl`, the shell enters the default VRF context by default. Switching to the mgmt VRF can be done using the command `ip vrf exec mgmt <cmd>`.

Existing Services in MGMT VRF on BlueField Arm

On the BlueField Arm, outside the HBN container, a set of existing services run in the mgmt VRF context as they need OOB network access:

- containerd
- kubelet
- ssh
- docker

These services can be restarted and queried for their status using the command `systemctl` while adding `@mgmt` to the original service name. For example:

- To restart containerd:

```
root@bf2:mgmt:/home/ubuntu# systemctl restart containerd@mgmt
```

- To query containerd status:


```
root@bf2:mgmt:/home/ubuntu# systemctl status containerd@mgmt
```

Note

The original version of these services (without @mgmt) are not used and must not be started.

Running New Service in MGMT VRF on BlueField Arm

If a service needs OOB access to run, it can be added to the set of services running in mgmt VRF context. Adding such a service is only possible on the BlueField Arm (i.e., outside the HBN container).

To add a service to the set of mgmt VRF services:

1. Add it to `/etc/vrf/systemd.conf` (if it is not present already). For example, NTP is already listed in this file.
2. Run the following:

```
root@bf2:mgmt:/home/ubuntu# systemctl daemon-reload
```

3. Stop and disable to the non-VRF version of the service to be able to start the mgmt VRF one:

```
root@bf2:mgmt:/home/ubuntu# systemctl stop ntp
root@bf2:mgmt:/home/ubuntu# systemctl disable ntp
root@bf2:mgmt:/home/ubuntu# systemctl enable ntp@mgmt
root@bf2:mgmt:/home/ubuntu# systemctl start ntp@mgmt
```

Configuration

To start configuring HBN, log into the HBN container:

```
sudo crictl exec -it $(crictl ps | grep hbn | awk '{print $1;}') bash
```

General Network Configuration

Flat Files Configuration

Add network interfaces and FRR configuration files to HBN to achieve the desired configuration:

- /etc/network/interfaces

Note

Refer to [NVIDIA® Cumulus® Linux documentation](#) for more information.

- /etc/frr/frr.conf; /etc/frr/daemons

Note

Refer to [NVIDIA® Cumulus® Linux documentation](#) for more information.

NVUE Configuration

This section assumes familiarity with [NVIDIA user experience \(NVUE\) Cumulus Linux documentation](#). The following subsections, only expand on HBN-specific aspects of NVUE.

NVUE Service

HBN installs NVUE by default and enables NVUE service at boot.

NVUE REST API

HBN enables REST API by default.

Users may run the cURL commands from the command line. Use the default HBN username `nvidia` and password `nvidia`.

To change the default password of the `nvidia` user or add additional users for NVUE access, refer to section "[HBN NVUE User Credentials](#)".

REST API example:

```
curl -u 'nvidia:nvidia' --insecure
https://<mgmt_ip>:8765/nvue_v1/vrf/default/router/bgp
{
  "configured-neighbors": 2,
  "established-neighbors": 2,
  "router-id": "10.10.10.201"
}
```

Note

For information about using the NVUE REST API, refer to the [NVUE API documentation](#).

NVUE CLI

For information about using the NVUE CLI, refer to the [NVUE CLI documentation](#)

NVUE Startup Configuration File

When the network configuration is saved using NVUE, HBN writes the configuration to the `/etc/nvue.d/startup.yaml` file.

Startup configuration is applied by following the supervisor daemon at boot time. `nvued-startup` will appear in `EXITED` state after applying the startup configuration.

```
# supervisorctl status nvued-startup
nvued-startup EXITED Apr 17 10:04 AM
```

Note

`nv config apply startup` applies the yaml configuration saved at `/etc/nvue.d/`.

Note

`nv config save` saves the running configuration to `/etc/nvue.d/startup.yaml`.

HBN Configuration Examples

HBN Default Configuration

After a fresh HBN installation, the default `/etc/network/interfaces` file would contain only the declaration of the two uplink SFs and a loopback interface.

```
source /etc/network/interfaces.d/*.intf

auto lo
iface lo inet loopback

auto p0_sf
iface p0_sf

auto p1_sf
iface p1_sf
```

FRR configuration files would also be present under `/etc/frr/` but no configuration would be enabled.

Layer-3 Routing

Native Routing with BGP and ECMP

HBN supports unicast routing with BGP and ECMP for IPv4 and IPv6 traffic. ECMP is achieved by distributing traffic using hash calculation based on the source IP , destination IP, and protocol type of the IP header.

Info

For TCP and UDP packets, it also includes source port and destination port.

ECMP Example

ECMP is implemented any time routes have multiple paths over uplinks or host ports. For example, 20.20.20.0/24 has 2 paths using both uplinks, so a path is selected based on a hash of the IP headers.

```
20.20.20.0/24 proto bgp metric 20
nextthop via 169.254.0.1 dev p0_sf weight 1 onlink <<<<< via uplink p0_sf
nextthop via 169.254.0.1 dev p1_sf weight 1 onlink <<<<< via uplink p1_sf
```

Info

HBN supports up to 16 paths for ECMP.

Sample NVUE Configuration for Native Routing

```
nv set interface lo ip address 10.10.10.1/32
nv set interface lo ip address 2010:10:10::1/128
nv set interface vlan100 type svi
nv set interface vlan100 vlan 100
nv set interface vlan100 base-interface br_default
nv set interface vlan100 ip address 2030:30:30::1/64
nv set interface vlan100 ip address 30.30.30.1/24
nv set bridge domain br_default vlan 100
nv set interface pf0hpf_sf,pf1hpf_sf bridge domain br_default access 100
nv set vrf default router bgp router-id 10.10.10.1
nv set vrf default router bgp autonomous-system 65501
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
```

```
nv set vrf default router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf default router bgp neighbor p0_sf remote-as external
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p0_sf address-family ipv4-unicast enable on
nv set vrf default router bgp neighbor p0_sf address-family ipv6-unicast enable on
nv set vrf default router bgp neighbor p1_sf remote-as external
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf address-family ipv4-unicast enable on
nv set vrf default router bgp neighbor p1_sf address-family ipv6-unicast enable on
```

Sample Flat Files Configuration for Native Routing

Example /etc/network/interfaces configuration:

```
auto lo
iface lo inet loopback
address 10.10.10.1/32
address 2010:10:10::1/128

auto p0_sf
iface p0_sf

auto p1_sf
iface p1_sf

auto pf0hpf_sf
iface pf0hpf_sf
bridge-access 100

auto pf1hpf_sf
iface pf1hpf_sf
bridge-access 100

auto vlan100
```

```
iface vlan100
address 2030:30:30::1/64
address 30.30.30.1/24
vlan-raw-device br_default
vlan-id 100

auto br_default
iface br_default
bridge-ports pf0hpf_sf pf1hpf_sf
bridge-vlan-aware yes
bridge-vids 100
bridge-pvid 1
```

Example `/etc/frr/daemons` configuration:

```
bgpd=yes
vtysh_enable=yes

FRR Config file @ /etc/frr/frr.conf -
!
frr version 7.5+cl5.3.0u0
frr defaults datacenter
hostname BLUEFIELD2
log syslog informational
no zebra nexthop kernel enable
!
router bgp 65501
bgp router-id 10.10.10.1
bgp bestpath as-path multipath-relax
neighbor p0_sf interface remote-as external
neighbor p0_sf advertisement-interval 0
neighbor p0_sf timers 3 9
neighbor p0_sf timers connect 10
neighbor p1_sf interface remote-as external
```



```

neighbor p1_sf advertisement-interval 0
neighbor p1_sf timers 3 9
neighbor p1_sf timers connect 10
!
address-family ipv4 unicast
redistribute connected
maximum-paths 64
maximum-paths ibgp 64
exit-address-family
!
address-family ipv6 unicast
redistribute connected
neighbor p0_sf activate
neighbor p1_sf activate
maximum-paths 64
maximum-paths ibgp 64
exit-address-family
!
line vty
!
end

```

Direct Routing on Host-facing Interfaces

Host-facing interfaces (PFs and VFs) are not restricted to be part of the bridge for routing. HBN supports L3-only configuration with direct routing on host-facing PFs and VFs.

Sample NVUE Configuration

```

nv set interface pf0hpf_sf ip address 30.30.11.1/24
nv set interface pf0hpf_sf ip address 2030:30:11::1/64
nv set interface pf0vf0_sf ip address 30.30.13.1/24
nv set interface pf0vf0_sf ip address 2030:30:13::1/64

```

Sample Flat File Configuration

```
auto pf0hpf_sf
iface pf0hpf_sf
address 2030:30:11::1/64
address 30.30.11.1/24

auto pf0vf0_sf
iface pf0vf0_sf
address 2030:30:13::1/64
address 30.30.13.1/24
```

BGP Peering with the Host

HBN supports the ability to establish a BGP session between the host and the HBN service running on BlueField Arm and allow the host to announce arbitrary route prefixes through the BlueField into the underlay fabric. The host can use any standard BGP protocol stack implementation to establish BGP peering with HBN.

Traffic to and from endpoints on the host gets offloaded.

Note

Both IPv4 and IPv6 unicast AFI/SAFI are supported.

It is possible to apply route filtering for these prefixes to limit the potential security impact in this configuration.

Sample NVUE Configuration for Host BGP Peering

The following code block shows configuration to peer to host at 45.3.0.4 and 2001:cafe:1ead::4. The BGP session can be established using IPv4 or IPv6 address.

Note

Either of these sessions can support IPv4 unicast and IPv6 unicast AFI/SAFI.

NVUE configuration for peering with host:

```
nv set vrf default router bgp autonomous-system 63642
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 45.3.0.4 nexthop-connected-check off
nv set vrf default router bgp neighbor 45.3.0.4 peer-group dpu_host
nv set vrf default router bgp neighbor 45.3.0.4 type numbered
nv set vrf default router bgp neighbor 2001:cafe:1ead::4 nexthop-connected-check
off
nv set vrf default router bgp neighbor 2001:cafe:1ead::4 peer-group dpu_host
nv set vrf default router bgp neighbor 2001:cafe:1ead::4 type numbered
nv set vrf default router bgp peer-group dpu_host address-family ipv4-unicast
enable on
nv set vrf default router bgp peer-group dpu_host address-family ipv6-unicast
enable on
nv set vrf default router bgp peer-group dpu_host remote-as external
```

Sample Flat Files Configuration for Host BGP peering

The following block shows configuration to peer to host at 45.3.0.4 and 2001:cafe:1ead::4. The BGP session can be established using IPv4 or IPv6 address.

frr.conf file:

```
router bgp 63642
  bgp router-id 27.0.0.4
  bgp bestpath as-path multipath-relax
  neighbor dpu_host peer-group
```

```
neighbor dpu_host remote-as external
neighbor dpu_host advertisement-interval 0
neighbor dpu_host timers 3 9
neighbor dpu_host timers connect 10
neighbor dpu_host disable-connected-check
neighbor fabric peer-group
neighbor fabric remote-as external
neighbor fabric advertisement-interval 0
neighbor fabric timers 3 9
neighbor fabric timers connect 10
neighbor 45.3.0.4 peer-group dpu_host
neighbor 2001:cafe:1ead::4 peer-group dpu_host
neighbor p0_sf interface peer-group fabric
neighbor p1_sf interface peer-group fabric
!
address-family ipv4 unicast
neighbor dpu_host activate
!
address-family ipv6 unicast
neighbor dpu_host activate
```

Sample FRR configuration on the Host

Any BGP implementation can be used on the host to peer to HBN and advertise endpoints. The following is an example using FRR BGP:

Sample FRR configuration on the host:

```
bf2-s12# sh run
Building configuration...

Current configuration:
!
frr version 7.2.1
frr defaults traditional
```

```
hostname bf2-s12
no ip forwarding
no ipv6 forwarding
!
router bgp 1000008
!
router bgp 1000008 vrf v_200_2000
neighbor 45.3.0.2 remote-as external
neighbor 2001:cafe:1ead::2 remote-as external
!
address-family ipv4 unicast
redistribute connected
exit-address-family
!
address-family ipv6 unicast
redistribute connected
neighbor 45.3.0.2 activate
neighbor 2001:cafe:1ead::2 activate
exit-address-family
!
line vty
!
end
```

Sample interfaces configuration on the host:

```
root@bf2-s12:/home/cumulus# ifquery -a
auto lo
iface lo inet loopback
address 27.0.0.7/32
address 2001:c000:10ff:f00d::7/128

auto v_200_2000
iface v_200_2000
address 60.1.0.1
```

```
address 60.1.0.2
address 60.1.0.3
address 2001:60:1::1
address 2001:60:1::2
address 2001:60:1::3
vrf-table auto
auto ens1f0np0
iface ens1f0np0
address 45.3.0.4/24
address 2001:cafe:1ead::4/64
gateway 45.3.0.1
gateway 2001:cafe:1ead::1
vrf v_200_2000
hwaddress 00:03:00:08:00:12
mtu 9162
```

VRF Route Leaking

VRFs are typically used when multiple independent routing and forwarding tables are desirable. However, users may want to reach destinations in one VRF from another VRF, as in the following cases:

- To make a service, such as a firewall available to multiple VRFs
- To enable routing to external networks or the Internet for multiple VRFs, where the external network itself is reachable through a specific VRF

Route leaking can be used to reach remote destinations as well as directly connected destinations in another VRF. Multiple VRFs can import routes from a single source VRF, and a VRF can import routes from multiple source VRFs. This can be used when a single VRF provides connectivity to external networks or a shared service for other VRFs. It is possible to control the routes leaked dynamically across VRFs with a route map.

When route leaking is used:

- The `redistribute` command (not `network` command) must be used in BGP to leak non-BGP routes (connected or static routes)

- It is not possible to leak routes between the default and non-default VRF

i Note

Ping or other IP traffic from a locally connected host in vrfX to a local interface IP address on the BlueField/HBN in vrfY does not work, even if VRF route-leaking is enabled between these two VRFs.

In the following example commands, routes in the BGP routing table of VRF BLUE dynamically leak into VRF RED:

```
nv set vrf RED router bgp address-family ipv4-unicast route-import from-vrf list BLUE
nv config apply
```

The following example commands delete leaked routes from VRF BLUE to VRF RED:

```
nv unset vrf RED router bgp address-family ipv4-unicast route-import from-vrf list
BLUE
nv config apply
```

To exclude certain prefixes from the import process, configure the prefixes in a route map.

The following example configures a route map to match the source protocol BGP and imports the routes from VRF BLUE to VRF RED. For the imported routes, the community is 11:11 in VRF RED.

```
nv set vrf RED router bgp address-family ipv4-unicast route-import from-vrf list BLUE
nv set router policy route-map BLUEtoRED rule 10 match type ipv4
nv set router policy route-map BLUEtoRED rule 10 match source-protocol bgp
nv set router policy route-map BLUEtoRED rule 10 action permit
nv set router policy route-map BLUEtoRED rule 10 set community 11:11
```

```
nv set vrf RED router bgp address-family ipv4-unicast route-import from-vrf route-  
map BLUEtoRED  
nv config
```

To check the status of the VRF route leaking, run:

- NVUE command:

```
nv show vrf <vrf-name> router bgp address-family ipv4-unicast route-import
```

- Vtysh command:

```
show ip bgp vrf <vrf-name> ipv4|ipv6 unicast route-leak command.
```

- For example:

```
nv show vrf RED router bgp address-family ipv4-unicast route-import  
operational applied  
-----  
from-vrf  
enable on  
route-map BLUEtoRED  
[list] BLUE BLUE  
[route-target] 10.10.10.1:3
```

To show more detailed status information, the following NVUE commands are available:

- `nv show vrf <vrf-name> router bgp address-family ipv4-unicast route-import from-vrf`
- `nv show vrf <vrf-name> router bgp address-family ipv4-unicast route-import from-vrf list`
- `nv show vrf <vrf-name> router bgp address-family ipv4-unicast route-import from-vrf list <leak-vrf-id>`

To view the BGP routing table, run:

- NVUE command:


```
nv show vrf <vrf-name> router bgp address-family ipv4-unicast
```

- Vtysh command:

```
show ip bgp vrf <vrf-name> ipv4|ipv6 unicast
```

To view the FRR IP routing table, run:

- Vtysh command:

```
show ip route vrf <vrf-name>
```

- Or:

```
net show route vrf <vrf-name>
```

Info

These commands show all routes, including routes leaked from other VRFs.

VLAN Subinterfaces

A VLAN subinterface is a VLAN device on an interface. The VLAN ID appends to the parent interface using dot (.) VLAN notation which is a standard way to specify a VLAN device in Linux.

For example:

- A VLAN with ID 100 which is a subinterface of p0_sf is annotated as p0_sf.100

- The subinterface `p0_sf.100` only receives packets that have a VLAN 100 tag on port `p0_sf`
- Any packets transmitted from `p0_sf.100` would have VLAN tag 100

In HBN, VLAN subinterfaces can be created on uplink ports as well as on the host-facing PF and VF ports. A VLAN subinterface only receives traffic tagged for that VLAN.

Note

VLAN subinterfaces are L3 interfaces and should not be added to a bridge.

In the following example, uplink subinterface on `p0_sf` with VLAN ID 10 and a host facing subinterface on VF ports `pf1vf0_sf` with VLAN ID 999 are created. The host-facing subinterface is also assigned with IPv4 and IPv6 addresses.

Subinterface configuration using NVUE commands:

```
nv set interface p0_sf.10 base-interface p0_sf
nv set interface p0_sf.10 type sub
nv set interface p0_sf.10 vlan 10

nv set interface pf1vf0_sf type swp
nv set interface pf1vf0_sf.999 base-interface pf1vf0_sf
nv set interface pf1vf0_sf.999 type sub
nv set interface pf1vf0_sf.999 vlan 999
nv set interface pf1vf0_sf ip address 30.30.14.1/24
nv set interface pf1vf0_sf ip address 2030:30:14::1/64
```

Same configuration using sample flat file in `/etc/network/interfaces`:

```
auto p0_sf.10
iface p0_sf.10
```

```
auto pf1vf0_sf.999
iface pf1vf0_sf.999
address 2030:30:40::1/64
address 30.30.40.1/24
```

Ethernet Virtual Private Network – EVPN

HBN supports VXLAN with EVPN control plane for intra-subnet bridging (L2) services for IPv4 and IPv6 traffic in the overlay.

For the underlay, only IPv4 or BGP unnumbered configuration is supported.

Note

HBN supports VXLAN encapsulation only over uplink parent interfaces.

Single VXLAN Device

With a single VXLAN device, a set of VXLAN network identifiers (VNIs) represents a single device model. The single VXLAN device has a set of attributes that belong to the VXLAN construct. Individual VNIs include VLAN-to-VNI mapping which allows users to specify which VLANs are associated with which VNIs. A single VXLAN device simplifies the configuration and reduces the overhead by replacing multiple traditional VXLAN devices with a single VXLAN device.

Users may configure a single VXLAN device automatically with NVUE, or manually by editing the `/etc/network/interfaces` file. When users configure a single VXLAN device with NVUE, NVUE creates a unique name for the device in the following format using the bridge name as the hash key: `vxlan<id>`.

This example configuration performs the following steps:

1. Creates a single VXLAN device (`vxlan21`).

2. Maps VLAN 10 to VNI 10 and VLAN 20 to VNI 20.
3. Adds the VXLAN device to the default bridge.

```
cumulus@leaf01:~$ nv set bridge domain bridge vlan 10 vni 10
cumulus@leaf01:~$ nv set bridge domain bridge vlan 20 vni 20
cumulus@leaf01:~$ nv set nve vxlan source address 10.10.10.1
cumulus@leaf01:~$ nv config apply
```

Alternately, users may edit the file `/etc/network/interfaces` as follows, then run the `ifreload -a` command to apply the SVD configuration.

```
auto lo
iface lo inet loopback
vxlan-local-tunnelip 10.10.10.1

auto vxlan21
iface vxlan21
bridge-vlan-vni-map 10=10 20=20
bridge-learning off

auto bridge
iface bridge
bridge-vlan-aware yes
bridge-ports vxlan21 pf0hpf_sf pf1hpf_sf
bridge-vids 10 20
bridge-pvid 1
```

Note

Users may not use a combination of single and traditional VXLAN devices.

Sample Switch Configuration for EVPN

The following is a sample NVUE config for underlay switches (NVIDIA® Spectrum® with Cumulus Linux) to enable EVPN deployments with HBN.

It assumes that the uplinks on all BlueField devices are connected to ports `swp1-4` on the switch.

```
nv set evpn enable on
nv set router bgp enable on

nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on

nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 63640
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor swp1 peer-group fabric
nv set vrf default router bgp neighbor swp1 type unnumbered
nv set vrf default router bgp neighbor swp2 peer-group fabric
nv set vrf default router bgp neighbor swp2 type unnumbered
nv set vrf default router bgp neighbor swp3 peer-group fabric
nv set vrf default router bgp neighbor swp3 type unnumbered
nv set vrf default router bgp neighbor swp4 peer-group fabric
nv set vrf default router bgp neighbor swp4 type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable
on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable
on
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-
tx off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable on
nv set vrf default router bgp peer-group fabric remote-as external
```

```
nv set vrf default router bgp router-id 27.0.0.10

nv set interface lo ip address 2001:c000:10ff:f00d::10/128
nv set interface lo ip address 27.0.0.10/32
nv set interface lo type loopback
nv set interface swp1,swp2,swp3,swp4 type swp
```

Layer-2 EVPN

Sample NVUE Configuration for L2 EVPN

The following is a sample NVUE configuration which has L2-VNIs (2000, 2001) for EVPN bridging on BlueField.

```
nv set bridge domain br_default encap 802.1Q
nv set bridge domain br_default type vlan-aware
nv set bridge domain br_default vlan 200 vni 2000 flooding enable auto
nv set bridge domain br_default vlan 200 vni 2000 mac-learning off
nv set bridge domain br_default vlan 201 vni 2001 flooding enable auto
nv set bridge domain br_default vlan 201 vni 2001 mac-learning off

nv set evpn enable on
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.4
nv set router bgp enable on
nv set system global anycast-mac 44:38:39:42:42:07
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on

nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 63642
nv set vrf default router bgp enable on
```

```
nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable
on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast policy
outbound route-map MY_ORIGIN_ASPATH_ONLY
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable
on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast policy
outbound route-map MY_ORIGIN_ASPATH_ONLY
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-
tx off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable on
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp router-id 27.0.0.4

nv set interface lo ip address 2001:c000:10ff:f00d::4/128
nv set interface lo ip address 27.0.0.4/32
nv set interface lo type loopback
nv set interface p0_sf,p1_sf,pf0hpf_sf,pf1hpf_sf type swp
nv set interface pf0hpf_sf bridge domain br_default access 200
nv set interface pf1hpf_sf bridge domain br_default access 201

nv set interface vlan200-201 base-interface br_default
nv set interface vlan200-201 ip ipv4 forward on
nv set interface vlan200-201 ip ipv6 forward on
nv set interface vlan200-201 ip vrr enable on
nv set interface vlan200-201 ip vrr state up
nv set interface vlan200-201 link mtu 9050
nv set interface vlan200-201 type svi
nv set interface vlan200 ip address 2001:cafe:1ead::3/64
nv set interface vlan200 ip address 45.3.0.2/24
nv set interface vlan200 ip vrr address 2001:cafe:1ead::1/64
```

```
nv set interface vlan200 ip vrr address 45.3.0.1/24
nv set interface vlan200 vlan 200
nv set interface vlan201 ip address 2001:cafe:1ead:1::3/64
nv set interface vlan201 ip address 45.3.1.2/24
nv set interface vlan201 ip vrr address 2001:cafe:1ead:1::1/64
nv set interface vlan201 ip vrr address 45.3.1.1/24
nv set interface vlan201 vlan 201
```

Sample Flat Files Configuration for L2 EVPN

The following is a sample flat files configuration which has L2-VNIs (vx-2000, vx-2001) for EVPN bridging on BlueField.

This file is located at `/etc/network/interfaces`:

```
auto lo
iface lo inet loopback
address 2001:c000:10ff:f00d::4/128
address 27.0.0.4/32
vxlan-local-tunnelip 27.0.0.4

auto p0_sf
iface p0_sf

auto p1_sf
iface p1_sf

auto pf0hpf_sf
iface pf0hpf_sf
bridge-access 200

auto pf1hpf_sf
iface pf1hpf_sf
bridge-access 201
```



```
auto vlan200
iface vlan200
address 2001:cafe:1ead::3/64
address 45.3.0.2/24
mtu 9050
address-virtual 00:00:5e:00:01:01 2001:cafe:1ead::1/64 45.3.0.1/24
vlan-raw-device br_default
vlan-id 200

auto vlan201
iface vlan201
address 2001:cafe:1ead:1::3/64
address 45.3.1.2/24
mtu 9050
address-virtual 00:00:5e:00:01:01 2001:cafe:1ead:1::1/64 45.3.1.1/24
vlan-raw-device br_default
vlan-id 201

auto vxlan48
iface vxlan48
bridge-vlan-vni-map 200=2000 201=2001
217=2017
bridge-learning off

auto br_default
iface br_default
bridge-ports pf0hpf_sf pf1hpf_sf vxlan48
bridge-vlan-aware yes
bridge-vids 200 201
bridge-pvid 1
```

This file tells the frr package which daemon to start and is located at `/etc/frr/daemons:`

```
bgpd=yes
ospfd=no
```

```
ospf6d=no
isisd=no
pimd=no
ldpd=no
pbrd=no
vrrpd=no
fabricd=no
nhripd=no
eigrpd=no
babeld=no
sharpd=no
fabricd=no
ripngd=no
ripd=no

vtysh_enable=yes
zebra_options=" -M cumulus_mlag -M snmp -A 127.0.0.1 -s 90000000"
bgpd_options=" -M snmp -A 127.0.0.1"
ospfd_options=" -M snmp -A 127.0.0.1"
ospf6d_options=" -M snmp -A ::1"
ripd_options=" -A 127.0.0.1"
ripngd_options=" -A ::1"
isisd_options=" -A 127.0.0.1"
pimd_options=" -A 127.0.0.1"
ldpd_options=" -A 127.0.0.1"
nhripd_options=" -A 127.0.0.1"
eigrpd_options=" -A 127.0.0.1"
babeld_options=" -A 127.0.0.1"
sharpd_options=" -A 127.0.0.1"
pbrd_options=" -A 127.0.0.1"
staticd_options=" -A 127.0.0.1"
fabricd_options=" -A 127.0.0.1"
vrrpd_options=" -A 127.0.0.1"

frr_profile="datacenter"
```

FRR configuration file is located at /etc/frr/frr.conf:

```
!---- Cumulus Defaults ----
frr defaults datacenter
log syslog informational
no zebra nexthop kernel enable
vrf default
outer bgp 63642 vrf default
bgp router-id 27.0.0.4
bgp bestpath as-path multipath-relax
timers bgp 3 9
bgp deterministic-med
! Neighbors
neighbor fabric peer-group
neighbor fabric remote-as external
neighbor fabric timers 3 9
neighbor fabric timers connect 10
neighbor fabric advertisement-interval 0
neighbor p0_sf interface peer-group fabric
neighbor p1_sf interface peer-group fabric
address-family ipv4 unicast
maximum-paths ibgp 64
maximum-paths 64
distance bgp 20 200 200
neighbor fabric activate
exit-address-family
address-family ipv6 unicast
maximum-paths ibgp 64
maximum-paths 64
distance bgp 20 200 200
neighbor fabric activate
exit-address-family
address-family l2vpn evpn
advertise-all-vni
neighbor fabric activate
```

Layer-3 EVPN with Symmetric Routing

In distributed symmetric routing, each VXLAN endpoint (VTEP) acts as a layer-3 gateway, performing routing for its attached hosts. However, both the ingress VTEP and egress VTEP route the packets (similar to traditional routing behavior of routing to a next-hop router). In a VXLAN encapsulated packet, the inner destination MAC address is the router MAC address of the egress VTEP to indicate that the egress VTEP is the next hop and that it must also perform the routing.

All routing happens in the context of a tenant (VRF). For a packet that the ingress VTEP receives from a locally attached host, the SVI interface corresponding to the VLAN determines the VRF. For a packet that the egress VTEP receives over the VXLAN tunnel, the VNI in the packet has to specify the VRF. For symmetric routing, this is a VNI corresponding to the tenant and is different from either the source VNI or the destination VNI. This VNI is a layer-3 VNI or interconnecting VNI. The regular VNI, which maps a VLAN, is the layer-2 VNI.

For more details about this, refer to the [Cumulus Linux User Manual](#) .

Info

HBN uses a one-to-one mapping between an L3 VNI and a tenant (VRF).

Info

The VRF to L3 VNI mapping has to be consistent across all VTEPs.

Info

An L3 VNI and an L2 VNI cannot have the same ID.

In an EVPN symmetric routing configuration, when the switch announces a type-2 (MAC/IP) route, in addition to containing two VNIs (L2 and L3 VNIs), the route also contains separate route targets (RTs) for L2 and L3. The L3 RT associates the route with the tenant VRF. By default, this is auto-derived using the L3 VNI instead of the L2 VNI. However, this is configurable.

For EVPN symmetric routing, users must perform the configuration listed in the following subsections. Optional configuration includes configuring a route distinguisher (RD) and RTs for the tenant VRF, and advertising the locally-attached subnets.

Sample NVUE Configuration for L3 EVPN

If using NVUE to configure EVPN symmetric routing, the following is a sample configuration using NVUE commands:

```
nv set bridge domain br_default vlan 111 vni 1000111
nv set bridge domain br_default vlan 112 vni 1000112
nv set bridge domain br_default vlan 213 vni 1000213
nv set bridge domain br_default vlan 214 vni 1000214
nv set evpn enable on
nv set interface lo ip address 6.0.0.19/32
nv set interface lo type loopback
nv set interface p0_sf description 'alias p0_sf to leaf-21 swp3'
nv set interface p0_sf,p1_sf,pf0hpf_sf,pf0vf0_sf,pf1hpf_sf,pf1vf0_sf type swp
nv set interface p1_sf description 'alias p1_sf to leaf-22 swp3'
nv set interface pf0hpf_sf bridge domain br_default access 111
nv set interface pf0hpf_sf description 'alias pf0hpf_sf to host-211 ens2f0np0'
nv set interface pf0vf0_sf bridge domain br_default access 112
nv set interface pf0vf0_sf description 'alias pf0vf0_sf to host-211 ens2f0np0v0'
nv set interface pf1hpf_sf bridge domain br_default access 213
nv set interface pf1hpf_sf description 'alias pf1hpf_sf to host-211 ens2f1np1'
```

```
nv set interface pf1vf0_sf bridge domain br_default access 214
nv set interface pf1vf0_sf description 'alias pf1vf0_sf to host-211 ens2f1np0v0'
nv set interface vlan111 ip address 60.1.1.21/24
nv set interface vlan111 ip address 2060:1:1:1::21/64
nv set interface vlan111 ip vrr address 60.1.1.250/24
nv set interface vlan111 ip vrr address 2060:1:1:1::250/64
nv set interface vlan111 vlan 111
nv set interface vlan111,213 ip vrf vrf2
nv set interface vlan111-112,213-214 ip vrr enable on
nv set interface vlan111-112,213-214 ip vrr mac-address 00:00:5e:00:01:01
nv set interface vlan111-112,213-214 ip ipv4 forward on
nv set interface vlan111-112,213-214 ip ipv6 forward on
nv set interface vlan111-112,213-214 type svi
nv set interface vlan112 ip address 50.1.1.21/24
nv set interface vlan112 ip address 2050:1:1:1::21/64
nv set interface vlan112 ip vrr address 50.1.1.250/24
nv set interface vlan112 ip vrr address 2050:1:1:1::250/64
nv set interface vlan112 vlan 112
nv set interface vlan112,214 ip vrf vrf1
nv set interface vlan213 ip address 60.1.210.21/24
nv set interface vlan213 ip address 2060:1:1:210::21/64
nv set interface vlan213 ip vrr address 60.1.210.250/24
nv set interface vlan213 ip vrr address 2060:1:1:210::250/64
nv set interface vlan213 vlan 213
nv set interface vlan214 ip address 50.1.210.21/24
nv set interface vlan214 ip address 2050:1:1:210::21/64
nv set interface vlan214 ip vrr address 50.1.210.250/24
nv set interface vlan214 ip vrr address 2050:1:1:210::250/64
nv set interface vlan214 vlan 214
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan source address 6.0.0.19
nv set platform
nv set router bgp enable on
nv set router policy route-map ALLOW_LOBR rule 10 action permit
nv set router policy route-map ALLOW_LOBR rule 10 match interface lo
```

```
nv set router policy route-map ALLOW_LOBR rule 20 action permit
nv set router policy route-map ALLOW_LOBR rule 20 match interface br_default
nv set router policy route-map ALLOW_VRF1 rule 10 action permit
nv set router policy route-map ALLOW_VRF1 rule 10 match interface vrf1
nv set router policy route-map ALLOW_VRF2 rule 10 action permit
nv set router policy route-map ALLOW_VRF2 rule 10 match interface vrf2
nv set router vrr enable on
nv set system global system-mac 00:01:00:00:1e:03
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast multipaths ebgp 16
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
route-map ALLOW_LOBR
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 650019
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor p0_sf address-family l2vpn-evpn add-path-tx
off
nv set vrf default router bgp neighbor p0_sf address-family l2vpn-evpn enable on
nv set vrf default router bgp neighbor p0_sf peer-group TOR_LEAF_SPINE
nv set vrf default router bgp neighbor p0_sf remote-as external
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf address-family l2vpn-evpn add-path-tx
off
nv set vrf default router bgp neighbor p1_sf address-family l2vpn-evpn enable on
nv set vrf default router bgp neighbor p1_sf peer-group TOR_LEAF_SPINE
nv set vrf default router bgp neighbor p1_sf remote-as external
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp path-selection routerid-compare on
nv set vrf default router bgp peer-group TOR_LEAF_SPINE address-family ipv4-unicast
enable on
nv set vrf default router bgp router-id 6.0.0.19
nv set vrf vrf1 evpn enable on
nv set vrf vrf1 evpn vni 104001
```

```
nv set vrf vrf1 loopback ip address 50.1.21.21/32
nv set vrf vrf1 loopback ip address 2050:50:50:21::21/128
nv set vrf vrf1 router bgp address-family ipv4-unicast enable on
nv set vrf vrf1 router bgp address-family ipv4-unicast redistribute connected enable
on
nv set vrf vrf1 router bgp address-family ipv4-unicast redistribute connected route-
map ALLOW_VRF1
nv set vrf vrf1 router bgp address-family ipv4-unicast route-export to-evpn enable
on
nv set vrf vrf1 router bgp address-family ipv6-unicast enable on
nv set vrf vrf1 router bgp address-family ipv6-unicast redistribute connected enable
on
nv set vrf vrf1 router bgp address-family ipv6-unicast redistribute connected route-
map ALLOW_VRF1
nv set vrf vrf1 router bgp address-family ipv6-unicast route-export to-evpn enable
on
nv set vrf vrf1 router bgp autonomous-system 650019
nv set vrf vrf1 router bgp enable on
nv set vrf vrf1 router bgp router-id 50.1.21.21
nv set vrf vrf2 evpn enable on
nv set vrf vrf2 evpn vni 104002
nv set vrf vrf2 loopback ip address 60.1.21.21/32
nv set vrf vrf2 loopback ip address 2060:60:60:21::21/128
nv set vrf vrf2 router bgp address-family ipv4-unicast enable on
nv set vrf vrf2 router bgp address-family ipv4-unicast redistribute connected enable
on
nv set vrf vrf2 router bgp address-family ipv4-unicast redistribute connected route-
map ALLOW_VRF2
nv set vrf vrf2 router bgp address-family ipv4-unicast route-export to-evpn enable
on
nv set vrf vrf2 router bgp address-family ipv6-unicast enable on
nv set vrf vrf2 router bgp address-family ipv6-unicast redistribute connected enable
on
nv set vrf vrf2 router bgp address-family ipv6-unicast redistribute connected route-
map ALLOW_VRF2
```



```
nv set vrf vrf2 router bgp address-family ipv6-unicast route-export to-evpn enable
on
nv set vrf vrf2 router bgp autonomous-system 650019
nv set vrf vrf2 router bgp enable on
nv set vrf vrf2 router bgp router-id 60.1.21.21
```

Sample Flat Files Configuration for L3 EVPN

The following is a sample flat files configuration which has L2 VNIs and L3 VNIs for EVPN bridging and symmetric routing on BlueField.

This file is located at `/etc/network/interfaces`:

```
auto lo
iface lo inet loopback
address 6.0.0.19/32
vxlan-local-tunnelip 6.0.0.19

auto vrf1
iface vrf1
address 2050:50:50:21::21/128
address 50.1.21.21/32
vrf-table auto

auto vrf2
iface vrf2
address 2060:60:60:21::21/128
address 60.1.21.21/32
vrf-table auto

auto p0_sf
iface p0_sf
alias alias p0_sf to leaf-21 swp3

auto p1_sf
```

```
iface p1_sf
alias alias p1_sf to leaf-22 swp3
```

```
auto pf0hpf_sf
iface pf0hpf_sf
alias alias pf0hpf_sf to host-211 ens2f0np0
bridge-access 111
```

```
auto pf0vf0_sf
iface pf0vf0_sf
alias alias pf0vf0_sf to host-211 ens2f0np0v0
bridge-access 112
```

```
auto pf1hpf_sf
iface pf1hpf_sf
alias alias pf1hpf_sf to host-211 ens2f1np1
bridge-access 213
```

```
auto pf1vf0_sf
iface pf1vf0_sf
alias alias pf1vf0_sf to host-211 ens2f1np0v0
bridge-access 214
```

```
auto vlan111
iface vlan111
address 2060:1:1:1::21/64
address 60.1.1.21/24
address-virtual 00:00:5e:00:01:01 2060:1:1:1::250/64 60.1.1.250/24
hwaddress 00:01:00:00:1e:03
vrf vrf2
vlan-raw-device br_default
vlan-id 111
```

```
auto vlan112
iface vlan112
address 2050:1:1:1::21/64
```

```
address 50.1.1.21/24
address-virtual 00:00:5e:00:01:01 2050:1:1:1::250/64 50.1.1.250/24
hwaddress 00:01:00:00:1e:03
vrf vrf1
vlan-raw-device br_default
vlan-id 112

auto vlan213
iface vlan213
address 2060:1:1:210::21/64
address 60.1.210.21/24
address-virtual 00:00:5e:00:01:01 2060:1:1:210::250/64 60.1.210.250/24
hwaddress 00:01:00:00:1e:03
vrf vrf2
vlan-raw-device br_default
vlan-id 213

auto vlan214
iface vlan214
address 2050:1:1:210::21/64
address 50.1.210.21/24
address-virtual 00:00:5e:00:01:01 2050:1:1:210::250/64 50.1.210.250/24
hwaddress 00:01:00:00:1e:03
vrf vrf1
vlan-raw-device br_default
vlan-id 214

auto vlan4058_I3
iface vlan4058_I3
vrf vrf1
vlan-raw-device br_default
address-virtual none
vlan-id 4058

auto vlan4059_I3
iface vlan4059_I3
```

```
vrf vrf2
vlan-raw-device br_default
address-virtual none
vlan-id 4059

auto vxlan48
iface vxlan48
bridge-vlan-vni-map 111=1000111 112=1000112 213=1000213 214=1000214
4058=104001 4059=104002
bridge-learning off

auto br_default
iface br_default
bridge-ports pf0hpf_sf pf0vf0_sf pf1hpf_sf pf1vf0_sf vxlan48
hwaddress 00:01:00:00:1e:03
bridge-vlan-aware yes
bridge-vids 111 112 213 214
bridge-pvid 1
```

FRR configuration is located at `/etc/frr/frr.conf`:

```
frr version 8.4.3
frr defaults datacenter
hostname doca-hbn-service-bf3-s05-1-ipmi
log syslog informational
no zebra nexthop kernel enable
service integrated-vtysh-config
!
vrf vrf1
vni 104001
exit-vrf
!
vrf vrf2
vni 104002
exit-vrf
```

```
!  
router bgp 650019  
  bgp router-id 6.0.0.19  
  bgp bestpath as-path multipath-relax  
  bgp bestpath compare-routerid  
  neighbor TOR_LEAF_SPINE peer-group  
  neighbor TOR_LEAF_SPINE advertisement-interval 0  
  neighbor TOR_LEAF_SPINE timers 3 9  
  neighbor TOR_LEAF_SPINE timers connect 10  
  neighbor p0_sf interface peer-group TOR_LEAF_SPINE  
  neighbor p0_sf remote-as external  
  neighbor p0_sf advertisement-interval 0  
  neighbor p0_sf timers 3 9  
  neighbor p0_sf timers connect 10  
  neighbor p1_sf interface peer-group TOR_LEAF_SPINE  
  neighbor p1_sf remote-as external  
  neighbor p1_sf advertisement-interval 0  
  neighbor p1_sf timers 3 9  
  neighbor p1_sf timers connect 10  
!  
  address-family ipv4 unicast  
    redistribute connected route-map ALLOW_LOBR  
    maximum-paths 16  
    maximum-paths ibgp 64  
  exit-address-family  
!  
  address-family l2vpn evpn  
    neighbor p0_sf activate  
    neighbor p1_sf activate  
    advertise-all-vni  
  exit-address-family  
exit  
!  
router bgp 650019 vrf vrf1  
  bgp router-id 50.1.21.21  
!
```

```
address-family ipv4 unicast
redistribute connected route-map ALLOW_VRF1
maximum-paths 64
maximum-paths ibgp 64
exit-address-family
!
address-family ipv6 unicast
redistribute connected route-map ALLOW_VRF1
maximum-paths 64
maximum-paths ibgp 64
exit-address-family
!
address-family l2vpn evpn
advertise ipv4 unicast
advertise ipv6 unicast
exit-address-family
exit
!
router bgp 650019 vrf vrf2
bgp router-id 60.1.21.21
!
address-family ipv4 unicast
redistribute connected route-map ALLOW_VRF2
maximum-paths 64
maximum-paths ibgp 64
exit-address-family
!
address-family ipv6 unicast
redistribute connected route-map ALLOW_VRF2
maximum-paths 64
maximum-paths ibgp 64
exit-address-family
!
address-family l2vpn evpn
advertise ipv4 unicast
advertise ipv6 unicast
```

```
exit-address-family
exit
!
route-map ALLOW_LOBR permit 10
match interface lo
exit
!
route-map ALLOW_LOBR permit 20
match interface br_default
exit
!
route-map ALLOW_VRF1 permit 10
match interface vrf1
exit
!
route-map ALLOW_VRF2 permit 10
match interface vrf2
exit
```

Multi-hop eBGP Peering for EVPN (Route Server in Symmetric EVPN Routing)

eBGP multi-hop peering for EVPN support in a route server-like role in EVPN topology, allows the deployment of EVPN on any cloud that supports IP transport.

R route servers and BF/HBN VTEPs are connected via the IP cloud. That is:

- Switches in the cloud provider need not be EVPN-aware
- Switches in the provider fabric provide IPv4 and IPv6 transport and do not have to support EVPN

Sample Route Server Configuration for EVPN

The following is a sample configuration of an Ubuntu server running FRR 9.0 stable, configured as EVPN route server and an HBN VTEP that is peering to two spine switches for IP connectivity and 3 Route servers for EVPN overlay control.

```

root@sn1:/home/cumulus# uname -a
Linux sn1 5.15.0-88-generic #98-Ubuntu SMP Mon Oct 2 15:18:56 UTC 2023 x86_64
x86_64 x86_64 GNU/Linux
root@sn1:/home/cumulus# dpkg -l frr
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
| | Name Version Architecture Description
+++-----
=====
ii frr 9.0.1-0~ubuntu22.04.1 amd64 FRRouting suite of internet protocols (BGP, OSPF,
IS-IS, ...)
root@sn1:/home/cumulus#

```

FRR configuration (frr.conf):

```

sn1# sh run
Building configuration...

Current configuration:
!
frr version 9.0.1
frr defaults datacenter
hostname sn1
no ip forwarding
no ipv6 forwarding
service integrated-vtysh-config
!
router bgp 4200065507
  bgp router-id 6.0.0.7
  timers bgp 60 180
  neighbor rclients peer-group
  neighbor rclients remote-as external

```



```
neighbor rclients ebgp-multihop 10
neighbor rclients update-source lo
neighbor rclients advertisement-interval 0
neighbor rclients timers 3 9
neighbor rclients timers connect 10
neighbor rcsuper peer-group
neighbor rcsuper remote-as external
neighbor rcsuper advertisement-interval 0
neighbor rcsuper timers 3 9
neighbor rcsuper timers connect 10
neighbor swp1 interface peer-group rcsuper
bgp listen range 6.0.0.0/24 peer-group rclients
!
address-family ipv4 unicast
redistribute connected
neighbor fabric route-map pass in
neighbor fabric route-map pass out
no neighbor rclients activate
maximum-paths 64
maximum-paths ibgp 64
exit-address-family
!
address-family l2vpn evpn
neighbor rclients activate
neighbor rcsuper activate
exit-address-family
exit
!
route-map pass permit 10
set community 11:11 additive
exit
!
end
sn1#
```

Interfaces configuration (/etc/network/interfaces):

```
root@sn1:/home/cumulus# ifquery -a
auto lo
iface lo inet loopback
address 6.0.0.7/32

auto lo
iface lo inet loopback

auto swp1
iface swp1

auto eth0
iface eth0
address 192.168.0.15/24
gateway 192.168.0.2

root@sn1:/home/cumulus#
```

Sample HBN configuration for deployments with EVPN Route Server

```
root@doca-hbn-service-bf2-s12-1-ipmi:/tmp# nv config show -o commands
nv set bridge domain br_default vlan 101 vni 10101
nv set bridge domain br_default vlan 102 vni 10102
nv set bridge domain br_default vlan 201 vni 10201
nv set bridge domain br_default vlan 202 vni 10202
nv set evpn enable on
nv set evpn route-advertise svi-ip off
nv set interface ilan3200 ip vrf internet1
nv set interface ilan3200 vlan 3200
nv set interface ilan3200,slan3201,vlan101-102,201-202,3001-3002 base-interface
br_default
nv set interface ilan3200,slan3201,vlan101-102,201-202,3001-3002 type svi
```

```
nv set interface lo ip address 6.0.0.13/32
nv set interface lo ip address 2001::13/128
nv set interface lo type loopback
nv set interface
p0_sf,p1_sf,pf0hpf_sf,pf0vf0_sf,pf0vf1_sf,pf0vf2_sf,pf0vf3_sf,pf0vf4_sf,pf0vf5_sf,pf0vf6_
type swp
nv set interface pf0vf0_sf bridge domain br_default access 101
nv set interface pf0vf1_sf bridge domain br_default access 102
nv set interface pf0vf2_sf bridge domain br_default access 201
nv set interface pf0vf3_sf bridge domain br_default access 202
nv set interface slan3201 ip vrf special1
nv set interface slan3201 vlan 3201
nv set interface vlan101 ip address 21.1.0.13/16
nv set interface vlan101 ip address 2020:0:1:1::13/64
nv set interface vlan101 ip vrr address 21.1.0.250/16
nv set interface vlan101 ip vrr address 2020:0:1:1::250/64
nv set interface vlan101 ip vrr mac-address 00:00:01:00:00:65
nv set interface vlan101 vlan 101
nv set interface vlan101-102,201-202 ip vrr enable on
nv set interface vlan101-102,3001 ip vrf tenant1
nv set interface vlan102 ip address 21.2.0.13/16
nv set interface vlan102 ip address 2020:0:1:2::13/64
nv set interface vlan102 ip vrr address 21.2.0.250/16
nv set interface vlan102 ip vrr address 2020:0:1:2::250/64
nv set interface vlan102 ip vrr mac-address 00:00:01:00:00:66
nv set interface vlan102 vlan 102
nv set interface vlan201 ip address 22.1.0.13/16
nv set interface vlan201 ip address 2020:0:2:1::13/64
nv set interface vlan201 ip vrr address 22.1.0.250/16
nv set interface vlan201 ip vrr address 2020:0:2:1::250/64
nv set interface vlan201 ip vrr mac-address 00:00:02:00:00:c9
nv set interface vlan201 vlan 201
nv set interface vlan201-202,3002 ip vrf tenant2
nv set interface vlan202 ip address 22.2.0.13/16
nv set interface vlan202 ip address 2020:0:2:2::13/64
nv set interface vlan202 ip vrr address 22.2.0.250/16
```

```
nv set interface vlan202 ip vrr address 2020:0:2:2::250/64
nv set interface vlan202 ip vrr mac-address 00:00:02:00:00:ca
nv set interface vlan202 vlan 202
nv set interface vlan3001 vlan 3001
nv set interface vlan3002 vlan 3002
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan source address 6.0.0.13
nv set platform
nv set router bgp autonomous-system 4200065011
nv set router bgp enable on
nv set router bgp router-id 6.0.0.13
nv set router vrr enable on
nv set system config snippet
nv set system global
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 6.0.0.7 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.7 type numbered
nv set vrf default router bgp neighbor 6.0.0.8 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.8 type numbered
nv set vrf default router bgp neighbor 6.0.0.9 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.9 type numbered
nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on

nv set vrf default router bgp peer-group fabric remote-as external
```

```
nv set vrf default router bgp peer-group rservers address-family ipv4-unicast enable
off
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn add-
path-tx off
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn enable
on
nv set vrf default router bgp peer-group rservers multihop-ttl 3
nv set vrf default router bgp peer-group rservers remote-as external
nv set vrf default router bgp peer-group rservers update-source lo
nv set vrf internet1 evpn enable on
nv set vrf internet1 evpn vni 42000
nv set vrf internet1 loopback ip address 8.1.0.13/32
nv set vrf internet1 loopback ip address 2008:0:1::13/64
nv set vrf internet1 router bgp address-family ipv4-unicast enable on
nv set vrf internet1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf internet1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf internet1 router bgp enable on
nv set vrf special1 evpn enable on
nv set vrf special1 evpn vni 42001
nv set vrf special1 loopback ip address 9.1.0.13/32
nv set vrf special1 loopback ip address 2009:0:1::13/64
nv set vrf special1 router bgp address-family ipv4-unicast enable on
nv set vrf special1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf special1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf special1 router bgp enable on
nv set vrf tenant1 evpn enable on
nv set vrf tenant1 evpn vni 30001
nv set vrf tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
```

```
nv set vrf tenant1 router bgp enable on
nv set vrf tenant1 router bgp router-id 6.0.0.13
nv set vrf tenant2 evpn enable on
nv set vrf tenant2 evpn vni 30002
nv set vrf tenant2 router bgp address-family ipv4-unicast enable on
nv set vrf tenant2 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant2 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf tenant2 router bgp enable on
nv set vrf tenant2 router bgp router-id 6.0.0.13
root@doca-hbn-service-bf2-s12-1-ipmi:/tmp#
```

Verifying BGP sessions in HBN:

```
doca-hbn-service-bf2-s12-1-ipmi# sh bgp sum

IPv4 Unicast Summary (VRF default):
BGP router identifier 6.0.0.13, local AS number 4200065011 vrf-id 0
BGP table version 20
RIB entries 21, using 4032 bytes of memory
Peers 2, using 40 KiB of memory
Peer groups 2, using 128 bytes of memory

Neighbor V AS MsgRcvd MsgSent TblVer InQ OutQ Up/Down State/PfxRcd PfxSnt
Desc
spine11(p0_sf) 4 65201 30617 30620 0 0 0 1d01h30m 9 11 N/A
spine12(p1_sf) 4 65201 30620 30623 0 0 0 1d01h30m 9 11 N/A

Total number of neighbors 2

IPv6 Unicast Summary (VRF default):
BGP router identifier 6.0.0.13, local AS number 4200065011 vrf-id 0
BGP table version 0
```

RIB entries 0, using 0 bytes of memory
Peers 2, using 40 KiB of memory
Peer groups 2, using 128 bytes of memory

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd	PfxSnt	Desc	
spine11	(p0_sf)	4	65201	30617	30620	0	0	0	1d01h30m	0	0	N/A
spine12	(p1_sf)	4	65201	30620	30623	0	0	0	1d01h30m	0	0	N/A

Total number of neighbors 2

L2VPN EVPN Summary (VRF default):

BGP router identifier 6.0.0.13, local AS number 4200065011 vrf-id 0
BGP table version 0
RIB entries 79, using 15 KiB of memory
Peers 3, using 60 KiB of memory
Peer groups 2, using 128 bytes of memory

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd	PfxSnt	Desc	
sn1	(6.0.0.7)	4	4200065507	31410	31231	0	0	0	00:27:51	69	95	N/A
sn2	(6.0.0.8)	4	4200065508	31169	31062	0	0	0	02:34:47	69	95	N/A
sn3	(6.0.0.9)	4	4200065509	31285	31059	0	0	0	02:34:47	69	95	N/A

Total number of neighbors 3

doca-hbn-service-bf2-s12-1-ipmi#

The command output shows that the HBN has BGP sessions with spine switches exchanging IPv4/IPv6 unicast. BGP sessions with route servers sn1, sn2, and sn3 only exchanging L2VPN EVPN AFI/SAFI.

Downstream VNI (DVNI)

Downstream VNI (symmetric EVPN route leaking) allows users to leak remote EVPN routes without having the source tenant VRF locally configured. A common use case is where upstream switches learn the L3VNI from downstream leaf switches and impose the learned L3VNI to the traffic VXLAN routed to the associated VRF. This eliminates the need to configure L3VNI-SVI interfaces on all leaf switches and enables shared service and hub-and-spoke scenarios .

To configure access to a shared service in a specific VRF, users must:

1. Configure route-target import statements, effectively leaking routes from remote tenants to the shared VRF.
2. Import shared VRF's route-target at the remote nodes.

The route target import or export statement takes the following format:

```
route-target import|export <asn>:<vni>
```

For example:

```
route-target import 65101:6000
```

For route target import statements, users can use `route-target import ANY:<vni>` for NVUE commands or `route-target import *:<vni>` in the `/etc/frr/frr.conf` file. ANY in NVUE commands or the asterisk (*) in the `/etc/frr/frr.conf` file use any ASN (a utonomous system number) as a wildcard.

The NVUE commands are as follows:

1. To configure a route import statement:

```
nv set vrf <vrf> router bgp route-import from-evpn route-target <asn>:<vni>
```

2. To configure a route export statement:

```
nv set vrf <vrf> router bgp route-export from-evpn route-target <asn>:<vni>
```


Important considerations when implementing DVNI configuration:

- EVPN symmetric mode supports downstream VNI with L3 VNIs and single VXLAN devices only
- You can configure multiple import and export route targets in a VRF
- You cannot leak (import) overlapping tenant prefixes into the same destination VRF

i Note

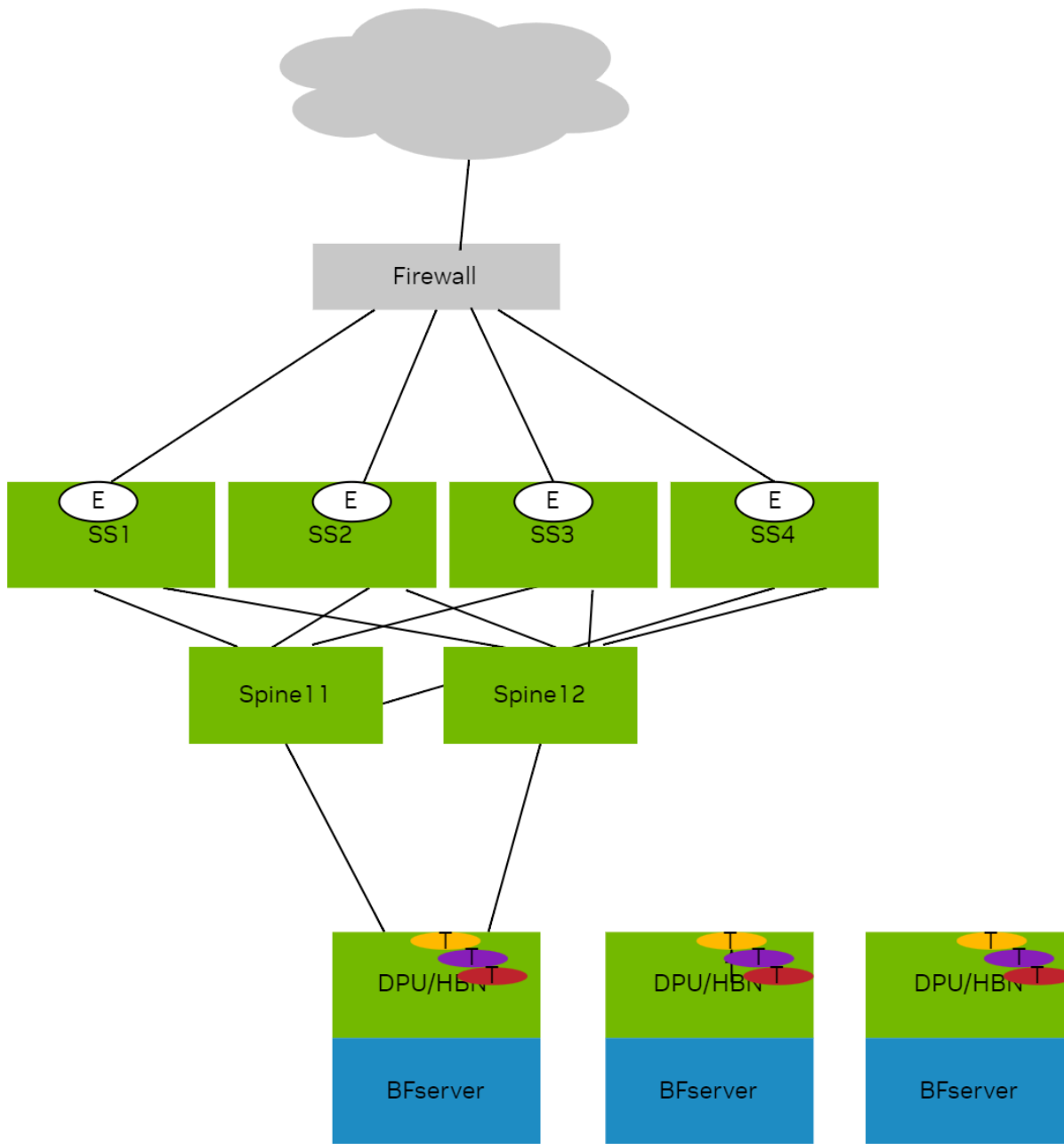
If symmetric EVPN configuration is using automatic import/export (which is often the case), when DVNI is configured, automatic import of tenant's VNI is disabled, isolating VRF from the tenant. User must specifically add 'route-target import auto' in such cases to avoid the problem.

DVNI Configurations for Shared Internet Service

Configuration example here considers a scenario where External/Internet connectivity is available via a firewall (FW), which is connected to a shared VRF (vrf external in this example).

The routes on super spine switches have external VRF configured in which the route-targets from remote tenants are imported.

On BlueField devices with HBN, a local tenant VRF imports route-target corresponding to the shared external VRF.



L3VNI:

Tenant	L3VNI	
tenant1	30001	On HBN VTEPs
tenant2	30002	On HBN VTEPs
tenant3	30003	On HBN VTEPs
tenant4	30004	On HBN VTEPs

Tenant	L3VNI	
tenant5	30005	On HBN VTEPs
tenant6	30006	On HBN VTEPs
external	60000	Configured on superspines and connects to external world

On BlueField devices with HBN, every tenant VRF on HBN one must import VNI of shared external VRF:

```

nv set vrf tenant1 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant1 router bgp route-import from-evpn route-target auto
nv set vrf tenant2 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant2 router bgp route-import from-evpn route-target auto
nv set vrf tenant3 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant3 router bgp route-import from-evpn route-target auto
nv set vrf tenant4 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant4 router bgp route-import from-evpn route-target auto
nv set vrf tenant5 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant5 router bgp route-import from-evpn route-target auto
nv set vrf tenant6 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant6 router bgp route-import from-evpn route-target auto
root@doca-hbn-service-bf3-s06-1-ipmi:/tmp#

```

On super spine switches (SS1 in this example), every remote tenant VRF that needs access to shared services has to be leaked to the shared external VRF.

```

nv set vrf external router bgp route-import from-evpn route-target ANY:30001
nv set vrf external router bgp route-import from-evpn route-target ANY:30002
nv set vrf external router bgp route-import from-evpn route-target ANY:30003
nv set vrf external router bgp route-import from-evpn route-target ANY:30004
nv set vrf external router bgp route-import from-evpn route-target ANY:30005
nv set vrf external router bgp route-import from-evpn route-target ANY:30006
nv set vrf external router bgp route-import from-evpn route-target auto
root@superspine1:mgmt:/home/cumulus#

```

All super spines in this case need this configuration.

DVNI Leaked Routes in VRF Table of HBN

Info

Each super spine here is advertising reachability providing 4-way overlay ECMP.

Kernel table for all tenant VRFs, showing the imported shared service:

```
root@doca-hbn-service-bf3-s06-1-ipmi:/tmp# ip -4 route show table all 6.0.0.4/32
6.0.0.4 table tenant1 proto bgp metric 20
nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48
weight 1 onlink
nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.13 ttl 0 tos 0 via 6.0.0.13 dev vxlan48
weight 1 onlink
nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.14 ttl 0 tos 0 via 6.0.0.14 dev vxlan48
weight 1 onlink
nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.15 ttl 0 tos 0 via 6.0.0.15 dev vxlan48
weight 1 onlink
6.0.0.4 table tenant2 proto bgp metric 20
nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48
weight 1 onlink
nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.13 ttl 0 tos 0 via 6.0.0.13 dev vxlan48
weight 1 onlink
nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.14 ttl 0 tos 0 via 6.0.0.14 dev vxlan48
weight 1 onlink
nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.15 ttl 0 tos 0 via 6.0.0.15 dev vxlan48
weight 1 onlink
6.0.0.4 table tenant3 proto bgp metric 20
```

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.13 ttl 0 tos 0 via 6.0.0.13 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.14 ttl 0 tos 0 via 6.0.0.14 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.15 ttl 0 tos 0 via 6.0.0.15 dev vxlan48
weight 1 onlink

6.0.0.4 table tenant4 proto bgp metric 20

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.13 ttl 0 tos 0 via 6.0.0.13 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.14 ttl 0 tos 0 via 6.0.0.14 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.15 ttl 0 tos 0 via 6.0.0.15 dev vxlan48
weight 1 onlink

6.0.0.4 table tenant5 proto bgp metric 20

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.13 ttl 0 tos 0 via 6.0.0.13 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.14 ttl 0 tos 0 via 6.0.0.14 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.15 ttl 0 tos 0 via 6.0.0.15 dev vxlan48
weight 1 onlink

6.0.0.4 table tenant6 proto bgp metric 20

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.13 ttl 0 tos 0 via 6.0.0.13 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.14 ttl 0 tos 0 via 6.0.0.14 dev vxlan48
weight 1 onlink

nexthop encap ip id 60000 src 0.0.0.0 dst 6.0.0.15 ttl 0 tos 0 via 6.0.0.15 dev vxlan48
weight 1 onlink

```
root@doca-hbn-service-bf3-s06-1-ipmi:/tmp#
```

FRR RIB table:

```
root@doca-hbn-service-bf3-s06-1-ipmi:/tmp# vtysh
```

```
Hello, this is FRRouting (version 8.4.3).
```

```
Copyright 1996-2005 Kunihiro Ishiguro, et al.
```

```
doca-hbn-service-bf3-s06-1-ipmi# sh ip route vrf tenant1
```

```
Codes: K - kernel route, C - connected, S - static, R - RIP,
```

```
O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
```

```
T - Table, A - Babel, D - SHARP, F - PBR, f - OpenFabric,
```

```
Z - FRR,
```

```
> - selected route, * - FIB route, q - queued, r - rejected, b - backup
```

```
t - trapped, o - offload failure
```

```
VRF tenant1:
```

```
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:36
```

```
B>* 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,  
00:05:38
```

```
* via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:38
```

```
* via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:38
```

```
* via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:38
```

```
B>* 6.6.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,  
00:05:38
```

```
B>* 6.6.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,  
00:05:38
```

```
B>* 6.6.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,  
00:05:38
```

```
B>* 6.6.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,  
00:05:38
```

```
B>* 7.1.0.6/32 [20/0] via 6.0.0.6, vlan4052_I3 onlink, weight 1, 00:05:37
```

```
C>* 7.1.0.16/32 is directly connected, tenant1, 00:10:36
```

```
B>* 7.1.0.18/32 [20/0] via 6.0.0.18, vlan4052_I3 onlink, weight 1, 00:05:37
```

```

B>* 7.1.0.20/32 [20/0] via 6.0.0.20, vlan4052_l3 onlink, weight 1, 00:05:37
C>* 21.1.0.0/16 is directly connected, vlan101, 00:10:36
C * 21.1.0.0/16 [0/1024] is directly connected, vlan101-v0, 00:10:36
C * 21.2.0.0/16 [0/1024] is directly connected, vlan102-v0, 00:10:36
C>* 21.2.0.0/16 is directly connected, vlan102, 00:10:36
B>* 101.12.4.0/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:38
B>* 101.13.4.0/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:38
B>* 101.14.4.0/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:38
B>* 101.15.4.0/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:38

```

```

doca-hbn-service-bf3-s06-1-ipmi# sh ip route vrf all

```

Codes: K - kernel route, C - connected, S - static, R - RIP,

O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,

T - Table, A - Babel, D - SHARP, F - PBR, f - OpenFabric,

Z - FRR,

> - selected route, * - FIB route, q - queued, r - rejected, b - backup

t - trapped, o - offload failure

VRF default:

```

B>* 6.0.0.6/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:06:47
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:06:47
B>* 6.0.0.7/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:05:48
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:05:48
B>* 6.0.0.8/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:05:38
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:05:38
B>* 6.0.0.9/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:05:28
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:05:28
B>* 6.0.0.10/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:06:49
B>* 6.0.0.11/32 [20/0] via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:06:47
B>* 6.0.0.12/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:06:47
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:06:47
B>* 6.0.0.13/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:06:47
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:06:47

```

```
B>* 6.0.0.14/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:06:47
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:06:47
B>* 6.0.0.15/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:06:47
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:06:47
C>* 6.0.0.16/32 is directly connected, lo, 00:10:42
B>* 6.0.0.18/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:06:47
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:06:47
B>* 6.0.0.20/32 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:06:47
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:06:47
B>* 192.168.0.0/24 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:05:48
* via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:05:48
```

VRF internet1:

```
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 8.1.0.6/32 [20/0] via 6.0.0.6, vlan4004_I3 onlink, weight 1, 00:05:43
C>* 8.1.0.16/32 is directly connected, internet1, 00:10:42
B>* 8.1.0.18/32 [20/0] via 6.0.0.18, vlan4004_I3 onlink, weight 1, 00:05:43
B>* 8.1.0.20/32 [20/0] via 6.0.0.20, vlan4004_I3 onlink, weight 1, 00:05:43
```

VRF mgmt:

```
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
C>* 10.88.0.0/16 is directly connected, eth0, 00:10:42
```

VRF special1:

```
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 9.1.0.6/32 [20/0] via 6.0.0.6, vlan4033_I3 onlink, weight 1, 00:05:43
C>* 9.1.0.16/32 is directly connected, special1, 00:10:42
B>* 9.1.0.18/32 [20/0] via 6.0.0.18, vlan4033_I3 onlink, weight 1, 00:05:43
B>* 9.1.0.20/32 [20/0] via 6.0.0.20, vlan4033_I3 onlink, weight 1, 00:05:43
```

VRF tenant1:

```
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
* via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
```



```

* via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.6.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 7.1.0.6/32 [20/0] via 6.0.0.6, vlan4052_l3 onlink, weight 1, 00:05:43
C>* 7.1.0.16/32 is directly connected, tenant1, 00:10:42
B>* 7.1.0.18/32 [20/0] via 6.0.0.18, vlan4052_l3 onlink, weight 1, 00:05:43
B>* 7.1.0.20/32 [20/0] via 6.0.0.20, vlan4052_l3 onlink, weight 1, 00:05:43
C>* 21.1.0.0/16 is directly connected, vlan101, 00:10:42
C * 21.1.0.0/16 [0/1024] is directly connected, vlan101-v0, 00:10:42
C * 21.2.0.0/16 [0/1024] is directly connected, vlan102-v0, 00:10:42
C>* 21.2.0.0/16 is directly connected, vlan102, 00:10:42
B>* 101.12.4.0/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.13.4.0/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.14.4.0/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.15.4.0/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44

```

VRF tenant2:

```

K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
* via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.6.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44

```

```
B>* 6.6.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 7.2.0.6/32 [20/0] via 6.0.0.6, vlan4037_I3 onlink, weight 1, 00:05:43
C>* 7.2.0.16/32 is directly connected, tenant2, 00:10:42
B>* 7.2.0.18/32 [20/0] via 6.0.0.18, vlan4037_I3 onlink, weight 1, 00:05:43
B>* 7.2.0.20/32 [20/0] via 6.0.0.20, vlan4037_I3 onlink, weight 1, 00:05:43
C * 22.1.0.0/16 [0/1024] is directly connected, vlan201-v0, 00:10:42
C>* 22.1.0.0/16 is directly connected, vlan201, 00:10:42
C * 22.2.0.0/16 [0/1024] is directly connected, vlan202-v0, 00:10:42
C>* 22.2.0.0/16 is directly connected, vlan202, 00:10:42
B>* 101.12.4.0/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.13.4.0/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.14.4.0/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.15.4.0/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
```

VRF tenant3:

```
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
* via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.6.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
```

```
B>* 6.6.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 7.3.0.6/32 [20/0] via 6.0.0.6, vlan4022_l3 onlink, weight 1, 00:05:43
C>* 7.3.0.16/32 is directly connected, tenant3, 00:10:42
B>* 7.3.0.18/32 [20/0] via 6.0.0.18, vlan4022_l3 onlink, weight 1, 00:05:43
B>* 7.3.0.20/32 [20/0] via 6.0.0.20, vlan4022_l3 onlink, weight 1, 00:05:43
C>* 23.17.0.0/16 is directly connected, pf0vf4_sf.3, 00:10:42
B>* 23.19.0.0/16 [20/0] via 6.0.0.18, vlan4022_l3 onlink, weight 1, 00:05:43
B>* 23.21.0.0/16 [20/0] via 6.0.0.20, vlan4022_l3 onlink, weight 1, 00:05:43
B>* 101.12.4.0/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.13.4.0/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.14.4.0/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.15.4.0/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
```

VRF tenant4:

```
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
* via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.6.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 7.4.0.6/32 [20/0] via 6.0.0.6, vlan4017_l3 onlink, weight 1, 00:05:43
C>* 7.4.0.16/32 is directly connected, tenant4, 00:10:42
B>* 7.4.0.18/32 [20/0] via 6.0.0.18, vlan4017_l3 onlink, weight 1, 00:05:43
```

```
B>* 7.4.0.20/32 [20/0] via 6.0.0.20, vlan4017_I3 onlink, weight 1, 00:05:43
C>* 24.17.0.0/16 is directly connected, pf0vf4_sf.4, 00:10:42
B>* 24.19.0.0/16 [20/0] via 6.0.0.18, vlan4017_I3 onlink, weight 1, 00:05:43
B>* 24.21.0.0/16 [20/0] via 6.0.0.20, vlan4017_I3 onlink, weight 1, 00:05:43
B>* 101.12.4.0/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.13.4.0/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.14.4.0/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.15.4.0/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
```

VRF tenant5:

```
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
* via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.6.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 7.5.0.6/32 [20/0] via 6.0.0.6, vlan4046_I3 onlink, weight 1, 00:05:43
C>* 7.5.0.16/32 is directly connected, tenant5, 00:10:42
B>* 7.5.0.18/32 [20/0] via 6.0.0.18, vlan4046_I3 onlink, weight 1, 00:05:43
B>* 7.5.0.20/32 [20/0] via 6.0.0.20, vlan4046_I3 onlink, weight 1, 00:05:43
C>* 25.17.0.0/16 is directly connected, pf0vf4_sf.5, 00:10:42
B>* 25.19.0.0/16 [20/0] via 6.0.0.18, vlan4046_I3 onlink, weight 1, 00:05:43
B>* 25.21.0.0/16 [20/0] via 6.0.0.20, vlan4046_I3 onlink, weight 1, 00:05:43
```

```
B>* 101.12.4.0/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.13.4.0/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.14.4.0/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.15.4.0/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
```

VRF tenant6:

```
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
* via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.6.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 6.6.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 7.6.0.6/32 [20/0] via 6.0.0.6, vlan4041_I3 onlink, weight 1, 00:05:43
C>* 7.6.0.16/32 is directly connected, tenant6, 00:10:42
B>* 7.6.0.18/32 [20/0] via 6.0.0.18, vlan4041_I3 onlink, weight 1, 00:05:43
B>* 7.6.0.20/32 [20/0] via 6.0.0.20, vlan4041_I3 onlink, weight 1, 00:05:43
C>* 26.17.0.0/16 is directly connected, pf0vf4_sf.6, 00:10:42
B>* 26.19.0.0/16 [20/0] via 6.0.0.18, vlan4041_I3 onlink, weight 1, 00:05:43
B>* 26.21.0.0/16 [20/0] via 6.0.0.20, vlan4041_I3 onlink, weight 1, 00:05:43
B>* 101.12.4.0/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.13.4.0/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
```

```
B>* 101.14.4.0/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
B>* 101.15.4.0/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1,
00:05:44
doca-hbn-service-bf3-s06-1-ipmi#
```

DVNI Debugging

BGP/Zebra debug:

```
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf
tenant1: import evpn prefix [5]:[0]:[32]:[6.0.0.4] parent 0xaaaafda63a90 flags 0x410
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new
pi dest 0xaaaafe524650 (l 2) pi 0xaaaafe5ae400 (l 1, f 0x4010)
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf
tenant2: import evpn prefix [5]:[0]:[32]:[6.0.0.4] parent 0xaaaafda63a90 flags 0x410
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new
pi dest 0xaaaafe51c420 (l 2) pi 0xaaaafe55d230 (l 1, f 0x4010)
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf
tenant3: import evpn prefix [5]:[0]:[32]:[6.0.0.4] parent 0xaaaafda63a90 flags 0x410
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new
pi dest 0xaaaafe51a670 (l 2) pi 0xaaaafe674820 (l 1, f 0x4010)
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf
tenant4: import evpn prefix [5]:[0]:[32]:[6.0.0.4] parent 0xaaaafda63a90 flags 0x410
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new
pi dest 0xaaaafe519fb0 (l 2) pi 0xaaaafe675e40 (l 1, f 0x4010)
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf
tenant5: import evpn prefix [5]:[0]:[32]:[6.0.0.4] parent 0xaaaafda63a90 flags 0x410
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new
pi dest 0xaaaafe55ae50 (l 2) pi 0xaaaafe5482f0 (l 1, f 0x4010)
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf
tenant6: import evpn prefix [5]:[0]:[32]:[6.0.0.4] parent 0xaaaafda63a90 flags 0x410
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new
pi dest 0xaaaafdaf3590 (l 2) pi 0xaaaafe48fbf0 (l 1, f 0x4010)
```



```

{
"dest-vtep": "6.0.0.15",
"dest-mac": "44:38:39:f0:00:15",
"is-dmac-null": 0,
"ref-cnt": 36
}
]
}
}
]
}
}root@doca-hbn-service-bf3-s06-1-ipmi:/tmp#

```

Sample DVNI Configuration

HBN configuration example for BlueField devices:

```

root@doca-hbn-service-bf3-s06-1-ipmi:/tmp# nv config show -o commands
nv set bridge domain br_default vlan 101 vni 10101
nv set bridge domain br_default vlan 102 vni 10102
nv set bridge domain br_default vlan 201 vni 10201
nv set bridge domain br_default vlan 202 vni 10202
nv set evpn enable on
nv set evpn route-advertise svi-ip off
nv set interface ilan3200 ip vrf internet1
nv set interface ilan3200 vlan 3200
nv set interface ilan3200,slan3201,vlan101-102,201-202,3001-3006 base-interface
br_default
nv set interface ilan3200,slan3201,vlan101-102,201-202,3001-3006 type svi
nv set interface lo ip address 6.0.0.16/32
nv set interface lo ip address 2001::16/128
nv set interface lo type loopback
nv set interface
p0_sf,p1_sf,pf0hpf_sf,pf0vf0_sf,pf0vf1_sf,pf0vf2_sf,pf0vf3_sf,pf0vf4_sf,pf0vf5_sf,pf0vf6_
type swp

```



```
nv set interface pf0vf0_sf bridge domain br_default access 101
nv set interface pf0vf1_sf bridge domain br_default access 102
nv set interface pf0vf2_sf bridge domain br_default access 201
nv set interface pf0vf3_sf bridge domain br_default access 202
nv set interface pf0vf4_sf.3 ip address 23.17.0.16/16
nv set interface pf0vf4_sf.3 ip address 2020:0:3:17::16/64
nv set interface pf0vf4_sf.3 vlan 3
nv set interface pf0vf4_sf.3,vlan3003 ip vrf tenant3
nv set interface pf0vf4_sf.3-6 base-interface pf0vf4_sf
nv set interface pf0vf4_sf.3-6 type sub
nv set interface pf0vf4_sf.4 ip address 24.17.0.16/16
nv set interface pf0vf4_sf.4 ip address 2020:0:4:17::16/64
nv set interface pf0vf4_sf.4 vlan 4
nv set interface pf0vf4_sf.4,vlan3004 ip vrf tenant4
nv set interface pf0vf4_sf.5 ip address 25.17.0.16/16
nv set interface pf0vf4_sf.5 ip address 2020:0:5:17::16/64
nv set interface pf0vf4_sf.5 vlan 5
nv set interface pf0vf4_sf.5,vlan3005 ip vrf tenant5
nv set interface pf0vf4_sf.6 ip address 26.17.0.16/16
nv set interface pf0vf4_sf.6 ip address 2020:0:6:17::16/64
nv set interface pf0vf4_sf.6 vlan 6
nv set interface pf0vf4_sf.6,vlan3006 ip vrf tenant6
nv set interface slan3201 ip vrf special1
nv set interface slan3201 vlan 3201
nv set interface vlan101 ip address 21.1.0.16/16
nv set interface vlan101 ip address 2020:0:1:1::16/64
nv set interface vlan101 ip vrr address 21.1.0.250/16
nv set interface vlan101 ip vrr address 2020:0:1:1::250/64
nv set interface vlan101 ip vrr mac-address 00:00:01:00:00:65
nv set interface vlan101 vlan 101
nv set interface vlan101-102,201-202 ip vrr enable on
nv set interface vlan101-102,3001 ip vrf tenant1
nv set interface vlan102 ip address 21.2.0.16/16
nv set interface vlan102 ip address 2020:0:1:2::16/64
nv set interface vlan102 ip vrr address 21.2.0.250/16
nv set interface vlan102 ip vrr address 2020:0:1:2::250/64
```

```
nv set interface vlan102 ip vrr mac-address 00:00:01:00:00:66
nv set interface vlan102 vlan 102
nv set interface vlan201 ip address 22.1.0.16/16
nv set interface vlan201 ip address 2020:0:2:1::16/64
nv set interface vlan201 ip vrr address 22.1.0.250/16
nv set interface vlan201 ip vrr address 2020:0:2:1::250/64
nv set interface vlan201 ip vrr mac-address 00:00:02:00:00:c9
nv set interface vlan201 vlan 201
nv set interface vlan201-202,3002 ip vrf tenant2
nv set interface vlan202 ip address 22.2.0.16/16
nv set interface vlan202 ip address 2020:0:2:2::16/64
nv set interface vlan202 ip vrr address 22.2.0.250/16
nv set interface vlan202 ip vrr address 2020:0:2:2::250/64
nv set interface vlan202 ip vrr mac-address 00:00:02:00:00:ca
nv set interface vlan202 vlan 202
nv set interface vlan3001 vlan 3001
nv set interface vlan3002 vlan 3002
nv set interface vlan3003 vlan 3003
nv set interface vlan3004 vlan 3004
nv set interface vlan3005 vlan 3005
nv set interface vlan3006 vlan 3006
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan source address 6.0.0.16
nv set platform
nv set router bgp autonomous-system 65011
nv set router bgp enable on
nv set router bgp router-id 6.0.0.16
nv set router vrr enable on
nv set system config snippet
nv set system global
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
```

```
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 6.0.0.7 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.7 type numbered
nv set vrf default router bgp neighbor 6.0.0.8 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.8 type numbered
nv set vrf default router bgp neighbor 6.0.0.9 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.9 type numbered
nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric bfd detect-multiplier 3
nv set vrf default router bgp peer-group fabric bfd enable on
nv set vrf default router bgp peer-group fabric bfd min-rx-interval 1000
nv set vrf default router bgp peer-group fabric bfd min-tx-interval 1000
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group rservers address-family ipv4-unicast enable
off
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn add-
path-tx off
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn enable
on
nv set vrf default router bgp peer-group rservers multihop-ttl 10
nv set vrf default router bgp peer-group rservers remote-as external
nv set vrf default router bgp peer-group rservers update-source lo
nv set vrf internet1 evpn enable on
nv set vrf internet1 evpn vni 42000
nv set vrf internet1 loopback ip address 8.1.0.16/32
nv set vrf internet1 loopback ip address 2008:0:1::16/64
nv set vrf internet1 router bgp address-family ipv4-unicast enable on
nv set vrf internet1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf internet1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
```

```
nv set vrf internet1 router bgp address-family ipv6-unicast enable on
nv set vrf internet1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf internet1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf internet1 router bgp enable on
nv set vrf special1 evpn enable on
nv set vrf special1 evpn vni 42001
nv set vrf special1 loopback ip address 9.1.0.16/32
nv set vrf special1 loopback ip address 2009:0:1::16/64
nv set vrf special1 router bgp address-family ipv4-unicast enable on
nv set vrf special1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf special1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf special1 router bgp address-family ipv6-unicast enable on
nv set vrf special1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf special1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf special1 router bgp enable on
nv set vrf tenant1 evpn enable on
nv set vrf tenant1 evpn vni 30001
nv set vrf tenant1 loopback ip address 7.1.0.16/32
nv set vrf tenant1 loopback ip address 2007:0:1::16/64
nv set vrf tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf tenant1 router bgp enable on
```

```
nv set vrf tenant1 router bgp neighbor 21.1.0.17 peer-group hostgroup
nv set vrf tenant1 router bgp neighbor 21.1.0.17 type numbered
nv set vrf tenant1 router bgp peer-group hostgroup address-family ipv4-unicast
enable on
nv set vrf tenant1 router bgp peer-group hostgroup address-family ipv6-unicast
enable on
nv set vrf tenant1 router bgp peer-group hostgroup remote-as external
nv set vrf tenant1 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant1 router bgp route-import from-evpn route-target auto
nv set vrf tenant1 router bgp router-id 6.0.0.16
nv set vrf tenant2 evpn enable on
nv set vrf tenant2 evpn vni 30002
nv set vrf tenant2 loopback ip address 7.2.0.16/32
nv set vrf tenant2 loopback ip address 2007:0:2::16/64
nv set vrf tenant2 router bgp address-family ipv4-unicast enable on
nv set vrf tenant2 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant2 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf tenant2 router bgp address-family ipv6-unicast enable on
nv set vrf tenant2 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf tenant2 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf tenant2 router bgp enable on
nv set vrf tenant2 router bgp neighbor 22.1.0.17 peer-group hostgroup
nv set vrf tenant2 router bgp neighbor 22.1.0.17 type numbered
nv set vrf tenant2 router bgp peer-group hostgroup address-family ipv4-unicast
enable on
nv set vrf tenant2 router bgp peer-group hostgroup address-family ipv6-unicast
enable on
nv set vrf tenant2 router bgp peer-group hostgroup remote-as external
nv set vrf tenant2 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant2 router bgp route-import from-evpn route-target auto
nv set vrf tenant2 router bgp router-id 6.0.0.16
nv set vrf tenant3 evpn enable on
```

```
nv set vrf tenant3 evpn vni 30003
nv set vrf tenant3 loopback ip address 7.3.0.16/32
nv set vrf tenant3 loopback ip address 2007:0:3::16/64
nv set vrf tenant3 router bgp address-family ipv4-unicast enable on
nv set vrf tenant3 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant3 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf tenant3 router bgp address-family ipv6-unicast enable on
nv set vrf tenant3 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf tenant3 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf tenant3 router bgp enable on
nv set vrf tenant3 router bgp neighbor 23.17.0.17 peer-group hostgroup
nv set vrf tenant3 router bgp neighbor 23.17.0.17 type numbered
nv set vrf tenant3 router bgp peer-group hostgroup address-family ipv4-unicast
enable on
nv set vrf tenant3 router bgp peer-group hostgroup address-family ipv6-unicast
enable on
nv set vrf tenant3 router bgp peer-group hostgroup remote-as external
nv set vrf tenant3 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant3 router bgp route-import from-evpn route-target auto
nv set vrf tenant3 router bgp router-id 6.0.0.16
nv set vrf tenant3 table auto
nv set vrf tenant4 evpn enable on
nv set vrf tenant4 evpn vni 30004
nv set vrf tenant4 loopback ip address 7.4.0.16/32
nv set vrf tenant4 loopback ip address 2007:0:4::16/64
nv set vrf tenant4 router bgp address-family ipv4-unicast enable on
nv set vrf tenant4 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant4 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf tenant4 router bgp address-family ipv6-unicast enable on
```

```
nv set vrf tenant4 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf tenant4 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf tenant4 router bgp enable on
nv set vrf tenant4 router bgp neighbor 24.17.0.17 peer-group hostgroup
nv set vrf tenant4 router bgp neighbor 24.17.0.17 type numbered
nv set vrf tenant4 router bgp peer-group hostgroup address-family ipv4-unicast
enable on
nv set vrf tenant4 router bgp peer-group hostgroup address-family ipv6-unicast
enable on
nv set vrf tenant4 router bgp peer-group hostgroup remote-as external
nv set vrf tenant4 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant4 router bgp route-import from-evpn route-target auto
nv set vrf tenant4 router bgp router-id 6.0.0.16
nv set vrf tenant4 table auto
nv set vrf tenant5 evpn enable on
nv set vrf tenant5 evpn vni 30005
nv set vrf tenant5 loopback ip address 7.5.0.16/32
nv set vrf tenant5 loopback ip address 2007:0:5::16/64
nv set vrf tenant5 router bgp address-family ipv4-unicast enable on
nv set vrf tenant5 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant5 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf tenant5 router bgp address-family ipv6-unicast enable on
nv set vrf tenant5 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf tenant5 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf tenant5 router bgp enable on
nv set vrf tenant5 router bgp neighbor 25.17.0.17 peer-group hostgroup
nv set vrf tenant5 router bgp neighbor 25.17.0.17 type numbered
nv set vrf tenant5 router bgp peer-group hostgroup address-family ipv4-unicast
enable on
```

```
nv set vrf tenant5 router bgp peer-group hostgroup address-family ipv6-unicast
enable on
nv set vrf tenant5 router bgp peer-group hostgroup remote-as external
nv set vrf tenant5 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant5 router bgp route-import from-evpn route-target auto
nv set vrf tenant5 router bgp router-id 6.0.0.16
nv set vrf tenant5 table auto
nv set vrf tenant6 evpn enable on
nv set vrf tenant6 evpn vni 30006
nv set vrf tenant6 loopback ip address 7.6.0.16/32
nv set vrf tenant6 loopback ip address 2007:0:6::16/64
nv set vrf tenant6 router bgp address-family ipv4-unicast enable on
nv set vrf tenant6 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant6 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf tenant6 router bgp address-family ipv6-unicast enable on
nv set vrf tenant6 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf tenant6 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf tenant6 router bgp enable on
nv set vrf tenant6 router bgp neighbor 26.17.0.17 peer-group hostgroup
nv set vrf tenant6 router bgp neighbor 26.17.0.17 type numbered
nv set vrf tenant6 router bgp peer-group hostgroup address-family ipv4-unicast
enable on
nv set vrf tenant6 router bgp peer-group hostgroup address-family ipv6-unicast
enable on
nv set vrf tenant6 router bgp peer-group hostgroup remote-as external
nv set vrf tenant6 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant6 router bgp route-import from-evpn route-target auto
nv set vrf tenant6 router bgp router-id 6.0.0.16
nv set vrf tenant6 table auto
root@doca-hbn-service-bf3-s06-1-ipmi:/tmp#
```

SS1 switch configuration example:


```
root@superspine1:mgmt:/home/cumulus# nv config show -o commands
nv set bridge domain br_default vlan 101 vni 10101
nv set bridge domain br_default vlan 102 vni 10102
nv set bridge domain br_default vlan 201 vni 10201
nv set bridge domain br_default vlan 202 vni 10202
nv set evpn enable on
nv set interface eth0 ip address 192.168.0.15/24
nv set interface eth0 ip gateway 192.168.0.2
nv set interface eth0 type eth
nv set interface lo ip address 6.0.0.12/32
nv set interface lo ip address 2001::12/128
nv set interface lo type loopback
nv set interface swp1-6 type swp
nv set interface swp6 ip address 101.12.4.12/24
nv set interface swp6 ip address 2101:12::4:12/112
nv set interface swp6 ip vrf external
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan source address 6.0.0.12
nv set platform
nv set router bgp autonomous-system 65300
nv set router bgp enable on
nv set router bgp router-id 6.0.0.12
nv set system config snippet
nv set system global system-mac 44:38:39:f0:00:12
nv set system hostname superspine1
nv set system ssh-server permit-root-login enabled
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor swp1 peer-group fabric
nv set vrf default router bgp neighbor swp1 type unnumbered
```

```
nv set vrf default router bgp neighbor swp2 peer-group fabric
nv set vrf default router bgp neighbor swp2 type unnumbered
nv set vrf default router bgp neighbor swp3 peer-group rservers
nv set vrf default router bgp neighbor swp3 type unnumbered
nv set vrf default router bgp neighbor swp4 peer-group rservers
nv set vrf default router bgp neighbor swp4 type unnumbered
nv set vrf default router bgp neighbor swp5 peer-group rservers
nv set vrf default router bgp neighbor swp5 type unnumbered
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric bfd detect-multiplier 3
nv set vrf default router bgp peer-group fabric bfd enable on
nv set vrf default router bgp peer-group fabric bfd min-rx-interval 1000
nv set vrf default router bgp peer-group fabric bfd min-tx-interval 1000
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group rservers address-family ipv4-unicast enable
on
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn add-
path-tx off
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn enable
on
nv set vrf default router bgp peer-group rservers remote-as external
nv set vrf external evpn enable on
nv set vrf external evpn vni 60000
nv set vrf external loopback ip address 6.6.0.12/32
nv set vrf external loopback ip address 2006:0:6::12/64
nv set vrf external router bgp address-family ipv4-unicast enable on
nv set vrf external router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf external router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf external router bgp address-family ipv6-unicast enable on
nv set vrf external router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf external router bgp address-family ipv6-unicast route-export to-evpn
enable on
```

```

nv set vrf external router bgp address-family l2vpn-evpn enable on
nv set vrf external router bgp enable on
nv set vrf external router bgp neighbor swp6 peer-group peer-group-fw
nv set vrf external router bgp neighbor swp6 type unnumbered
nv set vrf external router bgp peer-group peer-group-fw address-family ipv4-unicast
enable on
nv set vrf external router bgp peer-group peer-group-fw address-family ipv6-unicast
enable on
nv set vrf external router bgp peer-group peer-group-fw remote-as external
nv set vrf external router bgp route-import from-evpn route-target ANY:30001
nv set vrf external router bgp route-import from-evpn route-target ANY:30002
nv set vrf external router bgp route-import from-evpn route-target ANY:30003
nv set vrf external router bgp route-import from-evpn route-target ANY:30004
nv set vrf external router bgp route-import from-evpn route-target ANY:30005
nv set vrf external router bgp route-import from-evpn route-target ANY:30006
nv set vrf external router bgp route-import from-evpn route-target auto
root@superspine1:mgmt:/home/cumulus#

```

Gateway Application Using Downstream VNI and Subinterface

A DPU running the HBN service can be deployed in the role of a border gateway using a combination of HBN features, specifically, EVPN symmetric routing, downstream VNI, VRF route-leaking, and VLAN sub-interfaces. Such a border gateway can do the northbound traffic handoff (to external networks or the Internet) for one or more tenants. In this gateway configuration, the BlueField's uplinks must carry both the tenant traffic which would be in the "overlay" and VXLAN-encapsulated, as well as traffic to and from the external network or Internet, which would be direct-routed in the "underlay". This is accomplished by configuring and running VXLAN-EVPN on the uplink interfaces while configuring and using additional VLAN sub-interfaces on those same uplinks for the traffic to and from external networks. These VLAN sub-interfaces would be configured into an Internet or external VRF for separation from the VXLAN-encapsulated traffic which is carried over the default VRF.

With a BlueField running HBN able to act as a border gateway, there is no longer a dependence on physical switches and routers to terminate VXLAN traffic and perform this role, hence the requirements on the underlying network is simply to provide end-to-

end IP/UDP connectivity and facilitate the setup of overlay networks on top. Additionally, multiple border gateways can be easily deployed in the network, including dedicated gateways per tenant or shared gateways for groups of tenants.

(i) Note

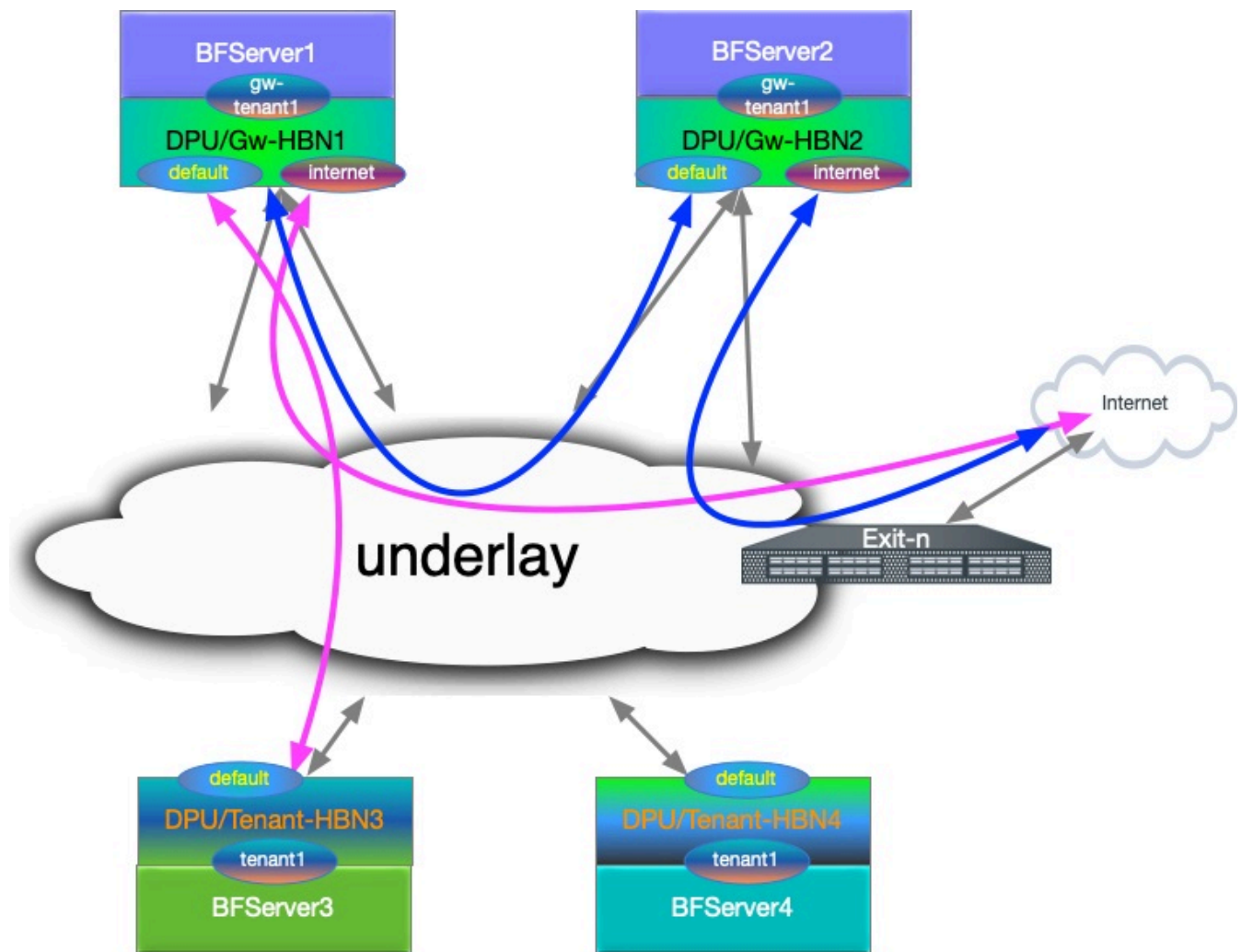
Since HBN currently does not support network address translation (NAT), a dedicated border gateway must be deployed per tenant, for those tenants that have overlapping IP addresses.

For more details and configuration of some of the key features that together enable the border gateway functionality, refer to sections on [Downstream VNIs](#) and [VLAN Subinterfaces](#).

Gateway Application Example

The following topology diagram and associated configuration snippets show two different use cases of border gateway deployment:

- tenant1 is an example of a tenant hosted on a server(s) with a non-gateway BlueField, using a dedicated border gateway on BlueField Gw-HBN1 for Internet connectivity. Traffic flow to and from the Internet for this tenant is marked in pink.
- gw_tenant1 is an example of a tenant hosted on a server(s) with a gateway BlueField. In this case, the border gateway for this tenant is provided by BlueField Gw-HBN2. Traffic flow to and from the Internet for this tenant is depicted in blue .



L3 VNI Origin Map

HBN	VRF	L3 VNI
gw-hbn1 and gw-hbn2	internet1	10000
gw-hbn1 and gw-hbn2	gw_tenant1	30000
tenant-hbn3 and tenant-hbn4	tenant1	20000

Configuration Snippet for Internet VRF

- Internet VRF is established in BGP sessions using sub-interface features with underlay switches (i.e., p0_sf.60 and p1_sf.60)

- The Internet VRF also imports all the tenant VRFs (local and remote) using the downstream VNI feature with from-EVPN syntax

```
nv set interface p0_sf.60,p1_sf.60,vlan10 ip vrf internet1
nv set vrf internet1 evpn enable on
nv set vrf internet1 evpn vni 10000
nv set vrf internet1 loopback ip address 6.2.0.1/32
nv set vrf internet1 loopback ip address 2001:cafe:feed::1/128
nv set vrf internet1 router bgp address-family ipv4-unicast enable on
nv set vrf internet1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf internet1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf internet1 router bgp address-family ipv6-unicast enable on
nv set vrf internet1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf internet1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf internet1 router bgp address-family l2vpn-evpn enable on
nv set vrf internet1 router bgp autonomous-system 65552
nv set vrf internet1 router bgp enable on
nv set vrf internet1 router bgp neighbor p0_sf.60 capabilities source-address
internet1
nv set vrf internet1 router bgp neighbor p0_sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p0_sf.60 type unnumbered
nv set vrf internet1 router bgp neighbor p1_sf.60 capabilities source-address
internet1
nv set vrf internet1 router bgp neighbor p1_sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p1_sf.60 type unnumbered
nv set vrf internet1 router bgp peer-group l3_pg1 address-family ipv4-unicast
enable on
nv set vrf internet1 router bgp peer-group l3_pg1 address-family ipv6-unicast
enable on
nv set vrf internet1 router bgp peer-group l3_pg1 remote-as external
nv set vrf internet1 router bgp route-export to-evpn route-target 65552:10000
nv set vrf internet1 router bgp route-import from-evpn route-target ANY:20000
```

```
nv set vrf internet1 router bgp route-import from-evpn route-target ANY:30000
nv set vrf internet1 router bgp route-import from-evpn route-target auto
nv set vrf internet1 router bgp router-id 27.0.0.5
```

Configuration Snippet for Gateway Local Tenant

- gw_tenant is stretched across 2 gateway and connected using L3 VNI
- gw_tenant has multiple SVIs, which are represented as vlan30 and vlan31 SVIs
- Internet L3 VNI is imported using DVNI. The example also explicitly adds route targets using auto.

gw_tenant VRF:

```
nv set interface vlan30-31 ip vrf gw_tenant1
nv set vrf gw_tenant1 evpn enable on
nv set vrf gw_tenant1 evpn vni 30000
nv set vrf gw_tenant1 loopback ip address 15.3.0.1/32
nv set vrf gw_tenant1 loopback ip address 2001:bad:c0de::1/128
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf gw_tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf gw_tenant1 router bgp autonomous-system 65552
nv set vrf gw_tenant1 router bgp enable on
nv set vrf gw_tenant1 router bgp route-export to-evpn route-target 65552:30000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target ANY:10000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target auto
```

```
nv set vrf gw_tenant1 router bgp router-id 27.0.0.5
```

Configuration Snippet for Remote Tenant

- tenant1 is stretched across 2 remote HBN VTEP and connected using L3 VNI
- tenant1 is importing Internet L3 VNI routes in tenant1 and adding its own using route-target auto

Tenant VRF:

```
nv set interface vlan20-21 ip vrf tenant1
nv set vrf tenant1 evpn enable on
nv set vrf tenant1 evpn vni 20000
nv set vrf tenant1 loopback ip address 15.1.0.1/32
nv set vrf tenant1 loopback ip address 2001:c001:c0de::1/128
nv set vrf tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf tenant1 router bgp autonomous-system 6300656
nv set vrf tenant1 router bgp enable on
nv set vrf tenant1 router bgp route-export to-evpn route-target 6300656:20000
nv set vrf tenant1 router bgp route-import from-evpn route-target ANY:10000
nv set vrf tenant1 router bgp route-import from-evpn route-target auto
nv set vrf tenant1 router bgp router-id 27.0.0.17
```


HBN Accelerated Routing Plan

The following subsections pick a few IP endpoints from the code snippets above and examine their route distribution.

- The gateway devices have a remote tenant
- Internet route is injected using the default originator from the exit node

Gateway-1 Route Info

- BGP sharing the uplink via a sub-interface feature in the Internet VRF.

```
root@hbn:/# ip -4 route show vrf internet1 default
default proto bgp metric 20
nexthop via 169.254.0.1 dev p0_sf.60 weight 1 onlink
nexthop via 169.254.0.1 dev p1_sf.60 weight 1 onlink

root@hbn:/# ip -6 route show vrf internet1 default
default proto bgp metric 20 pref medium
nexthop via fe80::202:ff:fe00:1b dev p0_sf.60 weight 1
nexthop via fe80::202:ff:fe00:23 dev p1_sf.60 weight 1
```

- Local Tenant routing information: The Internet is reached using L3 VNI via a peer gateway.

```
root@hbn:/# ip -4 route show vrf gw_tenant1 default
default encap ip id 10000 src 0.0.0.0 dst 27.0.0.7 ttl 0 tos 0 via 27.0.0.7 dev vxlan48
proto bgp metric 20 onlink

root@hbn:/# ip -6 route show vrf gw_tenant1 default
default encap ip id 10000 src 0.0.0.0 dst 27.0.0.7 ttl 0 tos 0 via ::ffff:27.0.0.7 dev
vxlan48 proto bgp metric 20 onlink pref medium
```

- Remote tenant routing reachability via gateway1 using DVNI CFG.
- Considering an IP endpoint from the remote tenant1 VRF on Tenant-HBN3.

```
root@hbn:/# ip -4 route show vrf internet1 15.1.0.1/32
15.1.0.1 encap ip id 20000 src 0.0.0.0 dst 27.0.0.17 ttl 0 tos 0 via 27.0.0.17 dev vxlan48
proto bgp metric 20 onlink
```

```
root@hbn:/# ip -6 route show vrf internet1 2001:c001:c0de::1/128
2001:c001:c0de::1 encap ip id 20000 src 0.0.0.0 dst 27.0.0.17 ttl 0 tos 0 via
::ffff:27.0.0.17 dev vxlan48 proto bgp metric 20 onlink pref medium
```

Tenant-HBN3 Route Info

- IP endpoint as gateway1 VRF loopback and DVNI handoff for the VNI is reaching the gateway1 node.

```
root@hbn:/# ip -4 route show vrf tenant1 6.2.0.1/32
6.2.0.1 encap ip id 10000 src 0.0.0.0 dst 27.0.0.5 ttl 0 tos 0 via 27.0.0.5 dev vxlan48
proto bgp metric 20 onlink
```

```
root@hbn:/# ip -6 route show vrf tenant1 2001:cafe:feed::1/128
2001:cafe:feed::1 encap ip id 10000 src 0.0.0.0 dst 27.0.0.5 ttl 0 tos 0 via ::ffff:27.0.0.5
dev vxlan48 proto bgp metric 20 onlink pref medium
```

- Internet VRF default route is reaching the remote tenant VRF.

```
root@hbn:/# ip -4 route show vrf tenant1 default
default proto bgp metric 20
nexthop encap ip id 10000 src 0.0.0.0 dst 27.0.0.5 ttl 0 tos 0 via 27.0.0.5 dev vxlan48
weight 1 onlink
nexthop encap ip id 10000 src 0.0.0.0 dst 27.0.0.7 ttl 0 tos 0 via 27.0.0.7 dev vxlan48
weight 1 onlink
```

```
root@hbn:/# ip -6 route show vrf tenant1 default
default proto bgp metric 20 pref medium
```

```
nexthop encap ip id 10000 src 0.0.0.0 dst 27.0.0.5 ttl 0 tos 0 via ::ffff:27.0.0.5 dev
vxlan48 weight 1 onlink
nexthop encap ip id 10000 src 0.0.0.0 dst 27.0.0.7 ttl 0 tos 0 via ::ffff:27.0.0.7 dev
vxlan48 weight 1 onlink
```

Gateway and Tenant Complete Configuration Example

Gateway-1 Full Configuration

```
nv set bridge domain br_default encap 802.1Q
nv set bridge domain br_default type vlan-aware
nv set bridge domain br_default untagged 1
nv set bridge domain br_default vlan 10,30-31
nv set evpn enable on
nv set interface lo ip address 27.0.0.5/32
nv set interface lo ip address 2001:c001:ff:f00d::5/128
nv set interface lo type loopback
nv set interface
p0_sf,p1_sf,pf0hpf_sf,pf0vf0_sf,pf0vf10_sf,pf0vf11_sf,pf0vf12_sf,pf0vf1_sf,pf0vf2_sf,pf0
type swp
nv set interface p0_sf.60 base-interface p0_sf
nv set interface p0_sf.60,p1_sf.60 type sub
nv set interface p0_sf.60,p1_sf.60 vlan 60
nv set interface p0_sf.60,p1_sf.60,vlan10 ip vrf internet1
nv set interface p1_sf.60 base-interface p1_sf
nv set interface pf0hpf_sf bridge domain br_default access 30
nv set interface pf0vf0_sf bridge domain br_default access 31
nv set interface vlan10 ip address 12.2.0.1/24
nv set interface vlan10 ip address 2001:c001:d00d::1/96
nv set interface vlan10 vlan 10
nv set interface vlan10,30-31 ip ipv4 forward on
nv set interface vlan10,30-31 ip ipv6 forward on
nv set interface vlan10,30-31 type svi
```

```
nv set interface vlan30 ip address 45.3.0.1/24
nv set interface vlan30 ip address 2001:b055:b00c::1/96
nv set interface vlan30 vlan 30
nv set interface vlan30-31 ip vrf gw_tenant1
nv set interface vlan31 ip address 45.3.1.1/24
nv set interface vlan31 ip address 2001:b055:b00c::1:0:1/96
nv set interface vlan31 vlan 31
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.5
nv set platform
nv set router bgp enable on
nv set system config snippet
nv set system global anycast-mac 44:38:39:42:42:17
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 65552
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 27.0.0.11 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.11 type numbered
nv set vrf default router bgp neighbor 27.0.0.12 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.12 type numbered
nv set vrf default router bgp neighbor p0_sf capabilities source-address lo
nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf capabilities source-address lo
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
```

```
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-tx
off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable off
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group fabric timers connection-retry 5
nv set vrf default router bgp peer-group fabric timers hold 30
nv set vrf default router bgp peer-group fabric timers keepalive 10
nv set vrf default router bgp peer-group rs_client address-family ipv4-unicast enable
off
nv set vrf default router bgp peer-group rs_client address-family ipv6-unicast enable
off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn add-
path-tx off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn enable
on
nv set vrf default router bgp peer-group rs_client multihop-ttl 5
nv set vrf default router bgp peer-group rs_client remote-as external
nv set vrf default router bgp peer-group rs_client timers connection-retry 5
nv set vrf default router bgp peer-group rs_client timers hold 30
nv set vrf default router bgp peer-group rs_client timers keepalive 10
nv set vrf default router bgp router-id 27.0.0.5
nv set vrf gw_tenant1 evpn enable on
nv set vrf gw_tenant1 evpn vni 30000
nv set vrf gw_tenant1 loopback ip address 15.3.0.1/32
nv set vrf gw_tenant1 loopback ip address 2001:bad:c0de::1/128
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
```

```
nv set vrf gw_tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf gw_tenant1 router bgp autonomous-system 65552
nv set vrf gw_tenant1 router bgp enable on
nv set vrf gw_tenant1 router bgp route-export to-evpn route-target 65552:30000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target ANY:10000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target auto
nv set vrf gw_tenant1 router bgp router-id 27.0.0.5
nv set vrf internet1 evpn enable on
nv set vrf internet1 evpn vni 10000
nv set vrf internet1 loopback ip address 6.2.0.1/32
nv set vrf internet1 loopback ip address 2001:cafe:feed::1/128
nv set vrf internet1 router bgp address-family ipv4-unicast enable on
nv set vrf internet1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf internet1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf internet1 router bgp address-family ipv6-unicast enable on
nv set vrf internet1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf internet1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf internet1 router bgp address-family l2vpn-evpn enable on
nv set vrf internet1 router bgp autonomous-system 65552
nv set vrf internet1 router bgp enable on
nv set vrf internet1 router bgp neighbor p0_sf.60 capabilities source-address
internet1
nv set vrf internet1 router bgp neighbor p0_sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p0_sf.60 type unnumbered
nv set vrf internet1 router bgp neighbor p1_sf.60 capabilities source-address
internet1
nv set vrf internet1 router bgp neighbor p1_sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p1_sf.60 type unnumbered
nv set vrf internet1 router bgp peer-group l3_pg1 address-family ipv4-unicast
enable on
nv set vrf internet1 router bgp peer-group l3_pg1 address-family ipv6-unicast
enable on
```

```
nv set vrf internet1 router bgp peer-group l3_pg1 remote-as external
nv set vrf internet1 router bgp route-export to-evpn route-target 65552:10000
nv set vrf internet1 router bgp route-import from-evpn route-target ANY:20000
nv set vrf internet1 router bgp route-import from-evpn route-target ANY:30000
nv set vrf internet1 router bgp route-import from-evpn route-target auto
nv set vrf internet1 router bgp router-id 27.0.0.5
```

Gateway-2 Full Configuration

```
nv set bridge domain br_default encap 802.1Q
nv set bridge domain br_default type vlan-aware
nv set bridge domain br_default untagged 1
nv set bridge domain br_default vlan 10,30-31
nv set evpn enable on
nv set interface lo ip address 27.0.0.7/32
nv set interface lo ip address 2001:c001:ff:f00d::7/128
nv set interface lo type loopback
nv set interface
p0_sf,p1_sf,pf0hpf_sf,pf0vf0_sf,pf0vf10_sf,pf0vf11_sf,pf0vf12_sf,pf0vf1_sf,pf0vf2_sf,pf0
type swp
nv set interface p0_sf.60 base-interface p0_sf
nv set interface p0_sf.60,p1_sf.60 type sub
nv set interface p0_sf.60,p1_sf.60 vlan 60
nv set interface p0_sf.60,p1_sf.60,vlan10 ip vrf internet1
nv set interface p1_sf.60 base-interface p1_sf
nv set interface pf0hpf_sf bridge domain br_default access 30
nv set interface pf0vf0_sf bridge domain br_default access 31
nv set interface vlan10 ip address 12.2.1.1/24
nv set interface vlan10 ip address 2001:c001:d00d::1:0:1/96
nv set interface vlan10 vlan 10
nv set interface vlan10,30-31 ip ipv4 forward on
nv set interface vlan10,30-31 ip ipv6 forward on
nv set interface vlan10,30-31 type svi
nv set interface vlan30 ip address 45.3.2.1/24
```

```
nv set interface vlan30 ip address 2001:b055:b00c::2:0:1/96
nv set interface vlan30 vlan 30
nv set interface vlan30-31 ip vrf gw_tenant1
nv set interface vlan31 ip address 45.3.3.1/24
nv set interface vlan31 ip address 2001:b055:b00c::3:0:1/96
nv set interface vlan31 vlan 31
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.7
nv set platform
nv set router bgp enable on
nv set system config snippet
nv set system global anycast-mac 44:38:39:42:42:19
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 65554
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 27.0.0.11 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.11 type numbered
nv set vrf default router bgp neighbor 27.0.0.12 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.12 type numbered
nv set vrf default router bgp neighbor p0_sf capabilities source-address lo
nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf capabilities source-address lo
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
```



```
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-tx
off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable off
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group fabric timers connection-retry 5
nv set vrf default router bgp peer-group fabric timers hold 30
nv set vrf default router bgp peer-group fabric timers keepalive 10
nv set vrf default router bgp peer-group rs_client address-family ipv4-unicast enable
off
nv set vrf default router bgp peer-group rs_client address-family ipv6-unicast enable
off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn add-
path-tx off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn enable
on
nv set vrf default router bgp peer-group rs_client multihop-ttl 5
nv set vrf default router bgp peer-group rs_client remote-as external
nv set vrf default router bgp peer-group rs_client timers connection-retry 5
nv set vrf default router bgp peer-group rs_client timers hold 30
nv set vrf default router bgp peer-group rs_client timers keepalive 10
nv set vrf default router bgp router-id 27.0.0.7
nv set vrf gw_tenant1 evpn enable on
nv set vrf gw_tenant1 evpn vni 30000
nv set vrf gw_tenant1 loopback ip address 15.3.0.2/32
nv set vrf gw_tenant1 loopback ip address 2001:bad:c0de::2/128
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf gw_tenant1 router bgp address-family l2vpn-evpn enable on
```

```
nv set vrf gw_tenant1 router bgp autonomous-system 65554
nv set vrf gw_tenant1 router bgp enable on
nv set vrf gw_tenant1 router bgp route-export to-evpn route-target 65554:30000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target ANY:10000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target auto
nv set vrf gw_tenant1 router bgp router-id 27.0.0.7
nv set vrf internet1 evpn enable on
nv set vrf internet1 evpn vni 10000
nv set vrf internet1 loopback ip address 6.2.0.2/32
nv set vrf internet1 loopback ip address 2001:cafe:feed::2/128
nv set vrf internet1 router bgp address-family ipv4-unicast enable on
nv set vrf internet1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf internet1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf internet1 router bgp address-family ipv6-unicast enable on
nv set vrf internet1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf internet1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf internet1 router bgp address-family l2vpn-evpn enable on
nv set vrf internet1 router bgp autonomous-system 65554
nv set vrf internet1 router bgp enable on
nv set vrf internet1 router bgp neighbor p0_sf.60 capabilities source-address
internet1
nv set vrf internet1 router bgp neighbor p0_sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p0_sf.60 type unnumbered
nv set vrf internet1 router bgp neighbor p1_sf.60 capabilities source-address
internet1
nv set vrf internet1 router bgp neighbor p1_sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p1_sf.60 type unnumbered
nv set vrf internet1 router bgp peer-group l3_pg1 address-family ipv4-unicast
enable on
nv set vrf internet1 router bgp peer-group l3_pg1 address-family ipv6-unicast
enable on
nv set vrf internet1 router bgp peer-group l3_pg1 remote-as external
```

```
nv set vrf internet1 router bgp route-export to-evpn route-target 65554:10000
nv set vrf internet1 router bgp route-import from-evpn route-target ANY:20000
nv set vrf internet1 router bgp route-import from-evpn route-target ANY:30000
nv set vrf internet1 router bgp route-import from-evpn route-target auto
nv set vrf internet1 router bgp router-id 27.0.0.7
```

Tenant-HBN-3 Full Configuration

```
nv set bridge domain br_default encap 802.1Q
nv set bridge domain br_default type vlan-aware
nv set bridge domain br_default untagged 1
nv set bridge domain br_default vlan 20-21
nv set evpn enable on
nv set interface lo ip address 27.0.0.17/32
nv set interface lo ip address 2001:c001:ff:f00d::11/128
nv set interface lo type loopback
nv set interface p0-1,pf0hpf,pf0vf0-12,pf1hpf,pf1vf0-4 type swp
nv set interface pf0hpf bridge domain br_default access 20
nv set interface pf0vf0 bridge domain br_default access 21
nv set interface vlan20 ip address 45.1.0.1/24
nv set interface vlan20 ip address 2001:c001:b00c::1/96
nv set interface vlan20 vlan 20
nv set interface vlan20-21 ip ipv4 forward on
nv set interface vlan20-21 ip ipv6 forward on
nv set interface vlan20-21 ip vrf tenant1
nv set interface vlan20-21 type svi
nv set interface vlan21 ip address 45.1.1.1/24
nv set interface vlan21 ip address 2001:c001:b00c::1:0:1/96
nv set interface vlan21 vlan 21
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.17
nv set platform
```

```
nv set router bgp enable on
nv set system global anycast-mac 44:38:39:42:42:21
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 6300656
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 27.0.0.11 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.11 type numbered
nv set vrf default router bgp neighbor 27.0.0.12 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.12 type numbered
nv set vrf default router bgp neighbor p0 capabilities source-address lo
nv set vrf default router bgp neighbor p0 peer-group fabric
nv set vrf default router bgp neighbor p0 type unnumbered
nv set vrf default router bgp neighbor p1 capabilities source-address lo
nv set vrf default router bgp neighbor p1 peer-group fabric
nv set vrf default router bgp neighbor p1 type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-tx
off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable off
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group fabric timers connection-retry 5
nv set vrf default router bgp peer-group fabric timers hold 30
nv set vrf default router bgp peer-group fabric timers keepalive 10
nv set vrf default router bgp peer-group rs_client address-family ipv4-unicast enable
off
nv set vrf default router bgp peer-group rs_client address-family ipv6-unicast enable
off
```

```
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn add-
path-tx off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn enable
on
nv set vrf default router bgp peer-group rs_client multihop-ttl 5
nv set vrf default router bgp peer-group rs_client remote-as external
nv set vrf default router bgp peer-group rs_client timers connection-retry 5
nv set vrf default router bgp peer-group rs_client timers hold 30
nv set vrf default router bgp peer-group rs_client timers keepalive 10
nv set vrf default router bgp router-id 27.0.0.17
nv set vrf tenant1 evpn enable on
nv set vrf tenant1 evpn vni 20000
nv set vrf tenant1 loopback ip address 15.1.0.1/32
nv set vrf tenant1 loopback ip address 2001:c001:c0de::1/128
nv set vrf tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf tenant1 router bgp autonomous-system 6300656
nv set vrf tenant1 router bgp enable on
nv set vrf tenant1 router bgp route-export to-evpn route-target 6300656:20000
nv set vrf tenant1 router bgp route-import from-evpn route-target ANY:10000
nv set vrf tenant1 router bgp route-import from-evpn route-target auto
nv set vrf tenant1 router bgp router-id 27.0.0.17
```

Tenant-HBN-4 Full Configuration

```
nv set bridge domain br_default encap 802.1Q
nv set bridge domain br_default type vlan-aware
nv set bridge domain br_default untagged 1
nv set bridge domain br_default vlan 20-21
nv set evpn enable on
nv set interface lo ip address 27.0.0.19/32
nv set interface lo ip address 2001:c001:ff:f00d::13/128
nv set interface lo type loopback
nv set interface p0-1,pf0hpf,pf0vf0-12,pf1hpf,pf1vf0-4 type swp
nv set interface pf0hpf bridge domain br_default access 20
nv set interface pf0vf0 bridge domain br_default access 21
nv set interface vlan20 ip address 45.1.2.1/24
nv set interface vlan20 ip address 2001:c001:b00c::2:0:1/96
nv set interface vlan20 vlan 20
nv set interface vlan20-21 ip ipv4 forward on
nv set interface vlan20-21 ip ipv6 forward on
nv set interface vlan20-21 ip vrf tenant1
nv set interface vlan20-21 type svi
nv set interface vlan21 ip address 45.1.3.1/24
nv set interface vlan21 ip address 2001:c001:b00c::3:0:1/96
nv set interface vlan21 vlan 21
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.19
nv set platform
nv set router bgp enable on
nv set system global anycast-mac 44:38:39:42:42:23
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
```

```
nv set vrf default router bgp autonomous-system 6300658
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 27.0.0.11 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.11 type numbered
nv set vrf default router bgp neighbor 27.0.0.12 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.12 type numbered
nv set vrf default router bgp neighbor p0 capabilities source-address lo
nv set vrf default router bgp neighbor p0 peer-group fabric
nv set vrf default router bgp neighbor p0 type unnumbered
nv set vrf default router bgp neighbor p1 capabilities source-address lo
nv set vrf default router bgp neighbor p1 peer-group fabric
nv set vrf default router bgp neighbor p1 type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-tx
off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable off
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group fabric timers connection-retry 5
nv set vrf default router bgp peer-group fabric timers hold 30
nv set vrf default router bgp peer-group fabric timers keepalive 10
nv set vrf default router bgp peer-group rs_client address-family ipv4-unicast enable
off
nv set vrf default router bgp peer-group rs_client address-family ipv6-unicast enable
off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn add-
path-tx off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn enable
on
nv set vrf default router bgp peer-group rs_client multihop-ttl 5
nv set vrf default router bgp peer-group rs_client remote-as external
nv set vrf default router bgp peer-group rs_client timers connection-retry 5
nv set vrf default router bgp peer-group rs_client timers hold 30
nv set vrf default router bgp peer-group rs_client timers keepalive 10
nv set vrf default router bgp router-id 27.0.0.19
```

```
nv set vrf tenant1 evpn enable on
nv set vrf tenant1 evpn vni 20000
nv set vrf tenant1 loopback ip address 15.1.0.2/32
nv set vrf tenant1 loopback ip address 2001:c001:c0de::2/128
nv set vrf tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast redistribute connected
enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast route-export to-evpn
enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast redistribute connected
enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast route-export to-evpn
enable on
nv set vrf tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf tenant1 router bgp autonomous-system 6300658
nv set vrf tenant1 router bgp enable on
nv set vrf tenant1 router bgp route-export to-evpn route-target 6300658:20000
nv set vrf tenant1 router bgp route-import from-evpn route-target ANY:10000
nv set vrf tenant1 router bgp route-import from-evpn route-target auto
nv set vrf tenant1 router bgp router-id 27.0.0.19
```

Access Control Lists

Access Control Lists (ACLs) are a set of rules that are used to filter network traffic. These rules are used to specify the traffic flows that must be permitted or blocked at networking device interfaces. There are two types of ACLs:

- Stateless ACLs – rules that are applied to individual packets. They inspect each packet individually and permit/block the packets based on the packet header information and the match criteria specified by the rule.
- Stateful ACLs – rules that are applied to traffic sessions/connections. They inspect each packet with respect to the state of the session/connection to which the packet

belongs to determine whether to permit/block the packet.

Stateless ACLs

HBN supports configuration of stateless ACLs for IPv4 packets, IPv6 packets, and Ethernet (MAC) frames. The following examples depict how stateless ACLs are configured for each case, with NVUE and with flat files (cl-acltool).

NVUE Examples for Stateless ACLs

NVUE IPv4 ACLs Example

The following is an example of an ingress IPv4 ACL that permits DHCP request packets ingressing on the pf0hpf_sf port towards the DHCP server:

```
root@hbn01-host01:~# nv set acl acl1_ingress type ipv4
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 match ip protocol udp
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 match ip dest-port 67
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 match ip source-port 68
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 action permit
```

Bind the ingress IPv4 ACL to host representor port pf0hpf_sf of BlueField in the inbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl1_ingress inbound
root@hbn01-host01:~# nv config apply
```

The following is an example of an egress IPv4 ACL that permits DHCP reply packets egressing out of the pf0hpf_sf port towards the DHCP client:

```
root@hbn01-host01:~# nv set acl acl2_egress type ipv4
root@hbn01-host01:~# nv set acl acl2_egress rule 200 match ip protocol udp
root@hbn01-host01:~# nv set acl acl2_egress rule 200 match ip dest-port 68
root@hbn01-host01:~# nv set acl acl2_egress rule 200 match ip source-port 67
root@hbn01-host01:~# nv set acl acl2_egress rule 200 action permit
```

Bind the egress IPv4 ACL to host representor port pf0hpf_sf of BlueField in the outbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl2_egress outbound
root@hbn01-host01:~# nv config apply
```

NVUE IPv6 ACLs Example

The following is an example of an ingress IPv6 ACL that permits traffic with matching dest-ip and protocol tcp ingress on port pf0hpf_sf:

```
root@hbn01-host01:~# nv set acl acl5_ingress type ipv6
root@hbn01-host01:~# nv set acl acl5_ingress rule 100 match ip protocol tcp
root@hbn01-host01:~# nv set acl acl5_ingress rule 100 match ip dest-ip
48:2034::80:9
root@hbn01-host01:~# nv set acl acl5_ingress rule 100 action permit
```

Bind the ingress IPv6 ACL to host representor port pf0hpf_sf of BlueField in the inbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl5_ingress inbound
root@hbn01-host01:~# nv config apply
```

The following is an example of an egress IPv6 ACL that permits traffic with matching source-ip and protocol tcp egressing out of port pf0hpf_sf:

```
root@hbn01-host01:~# nv set acl acl6_egress type ipv6
root@hbn01-host01:~# nv set acl acl6_egress rule 101 match ip protocol tcp
root@hbn01-host01:~# nv set acl acl6_egress rule 101 match ip source-ip
48:2034::80:9
root@hbn01-host01:~# nv set acl acl6_egress rule 101 action permit
```

Bind the egress IPv6 ACL to host representor port pf0hpf_sf of BlueField in the outbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl6_egress outbound
root@hbn01-host01:~# nv config apply
```

NVUE MAC ACLs Example

The following is an example of an ingress MAC ACL that permits traffic with matching source-mac and dest-mac ingressing to port pf0hpf_sf:

```
root@hbn01-host01:~# nv set acl acl3_ingress type mac
root@hbn01-host01:~# nv set acl acl3_ingress rule 1 match mac source-mac
00:00:00:00:00:0a
root@hbn01-host01:~# nv set acl acl3_ingress rule 1 match mac dest-mac
00:00:00:00:00:0b
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl3_ingress inbound
```

Bind the ingress MAC ACL to host representor port pf0hpf_sf of BlueField in the inbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl3_ingress inbound
root@hbn01-host01:~# nv config apply
```

The following is an example of an egress MAC ACL that permits traffic with matching source-mac and dest-mac egressing out of port pf0hpf_sf:

```
root@hbn01-host01:~# nv set acl acl4_egress type mac
root@hbn01-host01:~# nv set acl acl4_egress rule 2 match mac source-mac
00:00:00:00:00:0b
root@hbn01-host01:~# nv set acl acl4_egress rule 2 match mac dest-mac
00:00:00:00:00:0a
root@hbn01-host01:~# nv set acl acl4_egress rule 2 action permit
```

Bind the egress MAC ACL to host representor port pf0hpf_sf of BlueField in the outbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl4_egress outbound
```

```
root@hbn01-host01:~# nv config apply
```

Flat Files (cl-acltool) Examples for Stateless ACLs

For the same examples cited above, the following are the corresponding ACL rules which must be configured under `/etc/cumulus/acl/policy.d/<rule_name.rules>` followed by invoking `cl-acltool -i`. The rules in `/etc/cumulus/acl/policy.d/<rule_name.rules>` are configured using Linux `iptables/ip6tables/eatables`.

Flat Files IPv4 ACLs Example

The following example configures an ingress IPv4 ACL rule matching with DHCP request under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the ingress interface as the host representor of BlueField followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL acl1_ingress in dir inbound on interface pf1vf1_sf ##
-t filter -A FORWARD -m physdev --physdev-in pf1vf1_sf -p udp --sport 68 --dport 67 -
j ACCEPT
```

The following example configures an egress IPv4 ACL rule matching with DHCP reply under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the egress interface as the host representor of BlueField followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL acl2_egress in dir outbound on interface pf1vf1_sf ##
-t filter -A FORWARD -m physdev --physdev-out pf1vf1_sf -p udp --sport 67 --dport 68
-j ACCEPT
```

Flat File IPv6 ACLs Example

The following example configures an ingress IPv6 ACL rule matching with `dest-ip` and `tcp` protocol under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the ingress interface as the host representor of BlueField followed by invoking `cl-acltool -i`:

```
[ip6tables]
```

```
## ACL acl5_ingress in dir inbound on interface pf0hpf_sf ##  
-t filter -A FORWARD -m physdev --physdev-in pf0hpf_sf -d 48:2034::80:9 -p tcp -j  
ACCEPT
```

The following example configures an egress IPv6 ACL rule matching with `source-ip` and `tcp` protocol under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the egress interface as the host representor of BlueField followed by invoking `cl-acltool -i`:

```
[ip6tables]  
## ACL acl6_egress in dir outbound on interface pf0hpf_sf ##  
-t filter -A FORWARD -m physdev --physdev-out pf0hpf_sf -s 48:2034::80:9 -p tcp -j  
ACCEPT
```

Flat Files MAC ACLs Example

The following example configures an ingress MAC ACL rule matching with `source-mac` and `dest-mac` under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the ingress interface as the host representor of BlueField followed by invoking `cl-acltool -i`:

```
[ebtables]  
## ACL acl3_ingress in dir inbound on interface pf0hpf_sf ##  
-t filter -A FORWARD -m physdev --physdev-in pf0hpf_sf -s  
00:00:00:00:00:0a/ff:ff:ff:ff:ff:ff -d 00:00:00:00:00:0b/ff:ff:ff:ff:ff:ff -j ACCEPT
```

The following example configures an egress MAC ACL rule matching with `source-mac` and `dest-mac` under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with egress interface as host representor of BlueField followed by invoking `cl-acltool -i`:

```
[ebtables]  
## ACL acl4_egress in dir outbound on interface pf0hpf_sf ##  
-t filter -A FORWARD -m physdev --physdev-out pf0hpf_sf -s  
00:00:00:00:00:0b/ff:ff:ff:ff:ff:ff -d 00:00:00:00:00:0a/ff:ff:ff:ff:ff:ff -j ACCEPT
```

Stateful ACLs

Stateful ACLs facilitate monitoring and tracking traffic flows to enforce per-flow traffic filtering (unlike stateless ACLs which filter traffic on a per-packet basis). HBN supports stateful ACLs using reflexive ACL mechanism. Reflexive ACL mechanism is used to allow initiation of connections from "within" the network to "outside" the network and allow only replies to the initiated connections from "outside" the network (or vice versa).

HBN supports stateful ACL configuration for IPv4 traffic.

Stateful ACLs can be applied for native routed traffic (north-south underlay routed traffic in EVPN deployments), EVPN bridged traffic (east-west overlay bridged/L2 traffic in EVPN deployments) and EVPN routed traffic (east-west overlay routed traffic in EVPN deployments). Stateful ACLs applied for native routed traffic are called "Native-L3 stateful ACLs". Stateful ACLs applied for EVPN bridged traffic and EVPN routed traffic are called "EVPN-L2 stateful ACLs" and "EVPN-L3 stateful ACLs", respectively.

Stateful ACLs in HBN are disabled by default. To enable stateful ACL functionality, use the following NVUE commands:

```
root@hbn03-host00:~# nv set system reflexive-acl enable
root@hbn03-host00:~# nv config apply
```

If using flat-file configuration (and not NVUE), edit the file `/etc/cumulus/nl2docad.d/acl.conf` and set the knob `rlx.reflexive_acl_enable` to `TRUE`. To apply this change, execute:

```
root@hbn03-host00:~# supervisorctl start nl2doca-reload
```

NVUE Example for Native-L3 Stateful ACLs

The following is an example of allowing HTTP (TCP) connection originated by the host, where BlueField is hosted, to an HTTP server (with the IP address 11.11.11.11) on an external network. Two sets of ACLs matching with CONNTRACK state must be configured for a CONNTRACK entry to be established in the kernel which would be offloaded to hardware:

- Configure an ACL rule matching TCP/HTTP connection/flow details with CONNTRACK state of NEW, ESTABLISHED and bind it to the SVI in the inbound direction.
- Configure an ACL rule matching TCP/HTTP connection/flow details with CONNTRACK state of ESTABLISHED and bind it to the SVI in the outbound direction.

Native-L3 stateful ACLs should be bound to an SVI interface. In this example, SVI interface is vlan101.

1. Configure the ingress ACL rule:

```
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 action permit
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match conntrack new
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match conntrack established
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip dest-ip 11.11.11.11/32
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip dest-port 80
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip protocol tcp
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host type ipv4
```

2. Bind this ACL to the SVI interface in the inbound direction:

```
root@hbn03-host00:~# nv set interface vlan101 acl allow_tcp_conn_from_host inbound
root@hbn03-host00:~# nv config apply
```

3. Configure the egress ACL rule:

```
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 action permit
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match conntrack established
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match ip protocol tcp
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server type ipv4
root@hbn03-host00:~# nv config apply
```

4. Bind this ACL to the SVI interface in the outbound direction:

```
root@hbn03-host00:~# nv set interface vlan101 acl
allow_tcp_resp_from_server outbound
root@hbn03-host00:~# nv config apply
```

Note

If virtual router redundancy (VRR) is set, L3 stateful ACLs must be bound to all the related SVI interfaces. For example, if VRR is configured on SVI `vlan101` as follows in the `/etc/network/interfaces` file:

```
auto vlan101
iface vlan101
    address 45.3.1.2/24
    address-virtual 00:00:5e:00:01:01 45.3.1.1/24
    vlan-raw-device br_default
vlan-id 101
```

With this configuration, two SVI interfaces, `vlan101` and `vlan101-v0` would be created in the system:

```
root@hbn03-host00:~# ip -br addr show | grep vlan101
vlan101@br_default UP          45.3.1.2/24
fe80::204:4bff:fe8a:f100/64
vlan101-v0@vlan101 UP        45.3.1.1/24 metric 1024
fe80::200:5eff:fe00:101/64
```

In this case, stateful ACLs must be bound to both SVI interfaces (`vlan101` and `vlan101-v0`). In the stateful ACL described in the current section, the binding would be:


```
root@hbn03-host00:~# nv set interface vlan101,vlan101-
v0 acl allow_tcp_conn_from_host inbound
root@hbn03-host00:~# nv set interface vlan101,vlan101-
v0 acl allow_tcp_resp_from_server outbound
root@hbn03-host00:~# nv config apply
```

Flat Files (cl-acltool) Example for Native-L3 Stateful ACLs

For the same NVUE example for Native-L3 stateful ACLs cited above (HTTP server at IP address 11.11.11.11 on an external network), the following are the corresponding ACL rules which must be configured under `/etc/cumulus/acl/policy.d/<rule_name.rules>` followed by invoking `cl-acltool -i` to install the rules in BlueField hardware.

1. Configure an ingress ACL rule matching with TCP flow details and CONNTRACK state of NEW, ESTABLISHED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` with the ingress interface as the SVI followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL allow_tcp_conn_from_host in dir inbound on interface vlan101 ##
-t filter -A FORWARD -i vlan101 -p tcp -d 11.11.11.11/32 --dport 80 -m
conntrack --ctstate EST,NEW -m connmark ! --mark 7998 -j CONNMARK --set-
mark 7999
-t filter -A FORWARD -i vlan101 -p tcp -d 11.11.11.11/32 --dport 80 -m
conntrack --ctstate EST,NEW -j ACCEPT
```

Note

As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for ingress ACL rules are protocol dependent: 7999 for TCP, 7997 for UDP, and 7995 for ICMP.

2. Configure an egress ACL rule matching the TCP flow and CONNTRACK state of ESTABLISHED, RELATED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` file with the egress interface as SVI followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL allow_tcp_resp_from_server in dir outbound on interface vlan101 ##
-t filter -A FORWARD -o vlan101 -p tcp -s 11.11.11.11/32 --sport 80 -m
conntrack --ctstate EST -j CONNMARK --set-mark 7998
-t filter -A FORWARD -o vlan101 -p tcp -s 11.11.11.11/32 --sport 80 -m
conntrack --ctstate EST -j ACCEPT
```

Note

As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for egress ACL rules are protocol dependent: 7998 for TCP, 7996 for UDP, and 7994 for ICMP.

NVUE Example for EVPN-L2 Stateful ACLs

The following is an example allowing HTTP (TCP) connection originated by the host, hosting BlueField, to an HTTP server (with the IP address 192.168.5.5) accessible on the EVPN bridged network (L2 stretch). Two sets of ACLs matching with CONNTRACK state must be configured for a CONNTRACK entry to be established in the kernel which would be offloaded to hardware:

- Configure an ACL rule matching TCP/HTTP connection/flow details with a CONNTRACK state of NEW, ESTABLISHED, and bind it to the host interface in the inbound direction
- Configure an ACL rule matching TCP/HTTP connection/flow details with a CONNTRACK state of ESTABLISHED, and bind it to the host interface in the outbound direction

EVPN-L2 stateful ACLs should be bound to a host interface. In this example, the host interface is pf1vf7_sf.

1. Configure the ingress ACL rule:

```
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 action
permit
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match
conntrack new
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match
conntrack established
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip
dest-ip 192.168.5.5/32
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip
dest-port 80
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip
protocol tcp
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host type ipv4
```

2. Bind this ACL to the host interface in the inbound direction:

```
root@hbn03-host00:~# nv set interface pf1vf7_sf acl
allow_tcp_conn_from_host inbound
root@hbn03-host00:~# nv config apply
```

3. Configure the egress ACL rule:

```
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 action
permit
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match
conntrack established
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match ip
protocol tcp
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server type ipv4
root@hbn03-host00:~# nv config apply
```

4. Bind this ACL to the host interface in the outbound direction:

```
root@hbn03-host00:~# nv set interface pf1vf7_sf acl
allow_tcp_resp_from_server outbound
root@hbn03-host00:~# nv config apply
```

Flat Files (cl-acltool) Example for EVPN-L2 Stateful ACLs

For the same NVUE EVPN-L2 stateful ACLs example cited above (HTTP server at IP address 192.168.5.5 accessible over bridged network), the following are the corresponding ACL rules which must be configured under `/etc/cumulus/acl/policy.d/<rule_name.rules>` followed by invoking `cl-acltool -i`.

1. Configure an ingress ACL rule matching with TCP flow details and CONNTRACK state of NEW, ESTABLISHED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` with the ingress interface as the host representor of BlueField, followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL allow_tcp_conn_from_host in dir inbound on interface pf1vf7_sf ##
-t filter -A FORWARD -m physdev --physdev-in pf1vf7_sf -p tcp -d
192.168.5.5/32 --dport 80 -m conntrack --ctstate EST,NEW -m connmark ! --
mark 9998 -j CONNMARK --set-mark 9999
-t filter -A FORWARD -m physdev --physdev-in pf1vf7_sf -p tcp -d
192.168.5.5/32 --dport 80 -m conntrack --ctstate EST,NEW -j ACCEPT
```

Note

As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for ingress ACL rules are protocol dependent: 9999 for TCP, 9997 for UDP, and 9995 for ICMP.

2. Configure an egress ACL rule matching with TCP and CONNTRACK state of ESTABLISHED, RELATED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` with the egress interface as the host representor of BlueField, followed by invoking `cl-acltool -i:`

```
[iptables]
## ACL allow_tcp_resp_from_server in dir outbound on interface pf1vf7_sf ##
-t filter -A FORWARD -m physdev --physdev-out pf1vf7_sf -p tcp -s
192.168.5.5/32 --sport 80 -m conntrack --ctstate EST -j CONNMARK --set-mark
9998
-t filter -A FORWARD -m physdev --physdev-out pf1vf7_sf -p tcp -s
192.168.5.5/32 --sport 80 -m conntrack --ctstate EST -j ACCEPT
```

Note

As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for egress ACL rules are protocol dependent: 9998 for TCP, 9996 for UDP, and 9994 for ICMP.

NVUE Example for EVPN-L3 Stateful ACLs

The following is an example allowing an HTTP (TCP) connection originated by the host, hosting BlueField, to an HTTP server (with the IP address 21.1.1.2) accessible on the EVPN routed network (EVPN Symmetric Routing). Two sets of ACLs matching with CONNTRACK state must be configured for a CONNTRACK entry to be established in the kernel which would be offloaded to hardware:

- Configure an ACL rule matching TCP/HTTP connection/flow details with a CONNTRACK state of NEW, ESTABLISHED, and bind it to the host interface in the inbound direction
- Configure an ACL rule matching TCP/HTTP connection/flow details with a CONNTRACK state of ESTABLISHED, and bind it to the host interface in the outbound direction

EVPN-L3 stateful ACLs should be bound to an SVI interface. In this example, the SVI interface is vlan105.

1. Configure the ingress ACL rule:

```
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 action permit
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match conntrack new
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match conntrack established
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip dest-ip 21.1.1.2/32
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip dest-port 80
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip protocol tcp
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host type ipv4
```

2. Bind this ACL to the host interface in the inbound direction:

```
root@hbn03-host00:~# nv set interface vlan105 acl allow_tcp_conn_from_host inbound
root@hbn03-host00:~# nv config apply
```

3. Configure the egress ACL rule:

```
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 action permit
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match conntrack established
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match ip protocol tcp
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server type ipv4
root@hbn03-host00:~# nv config apply
```

4. Bind this ACL to the host interface in the outbound direction:

```
root@hbn03-host00:~# nv set interface vlan105 acl
allow_tcp_resp_from_server outbound
root@hbn03-host00:~# nv config apply
```

Flat Files (cl-acltool) Example for EVPN-L3 Stateful ACLs

For the same NVUE EVPN-L3 stateful ACLs example cited under "[NVUE Example for EVPN-L3 Stateful ACLs](#)" (HTTP server at IP address 21.1.1.2 accessible over EVPN routed overlay network), the following are the corresponding ACL rules which must be configured under `/etc/cumulus/acl/policy.d/<rule_name.rules>` followed by invoking `cl-acltool -i`.

1. Configure an ingress ACL rule matching with TCP flow details and CONNTRACK state of NEW, ESTABLISHED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` with the ingress interface as the SVI interface, followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL allow_tcp_conn_from_host in dir inbound on interface vlan105 ##
-t filter -A FORWARD -i vlan105 -p tcp -d 21.1.1.2/32 --dport 80 -m conntrack --
ctstate EST,NEW -m connmark ! --mark 7998 -j CONNMARK --set-mark 7999
-t filter -A FORWARD -i vlan105 -p tcp -d 21.1.1.2/32 --dport 80 -m conntrack --
ctstate EST,NEW -j ACCEPT
```

Note

As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for ingress ACL rules are protocol dependent: 7999 for TCP, 7997 for UDP, and 7995 for ICMP.

2. Configure an egress ACL rule matching with TCP and CONNTRACK state of ESTABLISHED, RELATED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` file with the egress interface as the SVI interface, followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL allow_tcp_resp_from_server in dir outbound on interface vlan105 ##
-t filter -A FORWARD -o vlan105 -p tcp -s 21.1.1.2/32 --sport 80 -m conntrack --
ctstate EST -j CONNMARK --set-mark 7998
-t filter -A FORWARD -o vlan105 -p tcp -s 21.1.1.2/32 --sport 80 -m conntrack --
ctstate EST -j ACCEPT
```

Note

As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for egress ACL rules are protocol dependent: 7998 for TCP, 7996 for UDP, and 7994 for ICMP.

DHCP Relay on HBN

DHCP is a client server protocol that automatically provides IP hosts with IP addresses and other related configuration information. A DHCP relay (agent) is a host that forwards DHCP packets between clients and servers. DHCP relays forward requests and replies between clients and servers that are not on the same physical subnet.

DHCP relay can be configured using either flat file (supervisord configuration) or through NVUE.

Configuration

HBN is a non-systemd based container. Therefore, the DHCP relay must be configured as explained in the following subsections.

Flat File Configuration (Supervisord)

The HBN initialization script installs default configuration files on BlueField in `/var/lib/hbn/etc/supervisor/conf.d/`. BlueField directory is mounted to `/etc/supervisor/conf.d` which achieves configuration persistence.

By default, DHCP relay is disabled. Default configuration applies to one instance of DHCPv4 relay and DHCPv6 relay in the default VRF.

NVUE Configuration

The user can use NVUE to configure and maintain DHCPv4 and DHCPv6 relays with CLI and REST API. NVUE generates all the required configurations and maintains the relay service.

DHCPv4 Relay Configuration

NVUE Example

The following configuration starts a relay service which listens for the DHCP messages on `p0_sf`, `p1_sf`, and `vlan482` and relays the requests to DHCP server `10.89.0.1` with gateway-interface as `lo`.

```
nv set service dhcp-relay default gateway-interface lo
nv set service dhcp-relay default interface p0_sf
nv set service dhcp-relay default interface p1_sf
nv set service dhcp-relay default interface vlan482 downstream
nv set service dhcp-relay default server 10.89.0.1
```

Flat Files Example

```
[program: isc-dhcp-relay-default]
command = /usr/sbin/dhcrelay --nl -d -i p0_sf -i p1_sf -id vlan482 -U lo 10.89.0.1
autostart = true
autorestart = unexpected
```

```
startsecs = 3
startretries = 3
exitcodes = 0
stopsignal = TERM
stopwaitsecs = 3
```

Where:

Option	Description
-i	Network interface to listen on for requests and replies
-iu	Upstream network interface
-id	Downstream network interface
-U [address]%%i fname	Gateway IP address interface. Use %% for IP%%ifname. % is used as an escape character.
--loglevel- debug	Debug logging. Location: /var/log/syslog.
-a	Append an agent option field to each request before forwarding it to the server with default values for circuit-id and remote-id
-r remote-id	Set a custom remote ID string (max of 255 chars). To use this option, you must also enable the -a option.
--use-pif- circuit-id	Set the underlying physical interface which receives the packet as the circuit-id. To use this option you must also enable the -a option.

DHCPv4 Relay Option 82

NVUE Example

The following NVUE command is used to enable option 82 insertion in DHCP packets with default values:

```
nv set service dhcp-relay default agent enable on
```

To provide a custom remote-id (e.g., host10) using NVUE:

```
nv set service dhcp-relay default agent remote-id host10
```

To use the underlying physical interface on which the request is received as circuit-id using NVUE:

```
nv set service dhcp-relay default agent use-pif-circuit-id enable on
```

Flat Files Example

```
[program: isc-dhcp-relay-default]
command = /usr/sbin/dhcrelay --nl -d -i p0_sf -i p1_sf -id vlan482 -U lo -a --use-pif-
circuit-id -r host10 10.89.0.1
autostart = true
autorestart = unexpected
startsecs = 3
startretries = 3
exitcodes = 0
stopsignal = TERM
stopwaitsecs = 3
```

DHCPv6 Relay Configuration

NVUE Example

The following NVUE command starts the DHCPv6 Relay service which listens for DHCPv6 requests on vlan482 and sends relayed DHCPv6 requests towards p0_sf and p1_sf.

```
nv set service dhcp-relay6 default interface downstream vlan482
nv set service dhcp-relay6 default interface upstream p0_sf
nv set service dhcp-relay6 default interface upstream p1_sf
```

Flat Files Example

```
[program: isc-dhcp-relay6-default]
command = /usr/sbin/dhcrelay --nl -6 -d -l vlan482 -u p0_sf -u p1_sf
autostart = true
autorestart = unexpected
startsecs = 3
startretries = 3
exitcodes = 0
stopsignal = TERM
stopwaitsecs = 3
```

Where:

Option	Description
-l [address]%%ifname[#index]	Downstream interface. Use %% for IP%%ifname. % is used as escape character.
-u [address]%%ifname	Upstream interface. Use %% for IP%%ifname. % is used as escape character.
-6	IPv6
--loglevel-debug	Debug logging located at /var/log/syslog

DHCP Relay and VRF Considerations

DHCP relay can be spawned inside a VRF context to handle the DHCP requests in that VRF. There can only be 1 instance each of DHCPv4 relay and DHCPv6 relay per VRF. To achieve that, the user can follow these guidelines:

- DHCPv4 on default VRF:

```
/usr/sbin/dhcrelay --nl -i <interface> -U [address]%%<interface> <server_ip>
```

- DHCPv4 on VRF:

```
/usr/sbin/ip vrf exec <vrf> /usr/sbin/dhcrelay --nl -i <interface> -U  
[address]%%<interface> <server_ip>
```

- DHCPv6 on default VRF:

```
/usr/sbin/dhcrelay --nl -6 -l <interface> -u <interface>
```

- DHCPv6 on VRF:

```
/usr/sbin/ip vrf exec <vrf> /usr/sbin/dhcrelay --nl -6 -l <interface> -u  
<interface>
```

Troubleshooting

HBN Container Stuck in init-sfs

The HBN container starts as `init-sfs` and should transition to `doca-hbn` within 2 minutes as can be seen using `crictl ps`. But sometimes it may remain as `init-sfs`.

This can happen if interface `p0_sf` is missing. Run the command `ip -br link show dev p0_sf` in BlueField and inside the container to check if `p0_sf` is present or not. If its missing, make sure the firmware is upgraded to the latest version. Perform [BlueField system-level reset](#) for the new firmware to take effect.

Host-side PF/VF Down After BlueField Reboot

In general, the host can use any interface manager to manage host interfaces belonging to BlueField. When the host uses an interface manager other than Netplan or NetworkManager, some ports may remain down after BlueField reboot.

Apply the following workaround if interfaces stay down:

1. Restart openibd:

```
systemctl restart openibd
```

2. Recreate SR-IOV interfaces if they are needed.

3. Replay interface config. For example:

- If using ifupdown2:

```
ifreload -a
```

- If using Netplan:

```
netplan apply
```

BGP Session not Establishing

One of the main causes of a BGP session not getting established is a mismatch in MTU configuration. Make sure the MTU on all interfaces is the same. For example, if BGP is failing on `p0`, check and verify that there is a matching MTU value for `p0`, `p0_sf_r`, `p0_sf`, and the remote peer of `p0`.

Generating Support Dump

HBN support dump can be generated using the `cl-support` command, inside the HBN container:

```
root@bf2:/tmp# cl-support
Please send /var/support/cl_support_bf2-s02-1-ipmi_20221025_180508.txz to
Cumulus support
```

The generated dump would be available in `/var/support` in the HBN container and would contain any process core dump as well as log files.

The `/var/support` directory is also mounted on the BlueField Arm side at `/var/lib/hbn/var/support`.

SFC Troubleshooting

To troubleshoot flows going through SFC interfaces, the first step is to disable the `nl2doca` service in the HBN container:

```
root@bf2:/tmp# supervisorctl stop nl2doca
nl2doca: stopped
```

Stopping `nl2doca` effectively stops hardware offloading and switches to software forwarding. All packets would appear on `tcpdump` capture on BlueField interfaces.

`tcpdump` can be performed on SF interfaces as well as VLAN, VXLAN, and uplinks to determine where a packet gets dropped or which flow a packet is taking.

General nl2doca Troubleshooting

The following steps can be used to make sure the `nl2doca` daemon is up and running:

1. Make sure there are no errors in the `nl2doca` log file at `/var/log/hbn/nl2docad.log`.
2. To check the status of the `nl2doca` daemon under supervisor, run:

```
supervisorctl status nl2doca
```

3. Use `ps` to check that the actual `nl2doca` process is running:

```
ps -eaf | grep nl2doca
root 18 1 0 06:31 ? 00:00:00 /bin/bash /usr/bin/nl2doca-docker-start
root 1437 18 0 06:31 ? 00:05:49 /usr/sbin/nl2docad
```

4. The core file should be in `/var/support/core/`.
5. Check if the `/cumulus/nl2docad/run/stats/punt` is accessible. Otherwise, `nl2doca` may be stuck and should be restarted:

```
supervisorctl restart nl2doca
```

nl2doca Offload Troubleshooting

If a certain traffic flow does not work as expected, disable `nl2doca` (i.e., disable hardware offloading):

```
supervisorctl stop nl2doca
```

With hardware offloading disabled, you can confirm it is an offloading issue if the traffic starts working. If it is not an offloading issue, use `tcpdump` on various interfaces to see where the packet gets dropped.

Offloaded entries can be checked in following files, which contain the programming status of every IP prefix and MAC address known to system.

- Bridge entries are available in the file `/cumulus/nl2docad/run/software-tables/17`. It includes all the MAC addresses in the system including local and remote MAC addresses.

Example format:

```
- flow-entry: 0xaaab0cef4190
flow-pattern:
fid: 112
dst mac: 00:00:5e:00:01:01
flow-actions:
```



```
SET VRF: 2
OUTPUT-PD-PORT: 20(TO_RTR_INTF)
STATS:
pkts: 1719
bytes: 191286
```

- Router entries are available in the file `/cumulus/nl2docad/run/software-tables/18`. It includes all the IP prefixes known to the system.

Example format for Entry with ECMP:

```
Entry with ECMP:
- flow-entry: 0xaaaada723700
flow-pattern:
IPV6: LPM
VRF: 0
destination-ip: ::/0
flow-actions :
ECMP: 2
STATS:
pkts: 0
bytes: 0
```

```
Entry without ECMP: - flow-entry: 0xaaaada7e1400
flow-pattern:
IPV4: LPM
VRF: 0
destination-ip: 60.1.0.93/32
flow-actions :
SET FID: 200
SMAC: 00:04:4b:a7:88:00
DMAC: 00:03:00:08:00:12
OUTPUT-PD-PORT: 19(TO_BR_INTF)
STATS:
pkts: 0
```

```
bytes: 0
```

- ECMP entries are available in the file `/cumulus/nl2docad/run/software-tables/19`. It includes all the next hops in the system.

Example format:

```
- ECMP: 2
ref-count: 2
num-next-hops: 2
entries:
- { index: 0, fid: 4100, src mac: 'b8:ce:f6:99:49:6a', dst mac: '00:02:00:00:00:0a' }
- { index: 1, fid: 4101, src mac: 'b8:ce:f6:99:49:6b', dst mac: '00:02:00:00:00:0e' }
```

To check counters for packets going to the kernel, run:

```
cat /cumulus/nl2docad/run/stats/punt
PUNT miss pkts:3154 bytes:312326
PUNT miss drop pkts:0 bytes:0
PUNT control pkts:31493 bytes:2853186
PUNT control drop pkts:0 bytes:0
ACL PUNT pkts:68 bytes:7364
ACL drop pkts:0 bytes:0
```

For a specific type of packet flow, programming can be referenced in block specific files. The typical flow is as follows:

For example, to check L2 EVPN ENCAP flows for remote MAC `8a:88:d0:b1:92:b1` on port `pf0vf0_sf`, the basic offload flow should look as follows: RxPort (`pf0vf0_sf`) -> BR (Overlay) -> RTR (Underlay) -> BR (Underlay) -> TxPort (one of the uplink `p0_sf` or `p1_sf` based on ECMP hash).

Step-by-step procedure:

1. Navigate to the interface file `/cumulus/nl2docad/run/software-tables/20`.

2. Check for the RxPort (pf0vf0_sf):

```
Interface: pf0vf0_sf
PD PORT: 6
HW PORT: 16
NETDEV PORT: 11
Bridge-id: 61
Untagged FID: 112
```

FID 112 is given to the receive port.

3. Check the bridge table file /cumulus/nl2docad/run/software-tables/17 with destination MAC 8a:88:d0:b1:92:b1 and FID 112:

```
flow-pattern:
fid: 112
dst mac: 8a:88:d0:b1:92:b1
flow-actions:
VXLAN ENCAP:
ENCAP dst ip: 6.0.0.26
ENCAP vni id: 1000112
SET VRF: 0
OUTPUT-PD-PORT: 20(TO_RTR_INTF)
STATS:
pkts: 100
bytes: 10200
```

4. Check the router table file /cumulus/nl2docad/run/software-tables/18 with destination IP 6.0.0.26 and VRF 0:

```
flow-pattern:
IPV4: LPM
VRF: 0
ip dst: 6.0.0.26/32
flow-actions :
```

```
ECMP: 1
OUTPUT PD PORT: 2(TO_BR_INTF)
STATS:
pkts: 300
bytes: 44400
```

5. Check the ECMP table file `/cumulus/nl2docad/run/software-tables/19` with ECMP 1:

```
- ECMP: 1
ref-count: 7
  num-next-hops: 2
  entries:
- { index: 0, fid: 4100, src mac: 'b8:ce:f6:99:49:6a', dst mac: '00:02:00:00:00:2f' }
- { index: 1, fid: 4115, src mac: 'b8:ce:f6:99:49:6b', dst mac: '00:02:00:00:00:33' }
```

6. The ECMP hash calculation picks one of these paths for next-hop rewrite. Check bridge table file for them (fid=4100, dst mac: 00:02:00:00:00:2f or fid=4115, dst mac: 00:02:00:00:00:33):

```
flow-pattern:
fid: 4100
dst mac: 00:02:00:00:00:2f
flow-actions:
OUTPUT-PD-PORT: 36(p0_sf)
STATS:
pkts: 1099
bytes: 162652
```

This will show the packet going out on the uplink.

NVUE Troubleshooting

To check the status of the NVUE daemon, run:

```
supervisorctl status nvued
```

To restart the NVUE daemon, run:

```
supervisorctl restart nvued
```

NVIDIA DOCA Management Service Guide

This guide provides instructions on how to use the DOCA Management Service on top of NVIDIA® BlueField® Networking Platform or ConnectX® Network Adapters.

Note

DOCA DMS service is currently supported at Alpha level.

Introduction

DOCA Management Service (DMS) is a one-stop shop for the user to configure and operate NVIDIA BlueField and ConnectX devices. DMS governs all scripts/tools of NVIDIA with an easy and industry-standard API created by the OpenConfig community. The user can configure BlueField or ConnectX for any mode whether locally (`ssh`) or remotely (`grpc`). It makes it easy to migrate and bootstrap any customer for any NVIDIA network device.

DMS exposes configurable BlueField/ConnectX parameters over the external interface to support a management station in an automated configuration of the NVIDIA Network Adapters. The exposed interface presents a uniform approach for BF/CX device configuration and keeps hidden details about the internal tools used for the configuration of BlueField or ConnectX features.

The DMS is a Client-Server architecture. Using a daemon, the service handles the discovery of resources, and is ready to receive commands from clients, the user can use DMSc (DMS Client) which delivers as part of the DMS, or use/create any other client.

i Info

Please refer to the [OpenConfig](#) site for an explanation of the OpenConfig protocol.

The Yang models describe a config tree which is easy to navigate and find any "config leaf" using XPath capabilities. Most gNMI/gNOI protocols are common with the OpenConfig community, utilizing gRPC protocol for transferring the command.

i Note

The DOCA Yang model is experimental.

i Note

The gNMI Subscribe mechanism for streaming telemetry is not currently supported yet.

i Info

DMS can run either on the host machine where BlueField or ConnectX devices are installed or on BlueField Arm itself (when [BlueField is operating in DPU mode](#)).

Requirements

DMS requires DOCA to be installed on the target system, where DMS Service will be running:

- DMS for Host - requires DOCA for Host package to be installed on the host system (with doca-networking or doca-all profiles).
- DMS for DPU (BlueField Arm) - requires DOCA Image to be installed on BlueField Arm.

Please follow these instructions to install DOCA: [NVIDIA DOCA Installation Guide for Linux](#).

Note

DMS supports only Linux-based environments today.

Service Deployment

DMS has 3 major components:

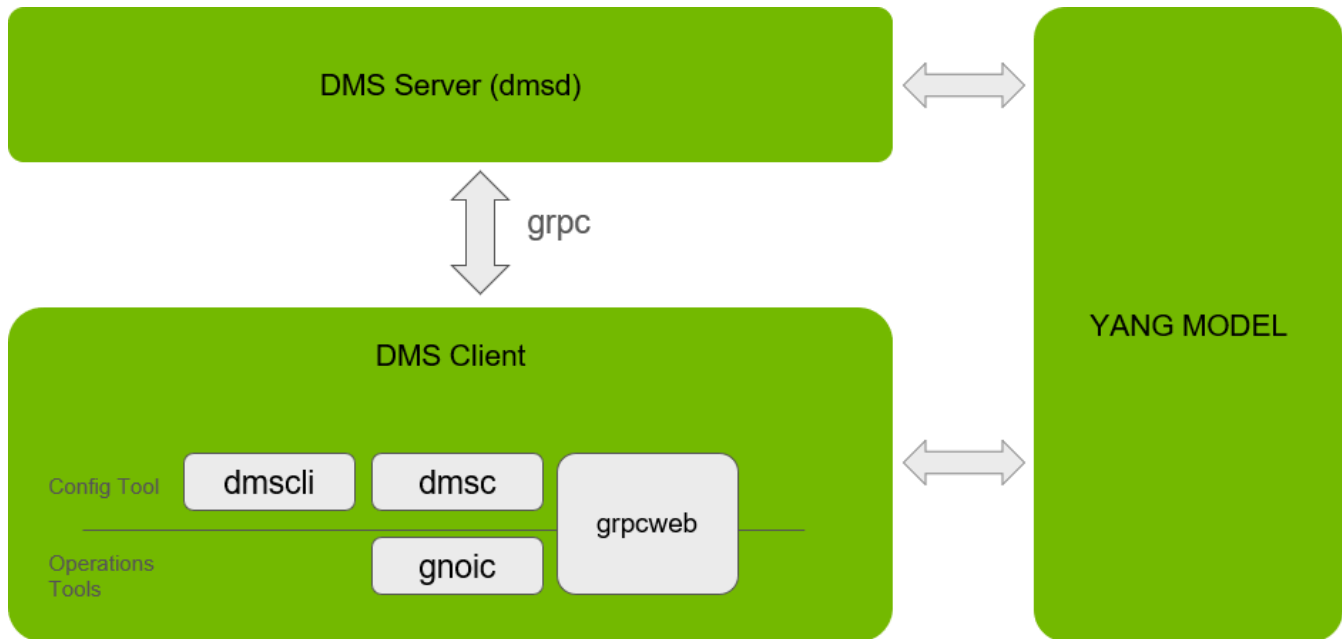
- DMSd – Server – DMS server inside the BlueField or on the host with an NVIDIA PCIe device
- DMSc – Client – DOCA provides OpenConfig client. Customers can choose to use this client, any other open-source client, or develop their own (gRPC-based) client.
- Yang files – Yang model files contain the data model used to configure the BlueField device, NVIDIA-specific extension to [common OpenConfig YANG Models](#).

OpenConfig consists of 2 main protocols:

- gNMI – gRPC Network Management Interface, protocol to configure of network device.

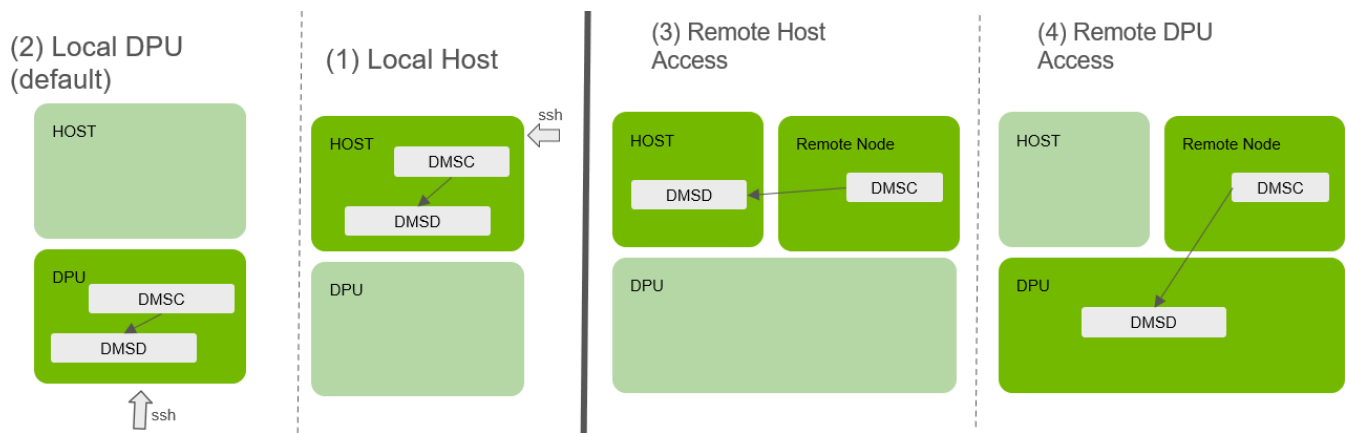
- gNOI – gRPC Network Operations Interface, a protocol to perform operational commands on network device (i.e., provision, upgrade, reboot).

The following is an architectural diagram of DMS:



The following diagram presents the DMS mode of operation, as the DMS client can operate from anywhere:

1. Both DMS client and server components are deployed on the Host
2. Both DMS client and server components are deployed on DPU (BlueField Arm)
3. DMS server component is deployed on the Host, while DMS client is deployed remotely (connecting to DMS server over management network)
4. DMS server component is deployed on DPU (BlueField Arm), while DMS client is deployed remotely (connecting to DMS server over management network)



Configuration

To see the full list of flags, user the help flag (i.e., `dmsd -help`, `dmsd -h`).

General Flags

- `-bind_address <string>` – Bind to `<address>:<port>` or just `:<port>` (default is `:9339`). Can be `localhost` for local use case, or an IP address for remote use case.
- `-v <value>` – log level for V logs
- `-target_pci <string>` – The target PCIe address (i.e., `03:00`). Auto-select if only one NVIDIA network device is present; otherwise, the PCIe address must be specified.

Security Flags

`-auth string` – this flag has 3 options:

- Shadow
 - Zero-touch, admin not required to create any dedicated additional user for DMS (re-use OS user)
 - Read the hashed password in real time on each client request
 - Use flags `-username -shadow`
 - Example: `-username root -shadow /etc/shadow/`

- To disable: `-noauth` flag
- Credentials
 - Admin must set a strong password
 - Use flags `-username -password`
 - Example: `-username root -password 123456`
 - To disable: `-noauth` flag
 - Can leave password flag empty to invoke prompt for password at demon boot
- Certificate File
 - The most secure option, based on (m)TLS
 - Example: `-ca /tmp/ca.crt -ca_key /tmp/ca.key`
 - To disable: `-notls` option

Provisioning Flags

- `-target_pci <string>` – The target PCIe address (i.e., 03:00). Auto-select if only one NVIDIA network device is present; otherwise, the PCIe address must be specified.
- `-image_folder <string>` – Specify image install folder. Can copy images directly to the folder to avoid transfer over the net. Default create folder: `/tmp/dms`.
- `-chunk_size_ack <uint>` – The chunk size of the image to respond with a `TransfreResponse` in bytes (default: 12000000)

Description

gNMI Command

In DMSc, the gNMI part is powered by the [GNMIC](#) project.

Info

For more information, please refer to [GNMIC documentation](#).

```
dmisc -a localhost:9339 -u root -p <password> --  
file /opt/mellanox/doca/service/dms/yang <command>
```

Prompt mode with autocomplete options can be invoked using the command `prompt`.

Get Request

Get requests happen in real-time without cache. Get command require providing the Yang Xpath as described in the following:

```
dmisc <flags> get --path /interfaces/interface[name=p0]/config/mtu  
[  
{  
  "source": "localhost:9339",  
  "timestamp": 1712485149723248511,  
  "time": "2024-04-07T10:19:09.723248511Z",  
  "updates": [  
    {  
      "Path": "interfaces/interface[name=p0]/config/mtu",  
      "values": {  
        "interfaces/interface/config/mtu": "1500"  
      }  
    }  
  ]  
}
```

Info

To insert params in the path, as an indication of the interface name (p0).

Set Request

Note

Some set commands cannot currently be detected with GET commands.

Set requests happen immediately, invoking tools to configure the OS.

Set commands require providing Yang Xpath as described in the following:

```
dmisc <flags> set --update /interfaces/interface[name=p0]/config/mtu:::int:::9216
{
  "source": "localhost:9339",
  "time": "1970-01-01T00:00:00Z",
  "results": [
    {
      "operation": "UPDATE",
      "path": "interfaces/interface[name=p0]/config/mtu"
    }
  ]
}
```

i Info

To insert params in the path, as an indication of the interface name (p0).

i Note

The value provided must be separated by value type and char.

i Note

Currently, only the --update flag is supported in set.

It is also possible to invoke a command JSON list:

```
dmsc <flags> set --request-file req.json
```

req.json example:

```
{
  "updates":
  [
    {
      "path": "/interfaces/interface[name=p0]/config/mtu",
      "value": 9216,
      "encoding": "uint"
    },
    {
```

```
"path": "/interfaces/interface[name=p0]/config/enabled",
"value": true,
"encoding": "bool"
}
]
}
```

gNOI Commands

In DMSc, the gNOI part is powered by [GNOIC](#) project, for full docs refer to [GNOIC docs](#)

```
dmsc -a localhost --port 9339 --tls-cert client.crt --tls-key client.key <command>
```

Prompt mode with autocomplete options can be invoked using the command `prompt`.

All commands are blocking unless specified otherwise.

OS

The following subsections present actions for provisioning a new DOCA Image (BFB) or firmware on BlueField.

Install

This command transmits the file from the client to the server and authenticates the file's validity:

```
dmsc <flags> os install --version <free_text_version> --pkg <bfb | cfg | fw path>
dmsc <flags> os install --version 2_7_0 --pkg DOCA_2.7.0_Ubuntu.bfb
dmsc <flags> os install --version 2_7_0 --pkg config.cfg
dmsc <flags> os install --version 1_3_5_custom.bfb --pkg custom.bfb
```

The file is saved to the folder specified in the `-image_folder` flag (default `/tmp/dms`) if the file authenticates successfully. The file's extension is autodetected and is written automatically if none is provided in the `--version` field. Users may copy the file to the folder

manually and invoke the command with file extension to authenticate the file. No file transfer is initiated if the file already exists in the folder and the version specified with the extension.

Activate

Activate the command deploy the BFB bundle/firmware to the hardware:

```
dmisc <flags> os activate --version 2_7_0 # Invoke all files under 2_7_0 name
dmisc <flags> os activate --version "2_7_0.bfb;0_0_1.cfg;24_29_0046.fw"
```

The `--version` flag provides a version to search for in the folder specified by the `-image_folder` flag (default `/tmp/dms`). If no extension is provided, the command uses all files under the version name.

To activate separate files, use the `--version` flag separated by semi-colon.

Note

After running the command to activate firmware, firmware reset is automatically invoked.

Verify

Verify command retrieves the firmware and BFB bundle version:

```
dmisc <flags> os verify
```

The return value consists of both versions separated by semi-colon.

(i) Note

Currently, the BFB bundle can only be retrieved if it was installed via DMS.

System

The following subsections provide actions for rebooting the BFB bundle/firmware on the BlueField.

(i) Note

Alpha version does not support components

Reboot Status

Verify BFB is on reboot operation

```
dmisc <flags> system reboot-status
```

The value returned is `false` if the system is active. It is `true` if the system is in reboot status.

If the status cannot be retrieved, the status appears as a failure and the message field indicates what the issue is.

Reboot

Reboot the BlueField Arm and firmware.


```
dmisc <flags> system reboot --delay 10s
```

This command is not blocking and returns immediately.

The flag `--delay` specifies the time interval to wait before invoking the reset.

NVIDIA DOCA Telemetry Service Guide

This guide provides instructions on how to use the DOCA Telemetry Service (DTS) container on top of NVIDIA® BlueField® DPU.

Introduction

DOCA Telemetry Service (DTS) collects data from built-in providers and from external telemetry applications. The following providers are available:

- Data providers:
 - sysfs
 - ethtool
 - tc (traffic control)
- Aggregation providers:
 - fluent_aggr
 - prometheus_aggr

Note

Sysfs provider is enabled by default.

DTS stores collected data into binary files under the `/opt/mellanox/doca/services/telemetry/data` directory. Data write is disabled by default due to BlueField storage restrictions.

DTS can export the data via Prometheus Endpoint (pull) or Fluent Bit (push).

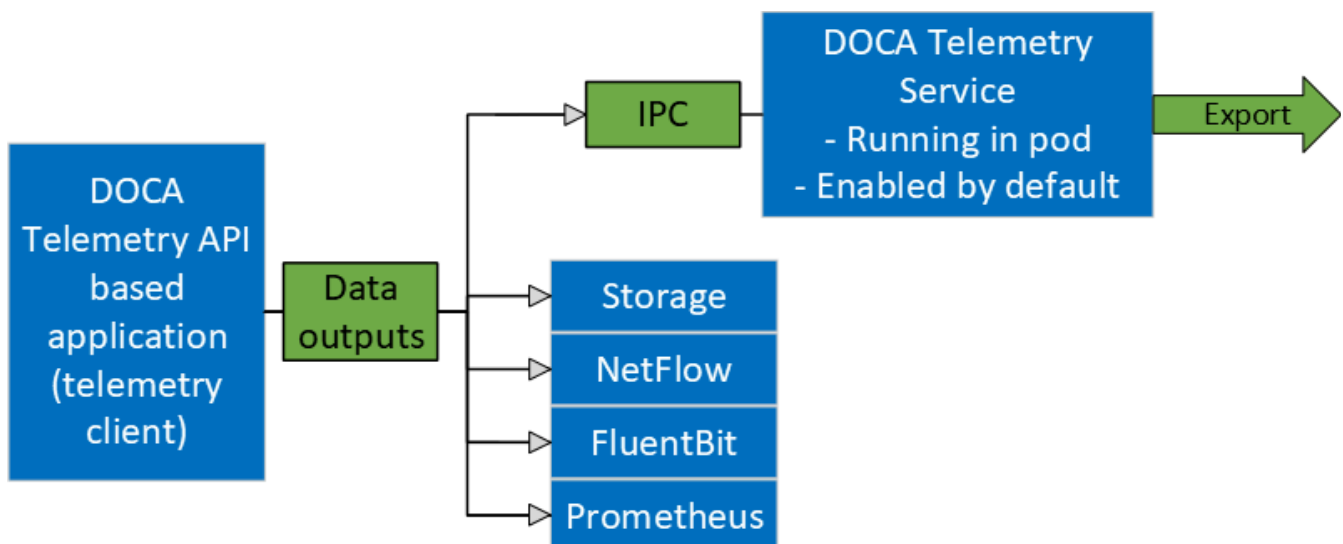
DTS allows exporting NetFlow packets when data is collected from the DOCA Telemetry NetFlow API client application. NetFlow exporter is enabled from `dts_config.ini` by setting NetFlow collector IP/address and port.

Service Deployment

Available Images

Built-in DOCA Service Image

DOCA Telemetry Service is enabled by default on the DPU and is shipped as part of the BlueField image. That is, every BlueField image contains a fixed service version so as to provide out-of-the-box support for programs based on the [DOCA Telemetry](#) library.



DOCA Service on NGC

In addition to the built-in image shipped with the BlueField boot image, DTS is also available on [NGC](#), NVIDIA's container catalog. This is useful in case a new version of the

service has been released and the user wants to upgrade from the built-in image. For service-specific configuration steps and deployment instructions, refer to the service's [container page](#).

Info

For more information about the deployment of DOCA containers on top of the BlueField DPU, refer to [NVIDIA DOCA Container Deployment Guide](#).

DPU Deployment

As mentioned above, DTS starts automatically on BlueField boot. This is done according to the .yaml file located at `/etc/kubelet.d/doca_telemetry_standalone.yaml`. Removing the .yaml file from this path stops the automatic DTS boot.

DTS files can be found under the directory `/opt/mellanox/doca/services/telemetry/`.

- Container folder mounts:
 - config
 - data
 - ipc_sockets
- Backup files:
 - `doca_telemetry_service_${version}_arm64.tar.gz` – DTS image
 - `doca_telemetry_standalone.yaml` – copy of the default boot .yaml file

Host Deployment

DTS supports x86_64 hosts. The providers and exporters all run from a single docker container.

1. Initialize and configure host DTS with the desired DTS version:

```
export DTS_IMAGE=nvcr.io/nvidia/doca/doca_telemetry:<desired-DTS-version>
docker run -v "/opt/mellanox/doca/services/telemetry/config:/config" --rm --
name doca-telemetry-init -it $DTS_IMAGE /bin/bash -c "DTS_CONFIG_DIR=host
/usr/bin/telemetry-init.sh"
```

Note

Per NGC policy, the "latest" tag does not exist. This means that when deploying DTS, the user must pick the desired tag from NGC and ensure that the `DTS_IMAGE` variable points to the full image. Example from version 1.16.5-doca2.6.0-host:

```
export
DTS_IMAGE=nvcr.io/nvidia/doca/doca_telemetry:1.16.5-
doca2.6.0-host
```

2. Run with:

```
docker run -d --net=host --uts=host --ipc=host \
--privileged \
--ulimit stack=67108864 --ulimit memlock=-1 \
--device=/dev/mst/ \
--device=/dev/infiniband/ \
--gpus all \
-v "/opt/mellanox/doca/services/telemetry/config:/config" \
-v "/opt/mellanox/doca/services/telemetry/ipc_sockets:/tmp/ipc_sockets" \
```

```
-v "/opt/mellanox/doca/services/telemetry/data:/data" \  
-v "/usr/lib/mft:/usr/lib/mft" \  
-v "/sys/kernel/debug:/sys/kernel/debug" \  
--rm --name doca-telemetry -it $DTS_IMAGE /usr/bin/telemetry-run.sh
```

Note

The following mounts are required by specific services only:

- hcapperf provider:
 - --device=/dev/mst/
 - -v "/usr/lib/mft:/usr/lib/mft"
 - -v "/sys/kernel/debug:/sys/kernel/debug"
- UCX/RDMA export modes:
 - --device=/dev/infiniband/
- GPU providers (nvidia-smi and dcgm):
 - --gpu all

Deployment with Grafana Monitoring

Refer to section "[Deploying with Grafana Monitoring](#)".

Configuration

The configuration of DTS is placed under `/opt/mellanox/doca/services/telemetry/config` by DTS during initialization. The user can interact with the `dts_config.ini` file and `fluent_bit_configs`

folder. `dst_config.ini` contains the main configuration for the service and must be used to enable/disable providers, exporters, data writing. More details are provided in the corresponding sections. For every update in this file, DST must be restarted. Interaction with `fluent_bit_configs` folder is described in section [Fluent Bit](#).

Init Scripts

The `InitContainers` section of the `.yaml` file has 2 scripts for config initialization:

- `/usr/bin/telemetry-init.sh` – generates the default configuration files if, and only if, the `/opt/mellanox/doca/services/telemetry/config` folder is empty.
- `/usr/bin/enable-fluent-forward.sh` – configures the destination host and port for Fluent Bit forwarding. The script requires that both the host and port are present, and only in this case it would start. The script overwrites the `/opt/mellanox/doca/services/telemetry/config/fluent_bit_configs` folder and configures the `.exp` file.

Enabling Fluent Bit Forwarding

To enable Fluent Bit forward, add the destination host and port to the command line found in the `initContainers` section of the `.yaml` file:

```
command: ["/bin/bash", "-c", "/usr/bin/telemetry-init.sh && /usr/bin/enable-fluent-forward.sh -i=127.0.0.1 -p=24224"]
```

Note

The host and port shown above are just an example. See section [Fluent Bit](#) to learn about manual configuration.

Generating Configuration

The configuration folder `/opt/mellanox/doca/services/telemetry/config` starts empty by default. Once the service starts, the initial scripts run as a part of the initial container and create configuration as described in section [Enabling Fluent Bit Forwarding](#).

Resetting Configuration

Resetting the configuration can be done by deleting the content found in the configuration folder and restarting the service to generate the default configuration.

Enabling Providers

Providers are enabled from the `dts_config.ini` configuration file. Uncomment the `enable-provider=$provider-name` line to allow data collection for this provider. For example, uncommenting the following line enables the `ethtool` provider:

```
#enable-provider=ethtool
```

Note

More information about telemetry providers can be found under the [Providers](#) section.

Remote Collection

Certain providers or components are unable to execute properly within the container due to various container limitations. Therefore, they would have to perform remote collection or execution.

The following steps enable remote collection:

1. Activate DOCA privileged executer (DPE), as DPE is how remote collection is achieved:

```
systemctl start dpe
```

2. Add `grpc` before provider-name (i.e., `enable-provider=grpc.$provider-name`). For example, the following line configures remote collection of the `hcaperf` provider:

```
enable-provider=grpc.hcaperf
```

3. If there are any configuration lines that are provider-specific, then add the `grpc` prefix as well. Building upon the previous example:

```
grpc.hcaperf.mlx5_0=sample  
grpc.hcaperf.mlx5_1=sample
```

Enabling Data Write

Uncomment the following line in `dts_config.ini`:

```
#output=/data
```

Note

Changes in `dts_config.ini` force the main DTS process to restart in 60 seconds to apply the new settings.

Enabling IPC with Non-container Program

For information on enabling IPC between DTS and an application that runs outside of a container, refer to section "[Using IPC with Non-container Application](#)" in the [DOCA Telemetry](#).

Description

Providers

DTS supports on-board data collection from `sysfs`, `ethtool`, and `tc` providers.

Fluent and Prometheus aggregator providers can collect the data from other applications.

Sysfs Counters List

The `sysfs` provider has several components: `ib_port`, `hw_port`, `mr_cache`, `eth`, `hwmon` and `bf_ptm`. By default, all the components (except `bf_ptm`) are enabled when the provider is enabled:

```
#disable-provider=sysfs
```

The components can be disabled separately. For instance, to disable `eth`:

```
enable-provider=sysfs
disable-provider=sysfs.eth
```

Note

`ib_port` and `ib_hww` are state counters which are collected per port. These counters are only collected for ports whose state is active.

- `ib_port` counters:

```
{hca_name}:{port_num}:ib_port_state
{hca_name}:{port_num}:VL15_dropped
{hca_name}:{port_num}:excessive_buffer_overrun_errors
{hca_name}:{port_num}:link_downed
{hca_name}:{port_num}:link_error_recovery
{hca_name}:{port_num}:local_link_integrity_errors
{hca_name}:{port_num}:multicast_rcv_packets
{hca_name}:{port_num}:multicast_xmit_packets
{hca_name}:{port_num}:port_rcv_constraint_errors
{hca_name}:{port_num}:port_rcv_data
{hca_name}:{port_num}:port_rcv_errors
{hca_name}:{port_num}:port_rcv_packets
{hca_name}:{port_num}:port_rcv_remote_physical_errors
{hca_name}:{port_num}:port_rcv_switch_relay_errors
{hca_name}:{port_num}:port_xmit_constraint_errors
{hca_name}:{port_num}:port_xmit_data
{hca_name}:{port_num}:port_xmit_discards
{hca_name}:{port_num}:port_xmit_packets
{hca_name}:{port_num}:port_xmit_wait
{hca_name}:{port_num}:symbol_error
{hca_name}:{port_num}:unicast_rcv_packets
{hca_name}:{port_num}:unicast_xmit_packets
```

- `ib_hw` counters:

```
{hca_name}:{port_num}:hw_state
{hca_name}:{port_num}:hw_duplicate_request
{hca_name}:{port_num}:hw_implied_nak_seq_err
{hca_name}:{port_num}:hw_lifespan
{hca_name}:{port_num}:hw_local_ack_timeout_err
{hca_name}:{port_num}:hw_out_of_buffer
{hca_name}:{port_num}:hw_out_of_sequence
{hca_name}:{port_num}:hw_packet_seq_err
```

```
{hca_name}:{port_num}:hw_req_cqe_error  
{hca_name}:{port_num}:hw_req_cqe_flush_error  
{hca_name}:{port_num}:hw_req_remote_access_errors  
{hca_name}:{port_num}:hw_req_remote_invalid_request  
{hca_name}:{port_num}:hw_resp_cqe_error  
{hca_name}:{port_num}:hw_resp_cqe_flush_error  
{hca_name}:{port_num}:hw_resp_local_length_error  
{hca_name}:{port_num}:hw_resp_remote_access_errors  
{hca_name}:{port_num}:hw_rnr_nak_retry_err  
{hca_name}:{port_num}:hw_rx_atomic_requests  
{hca_name}:{port_num}:hw_rx_dct_connect  
{hca_name}:{port_num}:hw_rx_icrc_encapsulated  
{hca_name}:{port_num}:hw_rx_read_requests  
{hca_name}:{port_num}:hw_rx_write_requests
```

- `ib_mr_cache` counters:

```
{hca_name}:mr_cache:size_{n}:cur  
{hca_name}:mr_cache:size_{n}:limit  
{hca_name}:mr_cache:size_{n}:miss  
{hca_name}:mr_cache:size_{n}:size
```

Note

Where n ranges from 0 to 24.

- `eth` counters:

```
{hca_name}:{device_name}:eth_collisions  
{hca_name}:{device_name}:eth_multicast  
{hca_name}:{device_name}:eth_rx_bytes  
{hca_name}:{device_name}:eth_rx_compressed
```

```
{hca_name}:{device_name}:eth_rx_crc_errors
{hca_name}:{device_name}:eth_rx_dropped
{hca_name}:{device_name}:eth_rx_errors
{hca_name}:{device_name}:eth_rx_fifo_errors
{hca_name}:{device_name}:eth_rx_frame_errors
{hca_name}:{device_name}:eth_rx_length_errors
{hca_name}:{device_name}:eth_rx_missed_errors
{hca_name}:{device_name}:eth_rx_nohandler
{hca_name}:{device_name}:eth_rx_over_errors
{hca_name}:{device_name}:eth_rx_packets
{hca_name}:{device_name}:eth_tx_aborted_errors
{hca_name}:{device_name}:eth_tx_bytes
{hca_name}:{device_name}:eth_tx_carrier_errors
{hca_name}:{device_name}:eth_tx_compressed
{hca_name}:{device_name}:eth_tx_dropped
{hca_name}:{device_name}:eth_tx_errors
{hca_name}:{device_name}:eth_tx_fifo_errors
{hca_name}:{device_name}:eth_tx_heartbeat_errors
{hca_name}:{device_name}:eth_tx_packets
{hca_name}:{device_name}:eth_tx_window_errors
```

- BlueField-2 hwmon counters:

```
{hwmon_name}:{l3cache}:CYCLES
{hwmon_name}:{l3cache}:HITS_BANK0
{hwmon_name}:{l3cache}:HITS_BANK1
{hwmon_name}:{l3cache}:MISSES_BANK0
{hwmon_name}:{l3cache}:MISSES_BANK1
{hwmon_name}:{pcie}:IN_C_BYTE_CNT
{hwmon_name}:{pcie}:IN_C_PKT_CNT
{hwmon_name}:{pcie}:IN_NP_BYTE_CNT
{hwmon_name}:{pcie}:IN_NP_PKT_CNT
{hwmon_name}:{pcie}:IN_P_BYTE_CNT
{hwmon_name}:{pcie}:IN_P_PKT_CNT
{hwmon_name}:{pcie}:OUT_C_BYTE_CNT
```

```
{hwmon_name}:{pcie}:OUT_C_PKT_CNT
{hwmon_name}:{pcie}:OUT_NP_BYTE_CNT
{hwmon_name}:{pcie}:OUT_NP_PKT_CNT
{hwmon_name}:{pcie}:OUT_P_PKT_CNT
{hwmon_name}:{tile}:MEMORY_READS
{hwmon_name}:{tile}:MEMORY_WRITES
{hwmon_name}:{tile}:MSS_NO_CREDIT
{hwmon_name}:{tile}:VICTIM_WRITE
{hwmon_name}:{tilenet}:CDN_DIAG_C_OUT_OF_CRED
{hwmon_name}:{tilenet}:CDN_REQ
{hwmon_name}:{tilenet}:DDN_REQ
{hwmon_name}:{tilenet}:NDN_REQ
{hwmon_name}:{trio}:TDMA_DATA_BEAT
{hwmon_name}:{trio}:TDMA_PBUF_MAC_AF
{hwmon_name}:{trio}:TDMA_RT_AF
{hwmon_name}:{trio}:TPIO_DATA_BEAT
{hwmon_name}:{triogen}:TX_DAT_AF
{hwmon_name}:{triogen}:TX_DAT_AF
```

- BlueField-3 hwmon counters:

```
{hwmon_name}:{llt}:GDC_BANK0_RD_REQ
{hwmon_name}:{llt}:GDC_BANK1_RD_REQ
{hwmon_name}:{llt}:GDC_BANK0_WR_REQ
{hwmon_name}:{llt}:GDC_BANK1_WR_REQ
{hwmon_name}:{llt_miss}:GDC_MISS_MACHINE_RD_REQ
{hwmon_name}:{llt_miss}:GDC_MISS_MACHINE_WR_REQ
{hwmon_name}:{mss}:SKYLIB_DDND_TX_FLITS
{hwmon_name}:{mss}:SKYLIB_DDND_RX_FLITS
```

- BlueField-3 bf_ptm counters:

```
bf:ptm:active_power_profile
bf:ptm:atx_power_available
bf:ptm:core_temp
```

```
bf:ptm:ddr_temp
bf:ptm:error_state
bf:ptm:power_envelope
bf:ptm:power_throttling_event_count
bf:ptm:power_throttling_state
bf:ptm:thermal_throttling_event_count
bf:ptm:thermal_throttling_state
bf:ptm:throttling_state
bf:ptm:total_power
bf:ptm:vr0_power
bf:ptm:vr1_power
```

Power Thermal Counters

The `bf_ptm` component collects BlueField-3 power thermal counters using [remote collection](#). It is disabled by default and can be enabled as follows:

1. Load kernel module `mlxbf-ptm`:

```
modprobe -v mlxbf-ptm
```

2. Enable component using remote collection:

```
enable-provider=grpc.sysfs.bf_ptm
```

Note

DPE server should be active before changing the `dts_config.ini` file. See section "[Remote Collection](#)" for details.

Ethtool Counters

Ethtool counters is the generated list of counters which corresponds to [Ethtool utility](#). Counters are generated on a per-device basis. See [this community post](#) for more information on mlx5 ethtool counters.

Traffic Control Info

The following TC objects are supported and reported regarding the ingress filters:

- Filters
 - [flower](#)
- Actions
 - [mirred](#)
 - [tunnel_key](#)

The info is provided as one of the following events:

- Basic filter event
- Flower/IPv4 filter event
- Flower/IPv6 filter event
- Basic action event
- Mirred action event
- Tunnel_key/IPv4 action event
- Tunnel_key/IPv6 action event

General notes:

- Actions always belong to a filter, so action events share the filter event's ID via the `event_id` data member

- Basic filter event only contains textual *kind* (so users can see which real life objects' support they are lacking)
- Basic action event only contains textual *kind* and some basic common statistics if available

Fluent Aggregator

fluent_aggr listens on a port for [Fluent Bit Forward protocol](#) input connections. Received data can be streamed via a [Fluent Bit](#) exporter.

The default port is 42442. This can be changed by updating the following option:

```
fluent-aggr-port=42442
```

Prometheus Aggregator

prometheus_aggr polls data from a list of Prometheus endpoints.

Each endpoint is listed in the following format:

```
prometheus_aggr_endpoint.{N}={host_name},{host_port_url},{poll_inteval_msec}
```

Where N starts from 0.

Aggregated data can be exported via a Prometheus Aggr Exporter endpoint.

Network Interfaces

ifconfig collects network interface data. To enable, set:

```
enable-provider=ifconfig
```


If the Prometheus endpoint is enabled, add the following configuration to cache every collected network interface and arrange the index according to their names:

```
prometheus-fset-indexes=name
```

Metrics are collected for each network interface as follows:

```
name
rx_packets
tx_packets
rx_bytes
tx_bytes
rx_errors
tx_errors
rx_dropped
tx_dropped
multicast
collisions
rx_length_errors
rx_over_errors
rx_crc_errors
rx_frame_errors
rx_fifo_errors
rx_missed_errors
tx_aborted_errors
tx_carrier_errors
tx_fifo_errors
tx_heartbeat_errors
tx_window_errors
rx_compressed
tx_compressed
rx_nohandler
```

HCA Performance

hcaperf collects HCA performance data. Since it requires access to an RDMA device, it must use [remote collection](#) on the DPU. On the host, the user runs the container in privileged mode and RDMA device mount.

The counter list is device dependent.

hcaperf DPU Configuration

To enable hcaperf in [remote collection](#) mode, set:

```
enable-provider=grpc.hcaperf

# specify HCAs to sample
grpc.hcaperf.mlx5_0=sample
grpc.hcaperf.mlx5_1=sample
```

Note

DPE server should be active before changing the `dts_config.ini` file. See section "[Remote Collection](#)" for details.

hcaperf Host Configuration

To enable hcaperf in regular mode, set:

```
enable-provider=hcaperf

# specify HCAs to sample
hcaperf.mlx5_0=sample
hcaperf.mlx5_1=sample
```

NVIDIA System Management Interface

The `nvidia-smi` provider collects GPU and GPU process information provided by the NVIDIA system management interface.

This provider is supported only on `x86_64` hosts with installed GPUs. All GPU cards supported by `nvidia-smi` are supported by this provider.

The counter list is GPU dependent. Additionally, per-process information is collected for the first 20 (by default) `nvidia_smi_max_processes` processes.

Counters can be either collected as string data "as is" in `nvidia-smi` or converted to numbers when `nvsmi_with_numeric_fields` is set.

To enable `nvidia-smi` provider and change parameters, set:

```
enable-provider=nvidia-smi

# Optional parameters:
#nvidia_smi_max_processes=20
#nvsmi_with_numeric_fields=1
```

NVIDIA Data Center GPU Manager

The `dcm` provider collects GPU information provided by the NVIDIA data center GPU manager (DCGM) API.

This provider is supported only on `x86_64` hosts with installed GPUs, and requires running the `nv-hostengine` service (refer to [DCGM documentation](#) for details).

DCGM counters are split into several groups by context:

- GPU – basic GPU information (always)
- COMMON – common fields that can be collected from all devices
- PROF – profiling fields
- ECC – ECC errors

- NVLINK / NVSWITCH / VGPU – fields depending on the device type

To enable DCGM provider and counter groups, set:

```
enable-provider=dcgm  
  
dcgm_events_enable_common_fields=1  
#dcgm_events_enable_prof_fields=0  
#dcgm_events_enable_ecc_fields=0  
#dcgm_events_enable_nvlink_fields=0  
#dcgm_events_enable_nvswitch_fields=0  
#dcgm_events_enable_vgpu_fields=0
```

BlueField Performance

The `bfperf` provider collects calculated performance counters of BlueField Arm cores. It requires the executable `bfperf_pmc`, which is integrated in the DOCA BFB bundle of BlueField-3, as well as an active [DPE](#).

To enable BlueField performance provider, set:

```
enable-provider=bfperf
```

Note

When running, the `bfperf` provider is expected to recurrently reset the counters of the `sysfs.hwmon` component. Consider disabling it if `bfperf` is enabled.

Data Outputs

DTS can send the collected data to the following outputs:

- Data writer (saves binary data to disk)
- Fluent Bit (push-model streaming)
- Prometheus endpoint (keeps the most recent data to be pulled)

Data Writer

The data writer is disabled by default to save space on BlueField. Steps for activating data write during debug can be found under section [Enabling Data Write](#).

The schema folder contains JSON-formatted metadata files which allow reading the binary files containing the actual data. The binary files are written according to the naming convention shown in the following example (apt install tree):

```
tree /opt/mellanox/doca/services/telemetry/data/  
/opt/mellanox/doca/services/telemetry/data/  
  {year}  
    {mmdd}  
    {hash}  
    {source_id}  
      {source_tag}{timestamp}.bin  
    {another_source_id}  
      {another_source_tag}{timestamp}.bin  
  schema  
  schema_{MD5_digest}.json
```

New binary files appears when the service starts or when binary file age/size restriction is reached. If no schema or no data folders are present, refer to the [Troubleshooting](#) section.

Note

`source_id` is usually set to the machine hostname. `source_tag` is a line describing the collected counters, and it is often set as the provider's

name or name of user-counters.

Reading the binary data can be done from within the DTS container using the following command:

```
crictrl exec -it <Container ID> /opt/mellanox/collectx/bin/clx_read -s /data/schema  
/data/path/to/datafile.bin
```

Note

The path to the data file must be an absolute path.

Example output:

```
{  
  "timestamp": 1634815738799728,  
  "event_number": 0,  
  "iter_num": 0,  
  "string_number": 0,  
  "example_string": "example_str_1"  
}  
{  
  "timestamp": 1634815738799768,  
  "event_number": 1,  
  "iter_num": 0,  
  "string_number": 1,  
  "example_string": "example_str_2"  
}  
...
```

Prometheus

The Prometheus endpoint keeps the most recent data to be pulled by the Prometheus server and is enabled by default.

To check that data is available, run the following command on BlueField:

```
curl -s http://0.0.0.0:9100/metrics
```

The command dumps every counter in the following format:

```
counter_name {list of meta fields} counter_value timestamp
```

Additionally, endpoint supports JSON and CSV formats:

```
curl -s http://0.0.0.0:9100/json/metrics  
curl -s http://0.0.0.0:9100/csv/metrics
```

Note

The default port for Prometheus can be changed in `dts_config.ini`.

Configuration Details

Prometheus is configured as a part of `dts_config.ini`.

By default, the Prometheus HTTP endpoint is set to port 9100. Comment this line out to disable Prometheus export.

```
prometheus=http://0.0.0.0:9100
```

Prometheus can use the data field as an index to keep several data records with different index values. Index fields are added to Prometheus labels.

```
# Comma-separated counter set description for Prometheus indexing:  
#prometheus-indexes=idx1,idx2  
  
# Comma-separated fieldset description for prometheus indexing  
#prometheus-fset-indexes=idx1,idx2
```

The default `fset` index is `device_name`. It allows Prometheus to keep `ethtool` data up for both the `p0` and `p1` devices.

```
prometheus-fset-indexes=device_name
```

If `fset` index is not set, the data from `p1` overwrites `p0`'s data.

For quick name filtering, the Prometheus exporter supports being provided with a comma-separated list of counter names to be ignored:

```
#prometheus-ignore-names=counter_name1,counter_name_2
```

For quick filtering of data by tag, the Prometheus exporter supports being provided with a comma-separated list of data source tags to be ignored.

Users should add tags for all streaming data since the Prometheus exporter cannot be used for streaming. By default, `FI_metrics` are disabled.

```
prometheus-ignore-tags=FI_metrics
```

Prometheus Aggregator Exporter

Prometheus aggregator exporter is an endpoint that keeps the latest aggregated data using `prometheus_aggr`.

This exporter labels data according to its source.

To enable this provider, users must set 2 parameters in `dts_config.ini`:

```
prometheus-aggr-exporter-host=0.0.0.0
prometheus-aggr-exporter-port=33333
```

Fluent Bit

Fluent Bit allows streaming to multiple destinations. Destinations are configured in `.exp` files that are documented in-place and can be found under:

```
/opt/mellanox/doca/services/telemetry/config/fluent_bit_configs
```

Fluent Bit allows exporting data via "Forward" protocol which connects to the Fluent Bit/FluentD instance on customer side.

Export can be enabled manually:

1. Uncomment the line with `fluent_bit_configs=...` in `dts_config.ini`.
2. Set `enable=1` in required `.exp` files for the desired plugins.
3. Additional configurations can be set according to instructions in the `.exp` file if needed.
4. Restart the DTS.
5. Set up receiving instance of Fluent Bit/FluentD if needed.
6. See the data on the receiving side.

Export file destinations are set by configuring `.exp` files or creating new ones. It is recommended to start by going over documented example files. Documented examples exist for the following supported plugins:

- forward
- file
- stdout

- kafka
- es (elastic search)
- influx

Note

All `.exp` files are disabled by default if not configured by `initContainer` entry point through `.yaml` file.

Note

To forward the data to several destinations, create several `forward_{num}.exp` files. Each of these files must have their own destination host and port.

Export File Configuration Details

Each export destination has the following fields:

- `name` – configuration name
- `plugin_name` – Fluent Bit plugin name
- `enable` – 1 or 0 values to enable/disable this destination
- `host` – the host for Fluent Bit plugin
- `port` – port for Fluent Bit plugin
- `msgpack_data_layout` – the msgpacked data format. Default is `flb_std`. The other option is `custom`. See section [Msgpack Data Layout](#) for details.

- `plugin_key=val` – key-value pairs of Fluent Bit plugin parameter (optional)
- `counterset/fieldset` – file paths (optional). See details in section [Cset/Fset Filtering](#).
- `source_tag=source_tag1,source_tag2` – comma-separated list of data page source tags for filtering. The rest tags are filtered out during export. Event tags are event provider names. All counters can be enabled/disabled only simultaneously with a `counters` keyword.

Note

Use # to comment a configuration line.

Msgpack Data Layout

Data layout can be configured using `.exp` files by setting `msgpack_data_layout=layout`. There are two available layouts: Standard and Custom.

The standard `flb_std` data layout is an array of 2 fields:

- timestamp double value
- a plain dictionary (key-value pairs)

The standard layout is appropriate for all Fluent Bit plugins. For example:

```
[timestamp_val, {"timestamp"->ts_val, type=>"counters/events",
"source"=>"source_val", "key_1"=>val_1, "key_2"=>val_2,...}]
```

The custom data layout is a dictionary of meta-fields and counter fields. Values are placed into a separate plain dictionary. Custom data format can be dumped with `stdout_raw` output plugin of Fluent-Bit installed or can be forwarded with `forward` output plugin.

Counters example:

```
{"timestamp"=>timestamp_val, "type"=>"counters", "source"=>"source_val",  
"values"=> {"key_1"=>val_1, "key_2"=>val_2,...}}
```

Events example:

```
{"timestamp"=>timestamp_val, "type"=>"events", "type_name"=>"type_name_val",  
"source"=>" source_val", "values"=>{"key_1"=>val_1, "key_2"=>val_2,...}}
```

Cset/Fset Filtering

Each export file can optionally use one `cset` and one `fset` file to filter UFM telemetry counters and events data.

- `cset` contains tokens per line to filter data with `"type"="counters"`.
- `fset` contains several blocks started with the header line `[event_type_name]` and tokens under that header. An `Fset` file is used to filter data with `"type"="events"`.

Note

Event type names could be prefixed to apply the same tokens to all fitting types. For example, to filter all `ethtool` events, use `[ethtool_event_*]`.

If several tokens must be matched simultaneously, use `<tok1>+<tok2>+<tok3>`. Exclusive tokens are available as well. For example, the line `<tok1>+<tok2>-<tok3>-<tok4>` filters names that match both `tok1` and `tok2` and do not match `tok3` or `tok4`.

The following are the details of writing `cset` files:

```
# Put tokens on separate lines  
# Tokens are the actual name 'fragments' to be matched  
# port$ # match names ending with token "port"
```

```
# ^port # match names starting with token "port"
# ^port$ # include name that is exact token "port"
# port+xmit # match names that contain both tokens "port" and "xmit"
# port-support # match names that contain the token "port" and do not match the
"- " token "support"
#
# Tip: To disable counter export put a single token line that fits nothing
```

The following are the details of writing fset files:

```
# Put your events here
# Usage:
#
# [type_name_1]
# tokens
# [type_name_2]
# tokens
# [type_name_3]
# tokens
# ...
# Tokens are the actual name 'fragments' to be matched
# port$ # match names ending with token "port"
# ^port # match names starting with token "port"
# ^port$ # include name that is exact token "port"
# port+xmit # match names that contain both tokens "port" and "xmit"
# port-support # match names that contain the token "port" and do not match the
"- " token "support"

# The next example will export all the "tc" events and all events with type prefix
"ethtool_" "ethtool" are filtered with token "port":
# [tc]
#
# [ethtool_*]
# packet
```

```
# To know which event type names are available check export and find field
"type_name"=>"ethtool_event_p0"
# ...
# Corner cases:
# 1. Empty fset file will export all events.
# 2. Tokens written above/without [event_type] will be ignored.
# 3. If cannot open fset file, warning will be printed, all event types will be exported.
```

NetFlow Exporter

NetFlow exporter must be used when data is collected as NetFlow packets from the telemetry client applications. In this case, DOCA Telemetry NetFlow API sends NetFlow data packages to DTS via IPC. DTS uses NetFlow exporter to send data to the NetFlow collector (3rd party service).

To enable NetFlow exporter, set `netflow-collector-ip` and `netflow-collector-port` in `dts_config.ini`. `netflow-collector-ip` could be set either to IP or an address.

For additional information, refer to the `dts_config.ini` file.

DOCA Privileged Executer

DOCA Privileged Executer (DPE) is a daemon that allows specific DOCA services (DTS included) to access BlueField information that is otherwise inaccessible from a container due to technology limitations or permission granularity issues.

When enabled, DPE enriches the information collected by DTS. However, DTS can still be used if DPE is disabled (default).

DPE Usage

DPE is controlled by systemd, and can be used as follows:

- To check DPE status:

```
sudo systemctl status dpe
```

- To start DPE:

```
sudo systemctl start dpe
```

- To stop DPE:

```
sudo systemctl stop dpe
```

DPE logs can be found in `/var/log/doca/telemetry/dpe.log`.

DPE Configuration File

DPE can be configured by the user. This section covers the syntax and implications of its configuration file.

Note

The DPU telemetry collected by DTS does not require for this configuration file to be used.

The DPE configuration file allows users to define the set of commands that DPE should support. This may be done by passing the `-f` option in the following line of `/etc/systemd/system/dpe.service`:

```
ExecStart=/opt/mellanox/doca/services/telemetry/dpe/bin/dpeserver -vv
```

To use the configuration file:

```
ExecStart=/opt/mellanox/doca/services/telemetry/dpe/bin/dpeserver -vvw -f
/path/to/dpe_config.ini
```

The configuration file supports the following sections:

- [server] - list of key=value lines for general server configuration. Allowed keys: socket.
- [commands] - list of bash command lines that are not using custom RegEx
- [commands_regex] - list of bash command lines that are using custom RegEx
- [regex_macros] - custom RegEx definitions used in the commands_regex section

Consider the following example configuration file:

```
[server]
socket=/tmp/dpe.sock

[commands]
hostname
cat /etc/os-release

[commands_regex]
crictrl inspect $HEXA # resolved as "crictrl inspect [a-f0-9]+"
lspci $BDF # resolved as "lspci ([0-9a-f]{4}\: |)[0-9a-f]{2}\:[0-9a-f]{2}\.[0-9a-f]"

[regex_macros]
HEXA=[a-f0-9]+
BDF=([0-9a-f]{4}\: |)[0-9a-f]{2}\:[0-9a-f]{2}\.[0-9a-f]
```

Note

DPE is shipped with a preconfigured file that matches the commands used by the standalone DTS version included in the same DOCA

installation. The file is located in
`/opt/mellanox/doca/services/telemetry/dpe/etc/dpe_config.ini`.

Note

Using a DPE configuration file allows for a fine-grained control over the interface exposed by it to the rest of the DOCA services. However, even when using the pre-supplied configuration file mentioned above, one should remember that it has been configured to match a fixed DTS version. That is, replacing the standalone DTS version with a new one downloaded from NGC means that the used configuration file might not cover additional features added in the new DTS version.

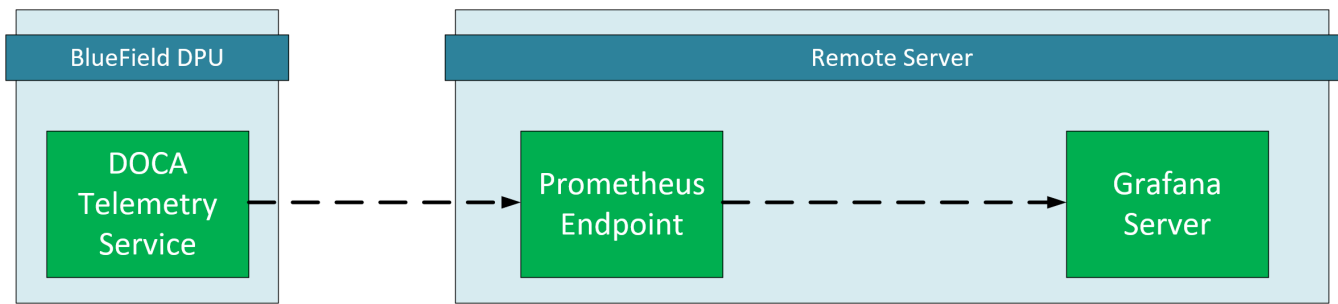
Deploying with Grafana Monitoring

This chapter provides an overview and deployment configuration of DOCA Telemetry Service with [Grafana](#).

Grafana Deployment Prerequisites

- BlueField DPU running DOCA Telemetry Service.
- Optional remote server to host Grafana and Prometheus.
- Prometheus installed on the host machine. Please refer to the [Prometheus website](#) for more information.
- Grafana installed on the host machine. Please refer to [Grafana Labs website](#) for more information.

Grafana Deployment Configuration



DTS Configuration (DPU Side)

DTS will be configured to export the sysfs counter using the Prometheus plugin.

i Note

Sysfs is used as an example, other counters are available. Please refer to the [.NVIDIA DOCA Telemetry Service Guide v2.7.0](#) for more information.

1. Make sure the sysfs counter is enabled.

```
vim /opt/mellanox/doca/services/telemetry/config/dts_config.ini  
  
enable-provider=sysfs
```

2. Enable Prometheus exporter by setting the prometheus address and port.

```
vim /opt/mellanox/doca/services/telemetry/config/dts_config.ini  
  
prometheus=http://0.0.0.0:9100
```

i Note

In this example, the Prometheus plugin exports data on localhost port 9100, this is an arbitrary value and can be changed.

Note

DTS must be restarted to apply changes.

Prometheus Configuration (Remote Server)

Please download Prometheus for your platform.

Prometheus is configured via command-line flags and a configuration file, `prometheus.yml`.

1. Open the `prometheus.yml` file and configure the DPU as the endpoint target.

```
vim prometheus.yml
# metrics_path defaults to '/metrics'
# scheme defaults to 'http'.

static_configs:
- targets: ["<dpu-ip>:<prometheus-port>"]
```

Where:

- `<dpu-ip>` is the DPU IP address. Prometheus reaches to this IP to pull data.
- `<prometheus-port>` the exporter port that set in [DTS configuration](#).

2. Run Prometheus server:

```
./prometheus --config.file="prometheus.yml"
```

Tip

Prometheus services are available as Docker images. Please refer to [Using Docker](#) in Prometheus' Installation guide.

Grafana Configuration (Remote Server)

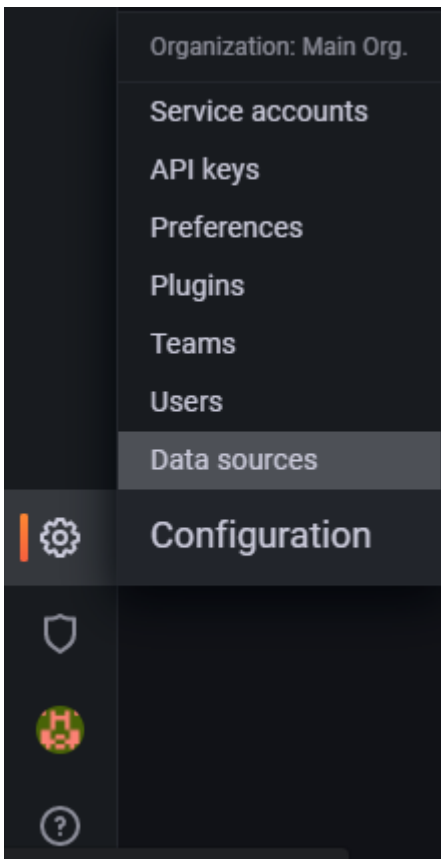
Please download and install Grafana for your platform.

1. Setup Grafana. Please refer to [Install Grafana](#) guide in Grafana documentation.
2. Log into the Grafana dashboard at <http://localhost:3000>.

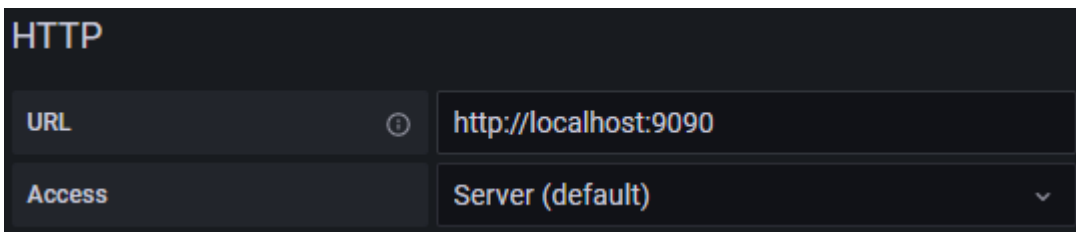
Note

Port 3000 is the default port number set by Grafana. This can be changed if needed. The default credentials are admin/admin.

3. Add Prometheus as data source by navigating to Settings > Data sources > Add data source > Prometheus.



4. Configure the Prometheus data source. Under the HTTP section, set the Prometheus server address.



Note

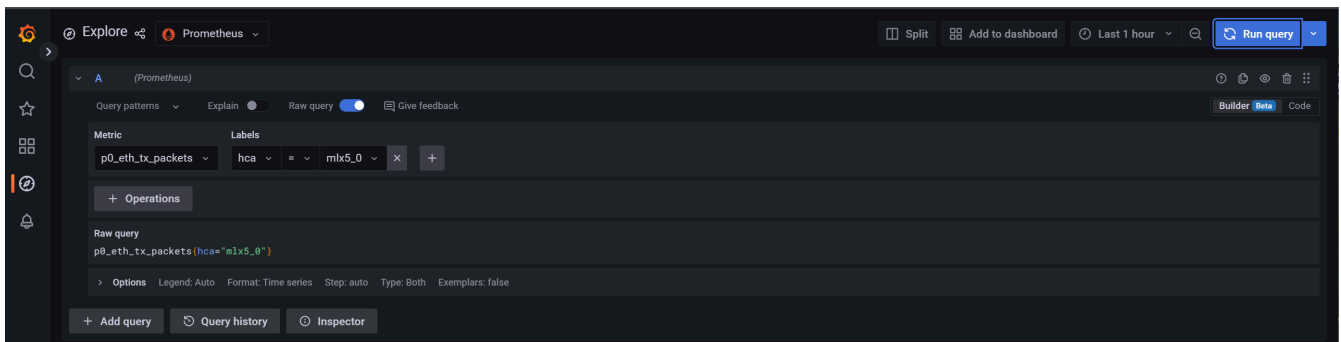
The Prometheus server's default listen port is 9090. Prometheus and Grafana are both running on the same server, thus the address is localhost.

5. Save and test.

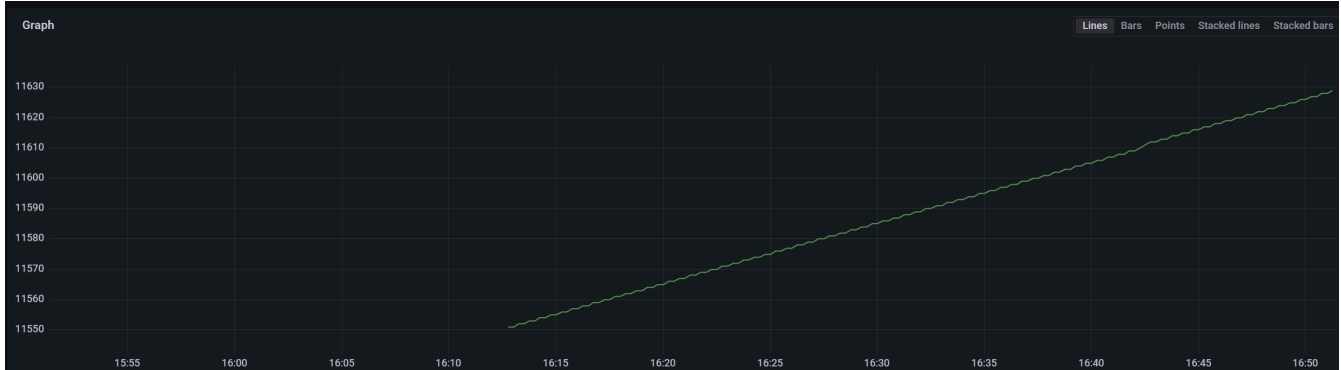
Exploring Telemetry Data

Go to the Explore page on the left-hand side, and choose a Prometheus provider.

Choose a metric to display and specify a label. The label can be used to filter out data based on the source and HCA devices.



Graph display after selecting a metric and specifying a label to filter by:



Troubleshooting

On top of the Troubleshooting section in the [NVIDIA DOCA Container Deployment Guide](#), here are additional troubleshooting tips for DTS:

- For general troubleshooting, refer to the [NVIDIA DOCA Troubleshooting Guide](#).
- If the pod's state fails to be marked as "Ready", refer to `/var/log/syslog`.
- Check if the service is configured to write data to the disk as this may cause the system to run out of disk space.

- If a PIC bus error occurs, configure the following files inside the container:

```
crictl exec -it <container-id> /bin/bash
# Add to /config/clx.env the following line:
"
export UCX_TLS=tcp
"
```

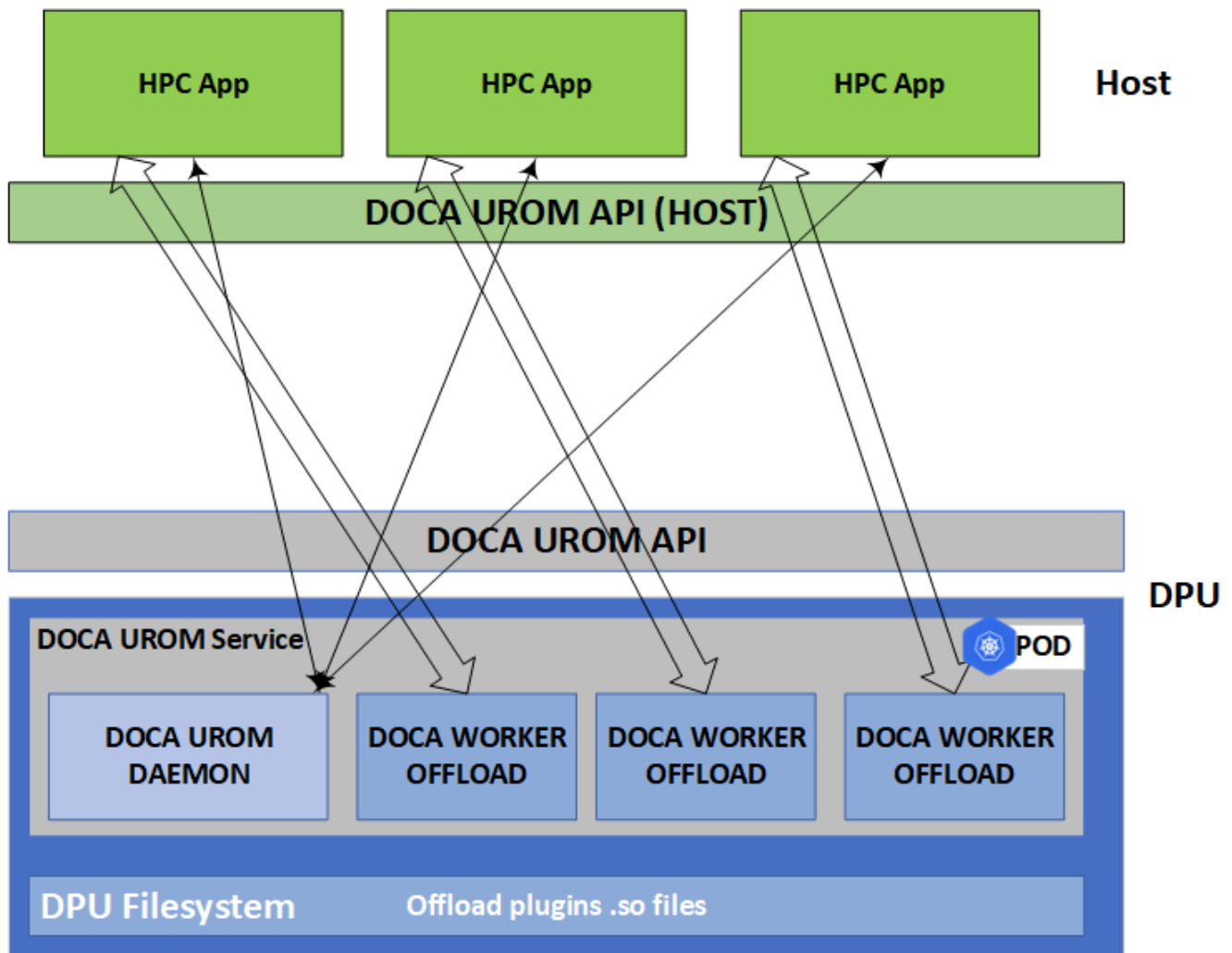
NVIDIA DOCA UROM Service Guide

This guide provides instructions on how to use the DOCA UROM Service on top of the NVIDIA® BlueField® networking platform.

Introduction

The DOCA UROM service provides a framework for offloading significant portions of HPC software stack directly from the host and to the BlueField device.

Using a daemon, the service handles the discovery of resources, the coordination between the host and BlueField, and the spawning, management, and teardown of the BlueField workers themselves.



The first step in initiating an offload request involves the UROM host application establishing a connection with the UROM service. Upon receiving the plugin discovery command, the UROM service responds by providing the application with a list of plugins available on the BlueField. The application then attaches the plugin IDs that correspond to the desired workers to their network identifiers. Finally, the service triggers UROM worker plugin instances on the BlueField to execute the parallel computing tasks. Within the service's Kubernetes pod, workers are spawned by the daemon in response to these offload requests. Each computation can utilize either a single library or multiple computational libraries.

Requirements

Before deploying the UROM service container, ensure that the following prerequisites are satisfied:

- Allocate huge pages as needed by DOCA (this requires root privileges):


```
$ sudo echo 2048 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

- Or alternatively:

```
$ sudo echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages  
$ sudo mkdir /mnt/huge  
$ sudo mount -t hugetlbfs nodev /mnt/huge
```

Service Deployment

For information about the deployment of DOCA containers on top of the BlueField, refer to the [NVIDIA BlueField Container Deployment Guide](#).

Service-specific configuration steps and deployment instructions can be found under the service's [container page](#).

Description

Plugin Discovery and Reporting

When the application initiates a connection request to the DOCA UROM Service, the daemon reads the `UROM_PLUGIN_PATH` environment variable. This variable stores directory paths to `.so` files for the plugins with multiple paths separated by semicolons. The daemon scans these paths sequentially and tries loading each `.so` file. Once the daemon finishes the scan, it reports the available BlueField plugins to the host application.

The host application gets the list of available plugins as a list of `doca_urom_service_plugin_info` structures:

```
struct doca_urom_service_plugin_info {  
    uint64_t id; // Unique ID to send commands to the plugin  
    uint64_t version; // Plugin version  
    char plugin_name[DOCA_UROM_PLUGIN_NAME_MAX_LEN]; // .so filename  
};
```

The UROM daemon is responsible for generating unique identifiers for the plugins, which are necessary to enable the worker to distinguish between different plugin tasks.

Loading Plugin in Worker

During the spawning of UROM workers by the UROM daemon, the daemon attaches a list of desired plugins in the worker command line. Each plugin is passed in a format of `so_path:id`.

As part of worker bootstrapping, the flow iterates all `.so` files and tries to load them by using `dlopen` system call and look for `urom_plugin_get_iface()` symbol to get the plugin operations interface.

Yaml File

The `.yaml` file downloaded from NGC can be easily edited according to users' needs:

```
env:  
# Service-Specific command line arguments  
- name: SERVICE_ARGS  
value: "-l 60 -m 4096"  
- name: UROM_PLUGIN_PATH  
value:  
"/opt/mellanox/doca/samples/doca_urom/plugins/worker_sandbox;/opt/mellanox/doca/samples/doca_u
```

- The `SERVICE_ARGS` are the runtime arguments received by the service:
 - `-l, --log-level <value>` – sets the (numeric) log level for the program `<10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>`
 - `--sdk-log-level` – sets the SDK (numeric) log level for the program `<10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>`
 - `-m, --max-msg-size` – specify UROM communication channel maximum message size

- The UROM_PLUGIN_PATH is an env variable that stores directory paths to .so files for the plugins

For each plugin on the BlueField, it is necessary to add a volume mount inside the service container. For example:

```
volumes:  
- name: urom-sandbox-plugin  
hostPath:  
  path: /opt/mellanox/doca/samples/doca_urom/plugins/worker_sandbox  
  type: DirectoryOrCreate  
...  
volumeMounts:  
- mountPath: /opt/mellanox/doca/samples/doca_urom/plugins/worker_sandbox  
  name: urom-sandbox-plugin
```

Troubleshooting

When troubleshooting a container deployment issues, it is highly recommended to follow the deployment steps and tips found in the "[Review Container Deployment](#)" section of the [NVIDIA BlueField Container Deployment Guide](#) .

One could also check the `/var/log/doca/urom` log files for more details about the running cycles of service components (daemon and workers).

The log file name for workers is `urom_worker_<pid>_dev.log` and for the daemon it is `urom_daemon_dev.log`.

Pod is Marked as "Ready" and No Container is Listed

Error

When deploying the container, the pod's STATE is marked as Ready and an image is listed, however, no container can be seen running:

```
$ sudo crictl pods
```

```
POD ID CREATED STATE NAME NAMESPACE ATTEMPT RUNTIME
3162b71e67677 4 seconds ago Ready          doca-urom-my-dpu
default 0 (default)
```

```
$ sudo crictl images
```

```
IMAGE TAG IMAGE ID SIZE
```

```
k8s.gcr.io/pause 3.2          2a060e2e7101d 487kB
```

```
nvcr.io/nvidia/doca/doca_urom 1.0.0-doca2.7.0 2af1e539eb7ab 86.8MB
```

```
$ sudo crictl ps
```

```
CONTAINER IMAGE CREATED STATE NAME ATTEMPT POD ID POD
```

Solution

In most cases, the container did start but immediately exited. This could be checked using the following command:

```
$ sudo crictl ps -a
```

```
CONTAINER IMAGE CREATED STATE NAME ATTEMPT POD ID POD
```

```
556bb78281e1d 2af1e539eb7ab 6 seconds ago Exited doca-urom 1
```

```
3162b71e67677 doca-urom-my-dpu
```

Should the container fail (i.e., reporting a state of `Exited`), it is recommended to examine the UROM's main log at `/var/log/doca/urom/urom_daemon_dev.log`.

In addition, for a short period of time after termination, the container logs could also be viewed using the container's ID:

```
$ sudo crictl logs 556bb78281e1d
```

```
...
```

Pod is Not Listed

Error

When placing the container's YAML file in the Kubelet's input folder, the service pod is not listed in the list of pods:

```
$ sudo crictl pods
POD ID CREATED STATE NAME NAMESPACE ATTEMPT RUNTIME
```

Solution

In most cases, the pod has not started because of the absence of the requested hugepages. This can be verified using the following command:

```
$ sudo journalctl -u kubelet -e. . .
Oct 04 12:12:19 <my-dpu> kubelet[2442376]: I1004 12:12:19.905064 2442376
predicate.go:103] "Failed to admit pod, unexpected error while attempting to recover from
admission failure" pod="default/doca-urom-service-<my-dpu>" err="preemption: error finding a set
of pods to preempt: no set of running pods found to reclaim resources: [(res: hugepages-2Mi, q:
104563999874), ]"
```

DOCA Switching

NVIDIA® BlueField® and NVIDIA® ConnectX® platforms provide robust support for diverse applications through hardware-based offloads, offering unparalleled scalability, performance, and efficiency.

This section lists the extensive switching capabilities enabled by DOCA libraries and services on these platforms. It includes detailed configurations of Open Virtual Switch (OVS) such as the setup of representors, virtualization options, and optional bridge configurations. These subsections guide users through the steps to effectively implement these software components.

DOCA Representors Model

Note

This model is only applicable when the BlueField is operating [DPU mode](#).

BlueField® DPU uses netdev representors to map each one of the host side physical and virtual functions:

1. Serve as the tunnel to pass traffic for the virtual switch or application running on the Arm cores to the relevant PF or VF on the Arm side.
2. Serve as the channel to configure the embedded switch with rules to the corresponding represented function.

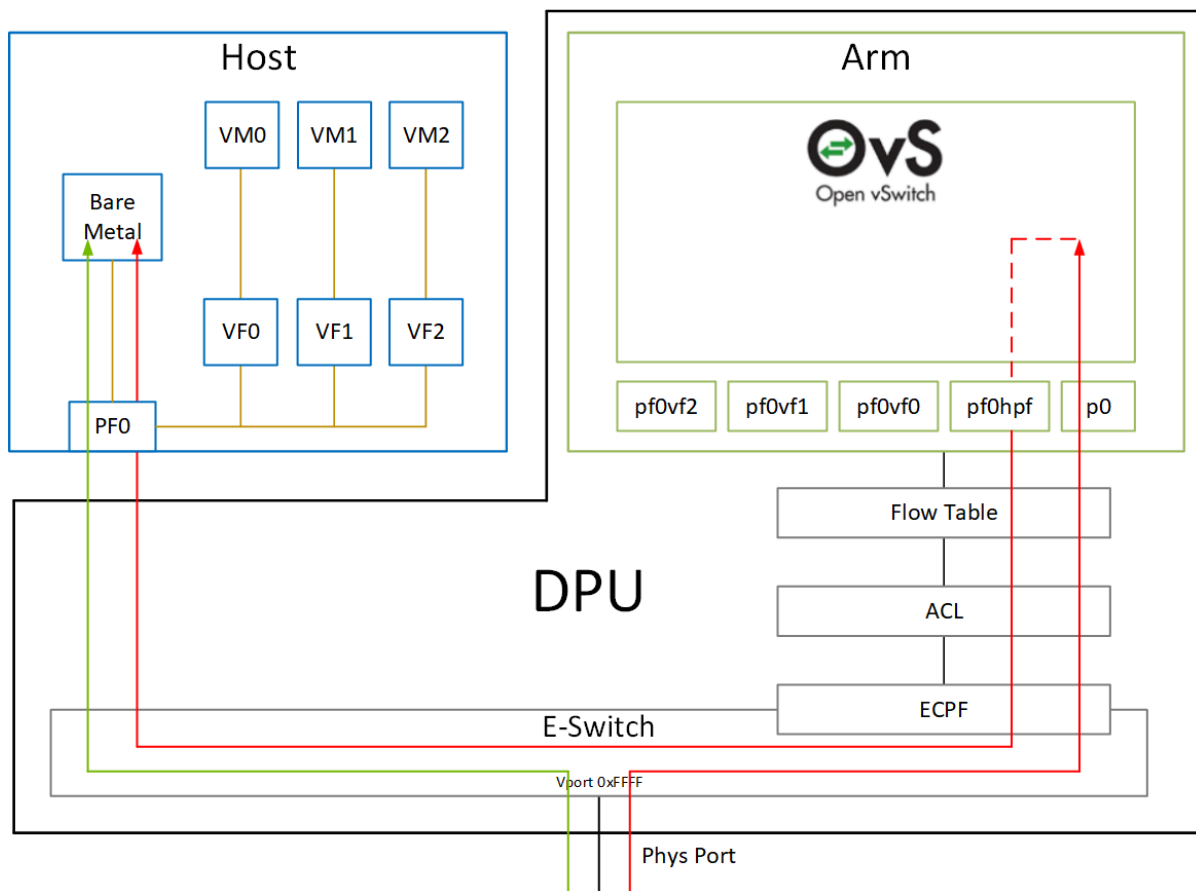
Those representors are used as the virtual ports being connected to OVS or any other virtual switch running on the Arm cores.

When in ECPF ownership mode, we see 2 representors for each one of the DPU's network ports: one for the uplink, and another one for the host side PF (the PF representor

created even if the PF is not probed on the host side). For each one of the VFs created on the host side a corresponding representor would be created on the Arm side. The naming convention for the representors is as follows:

- Uplink representors: p<port_number>
- PF representors: pf<port_number>hpf
- VF representors: pf<port_number>vf<function_number>

The diagram below shows the mapping of between the PCI functions exposed on the host side and the representors. For the sake of simplicity, we show a single port model (duplicated for the second port).



The red arrow demonstrates a packet flow through the representors, while the green arrow demonstrates the packet flow when steering rules are offloaded to the embedded switch. More details on that are available in the switch offload section.

Note

The MTU of host functions (PF/VF) must be smaller than the MTUs of both the uplink and corresponding PF/VF representor. For example, if the host PF MTU is set to 9000, both uplink and PF representor must be set to above 9000.

This section contains the following pages:

- [OpenvSwitch Offload \(OVS in DOCA\)](#)
- [VirtIO Acceleration through Hardware vDPA](#)
- [Bridge Offload](#)
- [Link Aggregation](#)
- [Controlling Host PF and VF Parameters](#)

OpenvSwitch Offload (OVS in DOCA)

Info

Note on naming conventions:

- OVS – Refers to the Open vSwitch distribution within DOCA framework
- OVS-DOCA – Describes the datapath offloading layer (DPIF) that utilizes the DOCA Flow library for offloading tasks. This layer is a component of OVS, along with additional DPIF implementations that facilitate offloading via DPDK or Kernel, known respectively as OVS-DPDK and OVS-Kernel.

Tip

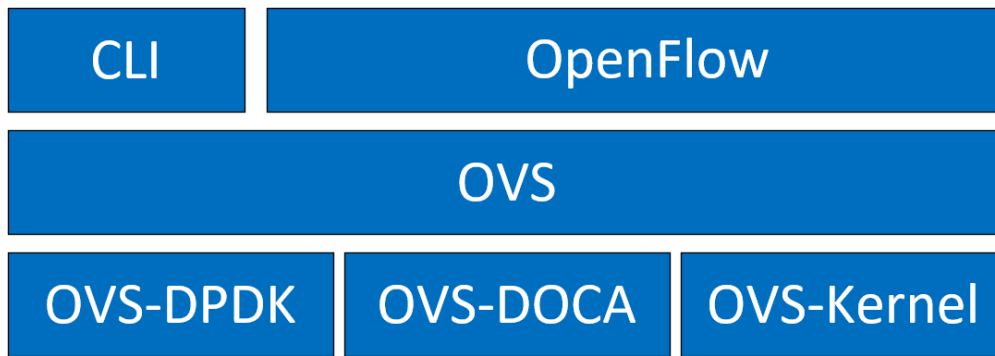
NVIDIA advises utilizing the OVS-DOCA DPIF to maximize efficiency, performance, scalability, and feature support.

Warning

The DPDK and Kernel DPIFs are maintained in their current form primarily for backward compatibility and are not planned to be updated with new features.

Open vSwitch (OVS) is a software-based network technology that enhances virtual machine (VM) communication within internal and external networks. Typically deployed in the hypervisor, OVS employs a software-based approach for packet switching, which can strain CPU resources, impacting system performance and network bandwidth utilization. Addressing this, NVIDIA's Accelerated Switching and Packet Processing (ASAP²) technology offloads OVS data-plane tasks to specialized hardware, like the embedded switch (eSwitch) within the NIC subsystem, while maintaining an unmodified OVS control-plane. This results in notably improved OVS performance without burdening the CPU.

NVIDIA's DOCA-OVS extends the traditional OVS-DPDK and OVS-Kernel data-path offload interfaces (DPIF), introducing OVS-DOCA as an additional DPIF implementation. DOCA-OVS, built upon NVIDIA's networking API, preserves the same interfaces as OVS-DPDK and OVS-Kernel while utilizing the DOCA Flow library with the additional OVS-DOCA DPIF. Unlike the use of the other DPIFs (DPDK, Kernel), OVS-DOCA DPIF exploits unique hardware offload mechanisms and application techniques, maximizing performance and features for NVIDIA NICs and DPUs. This mode is especially efficient due to its architecture and DOCA library integration, enhancing e-switch configuration and accelerating hardware offloads beyond what the other modes can achieve.



NVIDIA OVS installation contains all three OVS flavors. The following subsections describe the three flavors (default is OVS-Kernel) and how to configure each of them.

OVS and Virtualized Devices

When OVS is combined with NICs and DPUs (such as NVIDIA® ConnectX®-6 Lx/Dx and NVIDIA® BlueField®-2 and later), it utilizes the hardware data plane of ASAP². This data plane can establish connections to VMs using either SR-IOV virtual functions (VFs) or virtual host data path acceleration (vDPA) with virtio.

In both scenarios, an accelerator engine within the NIC accelerates forwarding and offloads the OVS rules. This integrated solution accelerates both the infrastructure (via VFs through SR-IOV or virtio) and the data plane. For DPUs (which include a NIC subsystem), an alternate virtualization technology implements full virtio emulation within the DPU, enabling the host server to communicate with the DPU as a software virtio device.

- When using ASAP² data plane over SR-IOV virtual functions (VFs), the VF is directly passed through to the VM, with the NVIDIA driver running within the VM.
- When using vDPA, the vDPA driver allows VMs to establish their connections through VirtIO. As a result, the data plane is established between the SR-IOV VF and the standard virtio driver within the VM, while the control plane is managed on the host by the vDPA application.

VirtIO Acceleration through Hardware vDPA

Hardware vDPA Installation

Hardware vDPA requires QEMU v2.12 (or with upstream 6.1.0) and DPDK v20.11 as minimal versions.

To install QEMU:

1. Clone the sources:

```
git clone https://git.qemu.org/git/qemu.git
cd qemu
git checkout v2.12
```

2. Build QEMU:

```
mkdir bin
cd bin
../configure --target-list=x86_64-softmmu --enable-kvm
make -j24
```

To install DPDK:

1. Clone the sources:

```
git clone git://dpdk.org/dpdk
cd dpdk
git checkout v20.11
```

2. Install dependencies (if needed):

```
yum install cmake gcc libnl3-devel libudev-devel make pkgconfig valgrind-devel
pandoc libibverbs libmlx5 libmnl-devel -y
```

3. Configure DPDK:

```
export RTE_SDK=$PWD
make config T=x86_64-native-linuxapp-gcc
cd build
sed -i 's/(CONFIG_RTE_LIBRTE_MLX5_PMD=)\n\1y/g' .config
sed -i 's/(CONFIG_RTE_LIBRTE_MLX5_VDPA_PMD=)\n\1y/g' .config
```

4. Build DPDK:

```
make -j
```

5. Build the vDPA application:

```
cd $RTE_SDK/examples/vdpa/
make -j
```

Hardware vDPA Configuration

To configure huge pages:

```
mkdir -p /hugepages
mount -t hugetlbfs hugetlbfs /hugepages
echo <more> > /sys/devices/system/node/node0/hugepages/hugepages-
1048576kB/nr_hugepages
echo <more> > /sys/devices/system/node/node1/hugepages/hugepages-
1048576kB/nr_hugepages
```

To configure a vDPA VirtIO interface in an existing VM's xml file (using libvirt):

1. Open the VM's configuration XML for editing:

```
virsh edit <domain name>
```

2. Perform the following:

1. Change the top line to:

```
<domain type='kvm'  
xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
```

2. Assign a memory amount and use 1GB page size for huge pages (size must be the same as that used for the vDPA application), so that the memory configuration looks as follows.

```
<memory unit='KiB'>4194304</memory>  
<currentMemory unit='KiB'>4194304</currentMemory>  
<memoryBacking>  
<hugepages>  
<page size='1048576' unit='KiB'/>  
</hugepages>  
</memoryBacking>
```

3. Assign an amount of CPUs for the VM CPU configuration, so that the `vcpu` and `cputune` configuration looks as follows:

```
<vcpu placement='static'>5</vcpu>  
<cputune>  
<vcpupin vcpu='0' cpuset='14'/>  
<vcpupin vcpu='1' cpuset='16'/>  
<vcpupin vcpu='2' cpuset='18'/>  
<vcpupin vcpu='3' cpuset='20'/>  
<vcpupin vcpu='4' cpuset='22'/>  
</cputune>
```

4. Set the memory access for the CPUs to be shared, so that the `cpu` configuration looks as follows:

```
<cpu mode='custom' match='exact' check='partial'>  
<model fallback='allow'>Skylake-Server-IBRS</model>  
<numa>
```

```
<cell id='0' cpus='0-4' memory='8388608' unit='KiB' memAccess='shared' />
</numa>
</cpu>
```

5. Set the emulator in use to be the one built in [step 2](#), so that the emulator configuration looks as follows:

```
<emulator><path to qemu executable></emulator>
```

6. Add a virtio interface using QEMU command line argument entries, so that the new interface snippet looks as follows:

```
<qemu:commandline>
<qemu:arg value='-chardev' />
<qemu:arg value='socket,id=charnet1,path=/tmp/sock-virtio0' />
<qemu:arg value='-netdev' />
<qemu:arg value='vhost-user,chardev=charnet1,queues=16,id=hostnet1' />
<qemu:arg value='-device' />
<qemu:arg value='virtio-net-
pci,mq=on,vectors=6,netdev=hostnet1,id=net1,mac=e4:11:c6:d3:45:f2,bus=p
page-per-vq=on,rx_queue_size=1024,tx_queue_size=1024' />
</qemu:commandline>
```

Note

In this snippet, the vhostuser socket file path, the amount of queues, the MAC and the PCIe slot of the virtio device can be configured.

Running Hardware vDPA

i Note

Hardware vDPA supports switchdev mode only.

1. Create the ASAP² environment:

1. Create the VFs.
2. Enter switchdev mode.
3. Set up OVS.

2. Run the vDPA application:

```
cd $RTE_SDK/examples/vdpa/build  
./vdpa -w <VF PCI BDF>,class=vdpa --log-level=pmd,info -- -i
```

3. Create a vDPA port via the vDPA application CLI:

```
create /tmp/sock-virtio0 <PCI DEVICE BDF>
```

i Note

The vhostuser socket file path must be the one used when configuring the VM.

4. Start the VM:

```
virsh start <domain name>
```

For further information on the vDPA application, visit the [Vdpa Sample Application DPDK](#) documentation.

Bridge Offload

Note

Bridge offload is supported switchdev mode only.

Note

Bridge offload is supported from kernel version 5.15 onward.

A Linux bridge is an in-kernel software network switch (based on and implementing a subset of IEEE 802.1D standard) used to connect Ethernet segments together in a protocol-independent manner. Packets are forwarded based on L2 Ethernet header addresses.

mlx5 provides the ability to offload bridge dataplane unicast packet forwarding and VLAN management to hardware.

Basic Configuration

1. Initialize the ASAP² environment:
 1. Create the VFs.
 2. Enter switchdev mode.
2. Create a bridge and add mlx5 representors to bridge:

```
ip link add name bridge0 type bridge
```



```
ip link set enp8s0f0_0 master bridge0
```

Configuring VLAN

1. Enable VLAN filtering on the bridge:

```
ip link set bridge0 type bridge vlan_filtering 1
```

2. Configure port VLAN matching (trunk mode). In this configuration, only packets with specified VID are allowed.

```
bridge vlan add dev enp8s0f0_0 vid 2
```

3. Configure port VLAN tagging (access mode). In this configuration, VLAN header is pushed/popped upon reception/transmission on port.

```
bridge vlan add dev enp8s0f0_0 vid 2 pvid untagged
```

VF LAG Support

Bridge supports offloading on bond net device that is fully initialized with mlx5 uplink representors and is in single (shared) FDB LAG mode. Details about initialization of LAG are provided in section "[SR-IOV VF LAG](#)".

To add a bonding net device to bridge:

```
ip link set bond0 master bridge0
```

For further information on interacting with Linux bridge via iproute2 bridge tool, refer to [man 8 bridge](#).

Link Aggregation

Network bonding enables combining two or more network interfaces into a single interface. It increases the network throughput, bandwidth and provides redundancy if one of the interfaces fails.

NVIDIA® BlueField® DPU has an option to configure network bonding on the Arm side in a manner transparent to the host. Under such configuration, the host would only see a single PF.

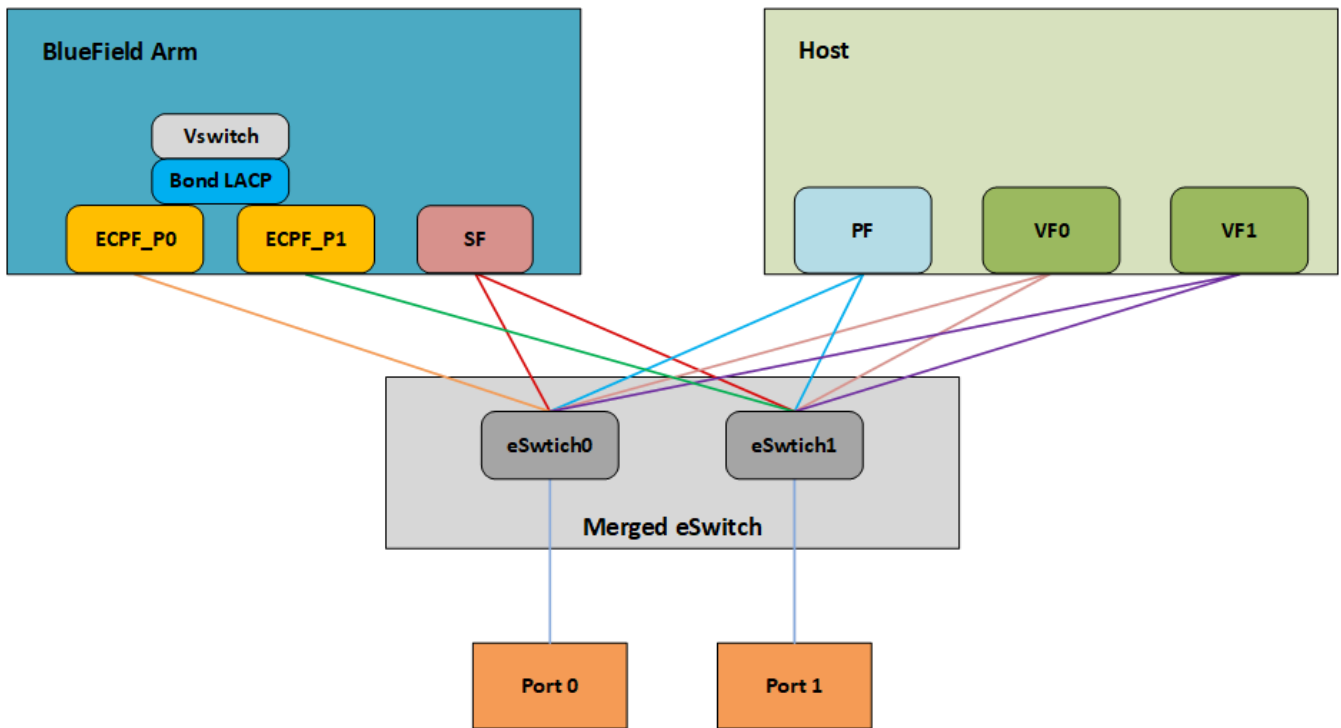
(i) Note

This functionality is supported when the DPU is set in embedded function ownership mode for both ports.

(i) Note

While LAG is being configured (starting with step 2 under section "[LAG Configuration](#)"), traffic cannot pass through the physical ports.

The diagram below describes this configuration:



LAG Modes

Two LAG modes are supported on BlueField:

- Queue Affinity mode
- Hash mode

Queue Affinity Mode

In this mode, packets are distributed according to the QPs.

1. To enable this mode, run:

```
$ mlxconfig -d /dev/mst/<device-name> s LAG_RESOURCE_ALLOCATION=0
```

Example device name: mt41686_pciconf0.

2. Add/edit the following field from /etc/mellanox/mlnx-bf.conf as follows:

```
LAG_HASH_MODE="no"
```

3. Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

Hash Mode

In this mode, packets are distributed to ports according to the hash on packet headers.

Note

For this mode, [prerequisite](#) steps 3 and 4 are not required.

1. To enable this mode, run:

```
$ mlxconfig -d /dev/mst/<device-name> s LAG_RESOURCE_ALLOCATION=1
```

Example device name: `mt41686_pciconf0`.

2. Add/edit the following field from `/etc/mellanox/mlnx-bf.conf` as follows:

```
LAG_HASH_MODE="yes"
```

3. Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

Prerequisites

1. Set the [LAG mode](#) to work with.
2. (Optional) Hide the second PF on the host. Run:

```
$ mlxconfig -d /dev/mst/<device-name> s HIDE_PORT2_PF=True NUM_OF_PF=1
```

Example device name: mt41686_pciconf0.

Note

Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

3. Delete any installed Scalable Functions (SFs) on the Arm side.
4. Stop the driver on the host side. Run:

```
$ systemctl stop openibd
```

5. The uplink interfaces (p0 and p1) on the Arm side must be disconnected from any OVS bridge.

LAG Configuration

1. Create the bond interface. Run:

```
$ ip link add bond0 type bond  
$ ip link set bond0 down  
$ ip link set bond0 type bond miimon 100 mode 4 xmit_hash_policy layer3+4
```

Note

While LAG is being configured (starting with the next step), traffic cannot pass through the physical ports.

2. Subordinate both the uplink representors to the bond interface. Run:

```
$ ip link set p0 down
$ ip link set p1 down
$ ip link set p0 master bond0
$ ip link set p1 master bond0
```

3. Bring the interfaces up. Run:

```
$ ip link set p0 up
$ ip link set p1 up
$ ip link set bond0 up
```

The following is an example of LAG configuration in Ubuntu:

```
# cat /etc/network/interfaces

# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source /etc/network/interfaces.d/*
auto lo
iface lo inet loopback
#p0
auto p0
iface p0 inet manual
bond-master bond1
#
#p1
auto p1
iface p1 inet manual
```

```
bond-master bond1
#bond1
auto bond1
iface bond1 inet static
address 192.168.1.1
netmask 255.255.0.0
mtu 1500
bond-mode 2
bond-slaves p0 p1
bond-miimon 100
pre-up (sleep 2 && ifup p0) &
pre-up (sleep 2 && ifup p1) &
```

As a result, only the first PF of the DPU would be available to the host side for networking and SR-IOV.

Warning

When in shared RQ mode (enabled by default), the uplink interfaces (p0 and p1) must always stay enabled. Disabling them will break LAG support and VF-to-VF communication on same host.

For OVS configuration, the bond interface is the one that needs to be added to the OVS bridge (interfaces p0 and p1 should not be added). The PF representor for the first port (pf0hpf) of the LAG must be added to the OVS bridge. The PF representor for the second port (pf1hpf) would still be visible, but it should not be added to OVS bridge. Consider the following examples:

```
ovs-vsctl add-br bf-lag
ovs-vsctl add-port bf-lag bond0
ovs-vsctl add-port bf-lag pf0hpf
```

Warning

Trying to change bonding configuration in Queue Affinity mode (including bringing the subordinated interface up/down) while the host driver is loaded would cause FW syndrome and failure of the operation. Make sure to unload the host driver before altering DPU bonding configuration to avoid this.

Note

When performing driver reload (openibd restart) or reboot, it is required to remove bond configuration and to reapply the configurations after the driver is fully up. Refer to steps 1-4 of "[Removing LAG Configuration](#)".

Removing LAG Configuration

1. If Queue Affinity mode LAG is configured (i.e., LAG_RESOURCE_ALLOCATION=0):

1. Delete any installed Scalable Functions (SFs) on the Arm side.
2. Stop driver (openibd) on the host side. Run:

```
systemctl stop openibd
```

2. Delete the LAG OVS bridge on the Arm side. Run:

```
ovs-vsctl del-br bf-lag
```

This allows for later restoration of OVS configuration for non-LAG networking.

3. Stop OVS service. Run:


```
systemctl stop openvswitch-switch.service
```

4. Run:

```
ip link set bond0 down  
modprobe -rv bonding
```

As a result, both of the DPU's network interfaces would be available to the host side for networking and SR-IOV.

5. For the host to be able to use the DPU ports, make sure to attach the ECPF and host representor in an OVS bridge on the Arm side. Refer to "[Virtual Switch on DPU](#)" for instructions on how to perform this.

6. Revert from HIDE_PORT2_PF, on the Arm side. Run:

```
mlxconfig -d /dev/mst/<device-name> s HIDE_PORT2_PF=False NUM_OF_PF=2
```

7. Restore default LAG settings in the DPU's firmware. Run:

```
mlxconfig -d /dev/mst/<device-name> s  
LAG_RESOURCE_ALLOCATION=DEVICE_DEFAULT
```

8. Delete the following line from `/etc/mellanox/mlnx-bf.conf` on the Arm side:

```
LAG_HASH_MODE=...
```

9. Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

LAG on Multi-host

Only LAG hash mode is supported with BlueField multi-host.

LAG Multi-host Prerequisites

1. Enable LAG hash mode.
2. Hide the second PF on the host. Run:

```
$ mlxconfig -d /dev/mst/<device-name> s HIDE_PORT2_PF=True NUM_OF_PF=1
```

3. Make sure NVME emulation is disabled:

```
$ mlxconfig -d /dev/mst/<device-name> s NVME_EMULATION_ENABLE=0
```

Example device name: mt41686_pciconf0.

4. The uplink interfaces (p0 and p4) on the Arm side, representing port0 and port1, must be disconnected from any OVS bridge. As a result, only the first PF of the DPU would be available to the host side for networking and SR-IOV.

LAG Configuration on Multi-host

1. Create the bond interface. Run:

```
$ ip link add bond0 type bond  
$ ip link set bond0 down  
$ ip link set bond0 type bond miimon 100 mode 4 xmit_hash_policy layer3+4
```

2. Subordinate both the uplink representors to the bond interface. Run:

```
$ ip link set p0 down  
$ ip link set p4 down  
$ ip link set p0 master bond0  
$ ip link set p4 master bond0
```

3. Bring the interfaces up. Run:

```
$ ip link set p0 up
$ ip link set p4 up
$ ip link set bond0 up
```

4. For OVS configuration, the bond interface is the one that must be added to the OVS bridge (interfaces p0 and p4 should not be added). The PF representor, pf0hpf, must be added to the OVS bridge with the bond interface. The rest of the uplink representors must be added to another OVS bridge along with their PF representors. Consider the following examples:

```
ovs-vsctl add-br br-lag
ovs-vsctl add-port br-lag bond0
ovs-vsctl add-port br-lag pf0hpf
ovs-vsctl add-br br1
ovs-vsctl add-port br1 p1
ovs-vsctl add-port br1 pf1hpf
ovs-vsctl add-br br2
ovs-vsctl add-port br2 p2
ovs-vsctl add-port br2 pf2hpf
ovs-vsctl add-br br3
ovs-vsctl add-port br3 p3
ovs-vsctl add-port br3 pf3hpf
```

Note

When performing driver reload (openibd restart) or reboot, you must remove bond configuration from NetworkManager, and to reapply the configurations after the driver is fully up.

Removing LAG Configuration on Multi-host

Refer to section "[Removing LAG Configuration](#)".

Controlling Host PF and VF Parameters

NVIDIA® BlueField® allows control over some of the networking parameters of the PFs and VFs running on the host side.

Setting Host PF and VF Default MAC Address

From the Arm, users may configure the MAC address of the physical function in the host. After sending the command, users must reload the NVIDIA driver in the host to see the newly configured MAC address. The MAC address goes back to the default value in the FW after system reboot.

Example:

```
$ echo "c4:8a:07:a5:29:59" > /sys/class/net/p0/smart_nic/pf/mac  
$ echo "c4:8a:07:a5:29:61" > /sys/class/net/p0/smart_nic/vf0/mac
```

Setting Host PF and VF Link State

vPort state can be configured to Up, Down, or Follow. For example:

```
$ echo "Follow" > /sys/class/net/p0/smart_nic/pf/vport_state
```

Querying Configuration

To query the current configuration, run:

```
$ cat /sys/class/net/p0/smart_nic/pf/config  
MAC : e4:8b:01:a5:79:5e
```

```
MaxTxRate : 0
State : Follow
```

Zero signifies that the rate limit is unlimited.

Disabling Host Networking PFs

It is possible to not expose ConnectX networking functions to the host for users interested in using storage or VirtIO functions only. When this feature is enabled, the host PF representors (i.e. pf0hpf and pf1hpf) will not be seen on the Arm.

- Without a PF on the host, it is not possible to enable SR-IOV, so VF representors will not be seen on the Arm either
- Without PFs on the host, there can be no SFs on it

To disable host networking PFs, run:

```
mlxconfig -d /dev/mst/mt41686_pciconf0 s NUM_OF_PF=0
```

To reactivate host networking PFs:

- For single-port DPUs, run:


```
mlxconfig -d /dev/mst/mt41686_pciconf0 s NUM_OF_PF=1
```

- For dual-port DPUs, run:

```
mlxconfig -d /dev/mst/mt41686_pciconf0 s NUM_OF_PF=2
```

Note

When there are no networking functions exposed on the host, the reactivation command must be run from the Arm.

 **Note**

Perform a BlueField system reboot for the mlxconfig settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

API References

This section contains the following pages:

- [NVIDIA DOCA Driver APIs](#)
- [NVIDIA DOCA Library APIs](#)

NVIDIA DOCA Driver APIs

The driver APIs for this DOCA version are available [here](#).

NVIDIA DOCA Library APIs

The library APIs for this DOCA version are available [here](#).

Miscellaneous (Runtime)

This section contains the following pages:

- [NVIDIA DOCA Glossary](#)
- [NVIDIA DOCA Crypto Acceleration](#)
- [NVIDIA DOCA Services Fluent Logger](#)
- [NVIDIA DOCA DPU CLI](#)
- [NVIDIA DOCA Emulated Devices](#)
- [NVIDIA BlueField Modes of Operation](#)
- [NVIDIA DOCA with OpenSSL](#)
- [NVIDIA BlueField DPU Scalable Function User Guide](#)
- [NVIDIA TLS Offload Guide](#)
- [NVIDIA DOCA Troubleshooting Guide](#)
- [NVIDIA DOCA Virtual Functions User Guide](#)

NVIDIA DOCA Glossary

Term	Description
ACS	Access control services
ASN	Autonomous system number
ATF	Arm-trusted firmware
BAR	Base address register

Term	Description
BDF address	Bus, device, function address. This is the device's PCIe bus address to uniquely identify the specific device.
BFB	BlueField bootstream
BGP	Border gateway protocol
BMC	Board management controller
BUF	Buffer
BSP	BlueField support package
CBS	Committed burst size
CIR	Committed information rate
CMDQ	Command queue
CPDS	Control pipe dynamic size
CTX	Context
DEK	Data encryption key
DMA	Direct memory access
DOCA	DPU SDK
DPA	Data path accelerator; a n auxiliary processor designed to accelerate data-path operations
DPCP	Direct packet control plane
DPDK	Data plane development kit
DPI	Deep packet inspection
DPIF	Datapath offload interface
DPU	Data processing unit, the third pillar of the data center with CPU and GPU. BlueField is available as a DPU and as a SuperNIC.
DW	Dword
EBS	Excess burst size
ECE	Enhanced connection establishment
ECPF	Embedded CPU physical function

Term	Description
EIR	Excess information rate
eMMC	Embedded multi-media card
ESP	EFI system partition
ESP header	Encapsulating security payload
EU	Execution unit. HW thread; a logical DPA processing unit.
FLR	Function level reset
FIPS	Federal Information Processing Standards
FPGA	Field-programmable gate arrays
FW	Firmware
GDB	GNU debugger
HCA	Host-channel adapter
Host	<p>When referring to "the host" this documentation is referring to the server host. When referring to the Arm based host, the documentation will specifically call out "Arm host".</p> <ul style="list-style-type: none"> • Server host OS refers to the Host Server OS (Linux or Windows) • Arm host refers to the AARCH64 Linux OS which is running on the BlueField Arm Cores
HW	Hardware
hwmon	Hardware monitoring
IB	InfiniBand
ICM	Interface configuration memory
ICV	Integrity check value
IDE	Integrated development environment
IKE	Internet key exchange
IR	Intermediate representation
IRQ	Interrupt request

Term	Description
KPI	Key performance indicator
LSO	Large send offload
LTO	Link-time optimization
MFT	Mellanox firmware tools
MLNX_OFED	Mellanox OpenFabrics Enterprise Distribution
MPU	Message passing interface
MSB	Most significant bit
MSI-X	Message signaled interrupts extended
MSS	Maximum segment size
MSS	Memory subsystem
MST	Mellanox software tools
MTU	Maximum transmission unit
NAT	Network address translation
NIC	Network interface card
NIST	National Institute of Standards and Technology
NS	Namespace
NUMA	Non-uniform memory access
OOB	Out-of-band
OS	Operating system
OVS	Open vSwitch
PBA	Pending bit array
PBS	Peak burst size
PCIe	PCI Express; Peripheral Component Interconnect Express
PF	Physical function
PE	Progress engine

Term	Description
PHC	Physical hardware clock
PIR	Peak information rate
PK	Platform key
PKA	Public key accelerator
POC	Proof of concept
PUD	Process under debug
RD	Route distinguisher
RDMA	Remote direct memory access
RegEx	Regular expression
REQ	Request
RES	Response
RN	Request node RN-F – Fully coherent request node RN-D – IO coherent request node with DVM support RN-I – IO coherent request node
RNG	Random number generator/generation
RoCE	Ethernet and RDMA over converged Ethernet
RQ	Receive queue
RShim	Random shim
RSP	Remote serial protocol
RT	Route target
RTOS	Real-time operating system
RTT	Round-trip time
RX	Receive
RXP	Regular expression processor
SA	Security association
SBSA	Server base system architecture

Term	Description
SDK	Software development kit
SF	Sub-function or scalable function
SFC	Services function chaining
SG	Scatter-gather
SHA	Secure hash algorithm
SNAP	Storage-defined network-accelerated processing
SPDK	Storage performance development kit
SPI	Security parameters index
SQ	Send queue
SR-IOV	Single-root IO virtualization
SuperNIC	a configuration of a DPU that is specific for E-W networking. BlueField has a SuperNIC configuration
SVI	Switch virtual interface
Sync event	Synchronization event
TAI	International Atomic Time
TIR	Transport interface receive
TIS	Transport interface send
TLS	Transport layer security
TX	Transmit
UDS	Unix domain socket
UEFI	Unified extensible firmware interface
UTC	Coordinated Universal Time
VF	Virtual function
VFE	Virtio full emulation
VM	Virtual machine
VMA	NVIDIA® Messaging Accelerator

Term	Description
VNI	<ul style="list-style-type: none"> • Virtual network identifier • VXLAN network identifier
VPI	Virtual protocol interconnect
VRF	Virtual routing and forwarding
VTEP	VXLAN tunnel endpoint
WorkQ or workq	Work queue
WQE	Work queue elements
WR	Write
XLIO	NVIDIA® Accelerated IO

NVIDIA DOCA Crypto Acceleration

NVIDIA® BlueField® DPU incorporates several Public Key Acceleration (PKA) engines to offload the processor of the Arm host, providing high-performance computation of PK algorithms. BlueField's PKA is useful for a wide range of security applications. It can assist with SSL acceleration, or a secure high-performance PK signature generator/checker and certificate related operations.

BlueField's PKA software libraries implement a simple, complete framework for crypto public key infrastructure (PKI) acceleration. It provides direct access to hardware resources from the user space, and makes available a number of arithmetic operations—some basic (e.g., addition and multiplication), and some complex (e.g., modular exponentiation and modular inversion)—and high-level operations such as RSA, Diffie-Hellman, Elliptic Curve Cryptography, and the Federal Digital Signature Algorithm (DSA as documented in FIPS-186) public-private key systems.

Some of the use cases for the BlueField PKA involve integrating OpenSSL software applications with BlueField's PKA hardware. The BlueField PKA dynamic engine for OpenSSL allows applications integrated with OpenSSL (e.g., StrongSwan) to accomplish a variety of security-related goals and to accelerate the cryptographic processing with the BlueField PKA hardware. OpenSSL versions $\geq 1.0.0$, $\leq 1.1.1$, and 3.0.2 are supported.

Note

With CentOS 7.6, only OpenSSL 1.1 (not 1.0) works with PKA engine and keygen. Use `openssl11` with PKA engine and keygen.

The engine supports the following operations:

- RSA
- DH
- DSA
- ECDSA
- ECDH
- Random number generation that is cryptographically secure.

Up to 4096-bit keys for RSA, DH, and DSA operations are supported. Elliptic Curve Cryptography support of (nist) prime curves for 160, 192, 224, 256, 384 and 521 bits.

For example:

To sign a file using BlueField's PKA engine:

```
$ openssl dgst -engine pka -sha256 -sign <privatekey> -out <signature> <filename>
```

To verify the signature, execute:

```
$ openssl dgst -engine pka -sha256 -verify <publickey> -signature <signature>  
<filename>
```

For further details on BlueField PKA, please refer to "PKA Driver Design and Implementation Architecture Document" and/or "PKA Programming Guide". Directions

and instructions on how to integrate the BlueField PKA software libraries are provided in the README files on [our PKA GitHub](#).

NVIDIA DOCA Services Fluent Logger

This guide provides instructions on how to use the logging infrastructure for DOCA services on top of NVIDIA® BlueField® DPU.

Introduction

[Fluent Bit](#) is a fast log collector that collects information from multiple sources and then forwards the data onward using Fluent.

On NVIDIA DPUs, the Fluent Bit logger can be easily configured to collect system data and the logs from the different DOCA services.

Deployment

The deployment is based on a recommended configuration template for the existing [Fluent Bit container](#).

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to [NVIDIA DOCA Container Deployment Guide](#).

The following is an example YAML file for deploying the Fluent Bit pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: fluent-bit
spec:
  hostNetwork: true
  containers:
  - name: fluent-bit
    image: fluent/fluent-bit:latest
    imagePullPolicy: Always
# Example resource definitions
```



```
resources:
requests:
memory: "100Mi"
cpu: "200m"
limits:
memory: "200Mi"
cpu: "300m"
volumeMounts:
- name: varlog
mountPath: /var/log
- name: config-file
mountPath: /fluent-bit/etc/fluent-bit.conf
volumes:
- name: varlog
hostPath:
path: /var/log
- name: config-file
hostPath:
path: /opt/mellanox/doca/services/fluent-bit.conf
type: File
```

As explained in the "[Configuration](#)" section, Fluent Bit uses a configuration file. As such, to ensure that the example YAML file is shared from the DPU to the deployed Fluent Bit container, use the following:

```
path: /opt/mellanox/doca/services/fluent-bit.conf
```

Note

The path below is just an example for where the user can place the `fluent-bit.conf` file. The file could be placed in a different directory on the DPU as long as the YAML file points to the updated location.

Configuration

The Fluent Bit configuration file should have the following sections:

- [SERVICE] – to define the service specifications
- [INPUT] – to define folders to collect logs from (there could be multiple inputs)
- [OUTPUT] – IP and port to stream the data to

Example configuration file:

```
[SERVICE]
Flush 2
Log_Level info
Daemon off
Parsers_File parsers.conf
HTTP_Server On
HTTP_Listen 0.0.0.0
HTTP_Port 2020

[INPUT]
Name tail
Tag kube.*
Path /var/log/containers/*.log
Parser docker
Mem_Buf_Limit 5MB
Skip_Long_Lines On
Refresh_Interval 10

[INPUT]
Name tail
Tag sys.*
Path /var/log/doca*/*.log
Mem_Buf_Limit 5MB
Skip_Long_Lines On
Refresh_Interval 10
```

[OUTPUT]

Name es

Match *

Host 10.20.30.40

Port 9201

Index fluent_bit

Type cpu_metrics

Note

The most important field to pay attention to is Path for the INPUT section. DOCA services report their logs to a unique directory under `/var/log/doca/<service_name>/*.log` per the respective DOCA service. As such, the configuration above defines the `/var/log/doca/*/*.log` input definition.

More information about the full specifications can be found in the [official Fluent Bit manual](#).

Troubleshooting

For container-related troubleshooting, refer to the "Troubleshooting" section in the [NVIDIA DOCA Container Deployment Guide](#).

For general troubleshooting, refer to the [NVIDIA DOCA Troubleshooting Guide](#).

When copying the above YAML file, it is possible that the container infrastructure logs give an error related to RFC 1123". These errors are usually a result of a spacing error in the file, which sometimes occur when copying the file as is from this page. To fix this issue, make sure that only the space character (' ') is used as a spacer in the file and not other whitespace characters that might have been added during the copy operation.

NVIDIA DOCA DPU CLI

This guide provides quick access to a useful set of CLI commands and utilities on the NVIDIA® BlueField® DPU environment.

Introduction

This guide provides a concise guide on useful commands for DOCA deployment and configuration.

The tables in this guide provide two categories of commands:

- General commands for Linux/networking environment
- DOCA/DPU-specific commands

Note

For more information about these commands, such as usage instructions, flag options, arguments and so on, use the `-h` option after the command or use the manual (e.g., `man lspci`).

General Commands

Command	Description
<code>ifconfig</code>	Used to configure kernel-resident network interfaces. It is used at boot time to set up interfaces as necessary. After that, it is usually only needed when debugging or when system tuning is needed. If no arguments are given, <code>ifconfig</code> displays the status of the currently active interfaces. If a single interface argument is given, it displays the status of the given interface only. If a single <code>-a</code> argument is given, it displays the status of all interfaces, even those that are down. Otherwise, it configures an interface.
<code>ethtool <devname></code>	Used to query and control network device driver and hardware settings, particularly for wired Ethernet devices. <devname> is the name of the network device on which <code>ethtool</code> should operate.

Command	Description
	<p>Note This command shows the speed of the network card of the DPU.</p>
lspci	Displays information about PCIe buses in the system and devices connected to them. By default, it shows a brief list of devices.
tcpdump	Dump traffic on a network. Usage: <code>tcpdump -i <interface></code> where <interface> is any port interface (physical/SF rep/VF port rep).
ovs-vsctl	Utility for querying and configuring <code>ovs-vswitchd</code> . The <code>ovs-vsctl</code> program supports the model of a bridge implemented by Open vSwitch in which a single bridge supports ports on multiple VLANs.
mount 10.0.0.10: /vol/myshare/ myshare/	<p>Used for mounting a work directory on the DPU.</p> <p>Note Must be used after creating a new directory named <code>myshare</code> under root (i.e., <code>mkdir /myshare</code>)</p>
scp	Secure copy (remote file copy program). Useful for copying files from BlueField to the host and vice versa.
iperf	Used for server-client connection. Useful to check if the network connection achieves the speed of the network card on the DPU (line rate).

DPU/DOCA Commands

Command	Description
ibdev2netdev	Displays available <code>mlx</code> interfaces
mst	Used to start MST service, to stop it, and for other operations with NVIDIA devices like reset and enabling remote access

Command	Description
<code>cat /etc/mlnx-release</code>	Displays the full BlueField image (bfb) version
<code>cat /etc/os-release</code>	Displays the details of the underlying OS installed on BlueField
<code>ibv_devinfo</code>	Displays the current InfiniBand connected devices and relevant information. Useful for checking current firmware version.
<code>ipmitool power cycle</code>	<p>Power cycle</p> <div style="background-color: #ffffcc; padding: 10px; border: 1px solid #ccc;"> <p>Note</p> <p>Prior to performing a power cycle, make sure to do a <u>graceful shutdown</u>.</p> </div>
<code>echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages</code>	DPDK setup. Allocates hugepages for DPDK environment abstraction layer (EAL).
<code>mlxdevm tool</code>	The <code>mlxdevm</code> tool is found under <code>/opt/mellanox/iproute2/sbin/</code> . With this tool it is possible to create an SF and set its state to active, configure a HW address and set it to trusted, deploy the created SF and print info about it.
<code>/opt/mellanox/iproute2/sbin/mlxdevm port add pci/<pci_address> flavour pcisf pfnm <correspondig_physical_function_number> sfnum <unique_sf_number></code>	Creates an SF in the flavor of the given PF with the given unique SF number. Example: <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc; margin-top: 10px;"> <pre>/opt/mellanox/iproute2/sbin/mlxdevm port add pci/0000:03:00.0 flavour pcisf pfnm 0 sfnum 4</pre> </div>
<code>/opt/mellanox/iproute2/sbin/mlxdevm port show</code>	Displays information about the available SFs
<code>/opt/mellanox/iproute2/sbin/mlxdevm port function set pci/0000:03:00.0/<sf_index> hw_addr <HW_address> trust on state active</code>	Configures SF capabilities such as setting the HW address, making it "trusted", and setting its state to active. <code><sf_index></code> the SF. To obtain this index, you may run <code>mlxdevm port show</code> . Example: <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc; margin-top: 10px;"> <pre>/opt/mellanox/iproute2/sbin/mlxdevm port function set pci/0000:03:00.0/229377 hw_addr 02:25:f2:8d:a2:4c trust on state</pre> </div>

Command	Description
	active
<pre>\$ echo mlx5_core.sf. <next_serial> > /sys/bus/auxiliary/driver s/mlx5_core.sf_cfg/unbi nd \$ echo mlx5_core.sf. <next_serial> > /sys/bus/auxiliary/driver s/mlx5_core.sf/bind</pre>	<p>These two commands deploy the created SF. The first command unbinds the SF from the default driver, while the second command binds the SF to the actual driver. The deployment phase should be done after the capabilities of the SF are configured. The SF is identified by <next_serial> which can be obtained by running the command below.</p>
<pre>ls /sys/bus/auxiliary/devic es/mlx5_core.sf.*</pre>	<p>Displays additional information about the created SFs and their "next serial numbers".</p> <p>For example, if <code>mlx5_core.sf.2</code> exists in the output of the command, then running <code>cat /sys/bus/auxiliary/devices/mlx5_core.sf.2/sfnum</code> would output the <code>sfnum</code> related to <code>mlx5_core.sf.2</code>.</p>
<pre>/opt/mellanox/iproute2/ sbin/mlxdevm port function set pci/<pci_address>/<sf_i ndex> state inactive /opt/mellanox/iproute2/ sbin/mlxdevm port del pci/<pci_address>/<sf_i ndex></pre>	<p>These two commands must be executed to delete a given SF. First, users must set the state of the SF to inactive, and only then should it be deleted.</p>
<pre>/opt/mellanox/iproute2/ sbin/mlxdevm port help</pre>	<p>Displays additional information about operations that can be used on created SF ports</p>
<pre>crictl pods</pre>	<p>Displays currently active K8S pods, and their IDs (it might take up to 20-30 seconds for the pod to start)</p>
<pre>crictl ps</pre>	<p>Displays currently active containers and their IDs</p>
<pre>crictl ps -a</pre>	<p>Displays all containers, including containers that recently finished their execution</p>
<pre>crictl logs <container- id></pre>	<p>Examines the logs of a given container</p>
<pre>crictl exec -it <container-id> /bin/bash</pre>	<p>Attaches a shell to a running container</p>
<pre>journalctl -u kubelet</pre>	<p>Examines the Kubelet logs. Useful when a pod/container fails to</p>

Command	Description
	spawn.
crictl stopp <pod-id>	Stops a running K8S pod
crictl stop <container-id>	Stops a running container
crictl rmi <image-id>	Removes a container image from the local K8S registry

NVIDIA DOCA Emulated Devices

For information on virtio-net emulation, please refer to [NVIDIA BlueField Virtio-net documentation](#).


VirtIO-net Emulated Devices

Virtio-net device emulation enables users to create VirtIO-net emulated PCIe devices in the system where the NVIDIA® BlueField® DPU is connected. This is done by the virtio-net-controller software module present in the DPU. Virtio-net emulated devices allow users to hot plug up to 31 virtio-net PCIe PF Ethernet NIC devices or 504 virtio-net PCIe VF Ethernet NIC devices in the host system where the DPU is plugged in.

DPU software also enables users to create virtio block PCIe PF and SR-IOV PCIe VF devices. This is covered in the *NVIDIA BlueField SNAP and virtio-blk SNAP Documentation*.

VirtIO-net Controller

Virtio-net-controller is a systemd service running on the DPU, with a user interface frontend to communicate with the background service. An SF representor is created for each virtio-net device created on the host. Virtio-net controller only uses an SF number ≥ 1000 . Refer to section "[Scalable Functions](#)" for more information.

 **Warning**

SF representor name is determined by udev rules. The default name is in the format of <prefix><pf_num><sf_num>. For example: en3f0pf0sf1001.

Each virtio-net PF/VF requires a dedicated SF and it should be reserved from mlxconfig (see section "[VirtIO-net PF Device Configuration](#)"). However, since an SF is a shared resource on the system, there may be other application-created SFs as well. In that case, PF_TOTAL_SF must be updated to consider those SFs. Otherwise, virtio-net is not able to create enough configured PF/VF.

Warning

Since the controller provides hardware resources and acknowledges (ACKs) the request from the host's virtio driver, it is mandatory to reboot the host OS first and the DPU second. This also applies to reconfiguring a controller from the DPU (e.g., reconfiguring LAG); unloading the virtio-net driver from guest side is recommended.

SystemD Service

Controller systemd service is enabled by default and runs automatically if VIRTIO_NET_EMULATION_ENABLE is true from mlxconfig.

1. To check controller service status, run:

```
$ systemctl status virtio-net-controller.service
```

2. To reload the service, make sure to unload virtio-net/virtio-pcie drivers on host. Then run:

```
$ systemctl restart virtio-net-controller.service
```

3. To monitor log output of the controller service, run:

```
$ journalctl -u virtio-net-controller -f
```

4. Before reloading MLNX_OFED or changing the driver to legacy mode from the ARP side, the controller service must be stopped first. Run:

```
$ systemctl stop virtio-net-controller.service
```

The controller service has an optional configuration file which allows users to customize several parameters. The configuration file should be defined on the DPU at the following path `/opt/mellanox/mlnx_virtnet/virtnet.conf`.

This file is read every time the controller starts. Dynamic change of `virtnet.conf` is not supported. It is defined as a JSON format configuration file. The currently supported options are:

- `ib_dev_p0` – RDMA device (e.g., `mlx5_0`) used to create SF on port 0. This port is the EMU manager when `is_lag` is 0. Default value is `mlx5_0`.
- `ib_dev_p1` – RDMA device (e.g., `mlx5_1`) used to create SF on port 1. Default value is `mlx5_1`.
- `ib_dev_lag` – RDMA LAG device (e.g., `mlx5_bond_0`) used to create SF on LAG. Default value is `mlx5_bond_0`. This port is EMU manager when `is_lag` is 1. `ib_dev_lag` and `ib_dev_p0/ib_dev_p1` cannot be configured simultaneously.
- `ib_dev_for_static_pf` – the RDMA device (e.g., `mlx5_0`) which the static virtio PF is created on
- `is_lag` – specifies whether LAG is used. Note that if LAG is used, make sure to use the correct IB dev for static PF.
- `static_pf` –
 - `mac_base` – base MAC address for static PFs. MACs are automatically assigned with the following pattern: `pf_mac pf_0`, `pf_mac+1 pf_1`, etc.

 **Warning**


Note that the controller does not validate the MAC address (other than its length). The user must ensure the MAC is valid and unique.

- features – virtio spec-defined feature bits for static PFs. If unsure, leave features out of the JSON file and a default value is automatically assigned.
- vf –
 - mac_base – base MAC address for static PFs. MACs are automatically assigned with the following pattern: pf_mac pf_0, pf_mac+1 pf_1, etc.
 - features – virtio spec-defined feature bits for static VFs. If unsure, leave features out of the JSON file and a default value is automatically assigned.
 - vfs_per_pf – number of VFs to create on each PF. This is mandatory if mac_base is specified.

 **Warning**

This value does not equal VIRTIO_NET_EMULATION_NUM_VF in mlxconfig. $vfs_per_pf \leq VIRTIO_NET_EMULATION_NUM_VF$.

- qp_num – number of QPs for each VF. If not specified, then the QP number assigned is taken from its parent PF.
- recovery – specifies whether recovery is enabled. If unspecified, recovery is enabled by default. To disable it, set recovery to 0.
- sf_pool_percent – determines the initial SF pool size as the percentage of PF_TOTAL_SF of mlxconfig. Valid range: [0, 100]. For instance, if the value is 5, it means an SF pool with 5% of PF_TOTAL_SF is created. 0 means no SF pool is reserved beforehand (default).

 **Warning**

PF_TOTAL_SF is shared by all applications. User must ensure the percent request is guaranteed or else the controller will not be able to reserve the requested SFs resulting in failure.

- `sf_pool_force_destroy` – specifies whether to destroy the SF pool. When set to 1, the controller destroys the SF pool when stopped/restarted (and the SF pool is recreated if `sf_pool_percent` is not 0 when starting), otherwise it does not. Default value is 0.

For example, the following definition has all static PFs using `mlx5_0` (port 0) as the data path device in a non-lag configuration:

```
{
  "ib_dev_p0": "mlx5_0",
  "ib_dev_p1": "mlx5_1",
  "ib_dev_for_static_pf": "mlx5_0",
  "is_lag": 0,
  "recovery": 1,
  "sf_pool_percent": 0,
  "sf_pool_force_destroy": 0,
  "static_pf": {
    "mac_base": "11:22:33:44:55:66",
    "features": "0x230047082b"
  },
  "vf": {
    "mac_base": "CC:48:15:FF:00:00",
    "features": "0x230047082b",
    "vfs_per_pf": 100,
    "qp_num": 4
  }
}
```

The following is an example for LAG configuration:

```
{
```

```
"ib_dev_lag": "mlx5_bond_0",
"ib_dev_for_static_pf": "mlx5_bond_0",
"is_lag": 1,
"recovery": 1,
"sf_pool_percent": 0,
"sf_pool_force_destroy": 0
}
```

User Frontend

To communicate with the service, a user frontend program (virtnet) is installed on the DPU. Run the following command to check its usage:

```
# virtnet -h
usage: virtnet [-h] [-v] {hotplug,unplug,list,query,modify,log} ...

Nvidia virtio-net-controller command line interface v1.0.9

positional arguments:
{hotplug,unplug,list,query,modify,log}
** Use -h for sub-command usage
hotplug hotplug virtnet device
unplug unplug virtnet device
list list all virtnet devices
query query all or individual virtnet device(s)
modify modify virtnet device
log set log level

optional arguments:
-h, --help show this help message and exit
-v, --version show program's version number and exit
```

Note that each positional argument has its own help menu as well. For example:

```
# virtnet log -h
usage: virtnet log [-h] -l {info,err,debug}
optional arguments:
-h, --help show this help message and exit
-l {info,err,debug}, --level {info,err,debug}
log level: info/err/debug
```

To operate a particular device, either the VUID or device index can be used to locate the device. Both attributes can be fetched from command "virtnet list". For example, to modify the MAC of a specific VF, you may run either of the following commands:

```
# virtnet modify -p 0 -v 0 device -m 0C:C4:7A:FF:22:98
```

Or:

```
# virtnet modify -u <VUID-string> device -m 0C:C4:7A:FF:22:98
```

Warning

The following modify options require unbinding the virtio device from virtio-net driver in the guest OS:

- MAC
- MTU
- Features
- Msix_num
- max_queue_size

For example:

- On the guest OS:

```
$ echo "bdf of virtio-dev" > /sys/bus/pci/drivers/virtio-  
pci/unbind
```

- On the Arm side:

```
$ virtnet modify ...
```

- On the guest OS:

```
$ echo "bdf of virtio-dev" > /sys/bus/pci/drivers/virtio-  
pci/bind
```

Controller Recovery

Recovering the control and data planes is possible if communications are interrupted so the original traffic can resume.

Recovery depends on the JSON files stored in `/opt/mellanox/mlnx_virtnet/recovery` where there is a file that corresponds to each device (either PF or VF). The following is an example of the data stored in these files:

```
{  
  "port_ib_dev": "mlx5_0",  
  "pf_id": 0,  
  "function_type": "pf",  
  "bdf_raw": 26624,  
  "device_type": "hotplug",  
  "mac": "0c:c4:7a:ff:22:93",  
  "pf_num": 0,  
  "sf_num": 2000,  
  "mq": 1
```

```
}
```

Warning

These files should not be modified under normal circumstances. They are internal to the controller.

Warning

Controller recovery is enabled by default and does not need user configuration or intervention unless a system reset is needed or BlueField configuration is changed (i.e., any of the mlxconfig options `PCI_SWITCH_EMULATION_NUM_PORT`, `VIRTIO_NET_EMULATION_NUM_VF`, or `VIRTIO_NET_EMULATION_NUM_PF`). To this end, the files under `/opt/mellanox/mlnx_virtnet/recovery` must be deleted.

The first time LAG is configured with a controller, recover files must be cleaned up to ensure the controller does not try to recover devices with the previous IB parent device.

Controller Live Update

Live update minimizes network interface down time by performing online upgrade of the virtio-net controller without necessitating a full restart.

To perform a live update, you must install a newer version of the controller either using the rpm or deb package (depending on the OS distro used). Run:

For Ubuntu/Debian	<code>dpkg --force-all -i virtio-net-controller-x.y.z-</code>
----------------------	---

	1.mlnx.aarch64.deb
For CentOS/RedHat	<pre>rpm -Uvh virtio-net-controller-x.y.z-1.mlnx.aarch64.rpm --force</pre>

It is recommended to use the following command to verify the versions of the controller currently running and the one just installed:

```
virtnet version
```

If the versions that are correct, issue the following command to start the live update process:

```
virtnet update --start  
virtnet update -s
```

Warning

If an error appears regarding the "update" command not being supported, this implies that the controller version you are trying to install is too old. Reinstalling the proper version will resolve this issue.

During the update process, the following command may be used to check the update status:

```
virtnet update status  
virtnet update -t
```

During the update, all existing virtnet commands (e.g., list, query, modify) are still supported. VF creation/deletion works as well.

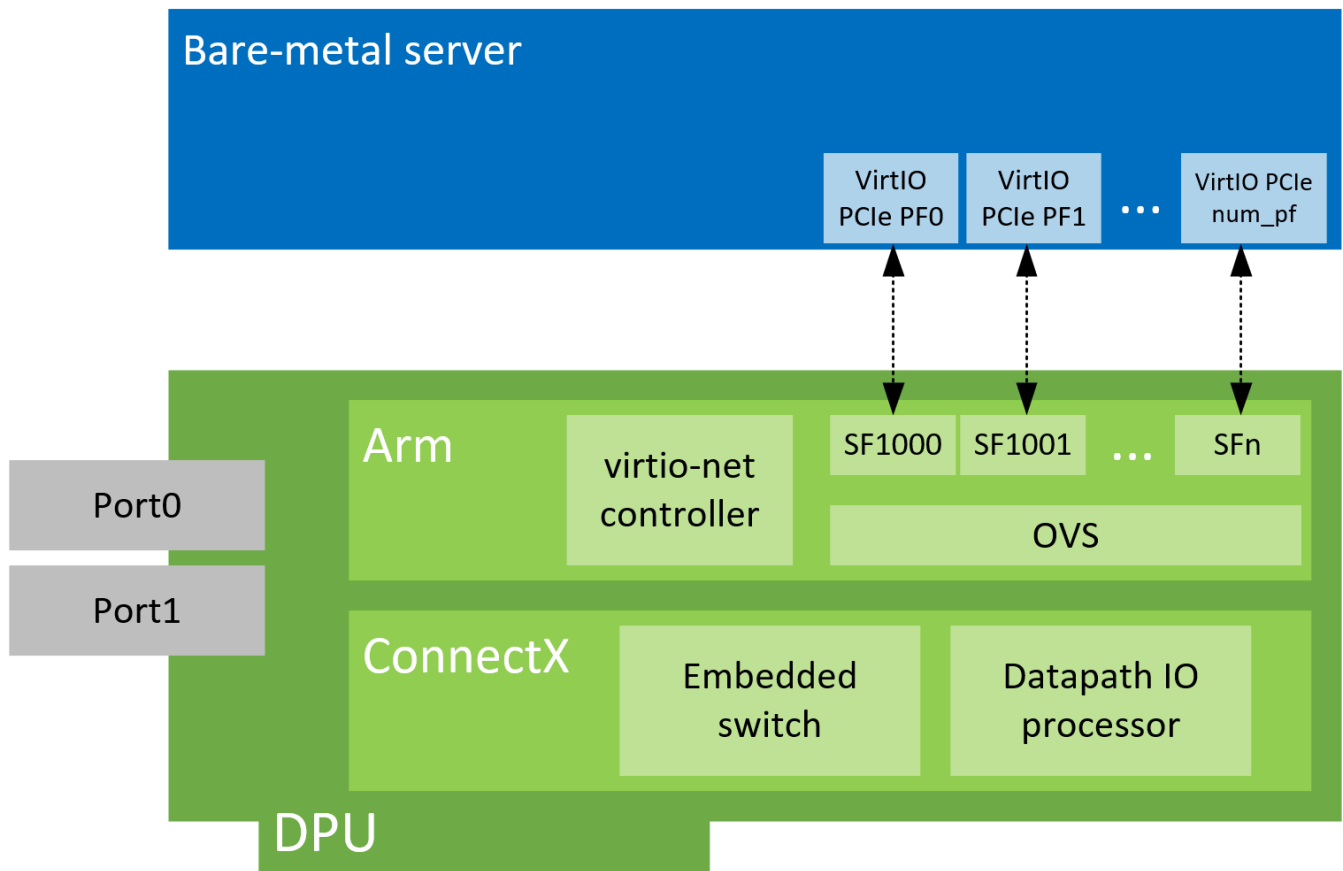
When the update process completes successfully, the command `virtnet update status` will reflect the status accordingly.

Warning

If a device is actively migrating, the existing `virtnet` commands will appear as "migrating" for that specific device so that user can retry later.

VirtIO-net PF Devices

This section covers managing virtio-net PCIe PF devices using virtio-net controller.



VirtIO-net PF Device Configuration

1. Run the following commands on the DPU if it is not already configured to DPU mode:

```
$ mst start
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s INTERNAL_CPU_MODEL=1
```

2. Add the following kernel boot parameters to the Linux boot arguments from the host OS:

```
pci=realloc
```

3. Cold reboot the host system.
4. Apply the following configuration on the DPU:

```
$ mst start
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_ENABLE=0
PER_PF_NUM_SF=1
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s \
PCI_SWITCH_EMULATION_ENABLE=1 \
PCI_SWITCH_EMULATION_NUM_PORT=16 \
VIRTIO_NET_EMULATION_ENABLE=1 \
VIRTIO_NET_EMULATION_NUM_VF=0 \
VIRTIO_NET_EMULATION_NUM_PF=0 \
VIRTIO_NET_EMULATION_NUM_MSIX=10 \
SRIOV_EN=0 \
PF_SF_BAR_SIZE=10 \
PF_TOTAL_SF=64
$ mlxconfig -d /dev/mst/mt41686_pciconf0.1 s \
PF_SF_BAR_SIZE=10 \
PF_TOTAL_SF=64
```

5. Cold reboot the host system a second time.

Creating Modern Hotplug VirtIO-net PF Device


Virtio emulated network PCIe devices are created and destroyed using virtio-net-controller application console. When this application is terminated, all created virtio-net emulated devices are hot unplugged.

1. Create a hotplug virtio-net device. Run:

```
$ virtnet hotplug -i mlx5_0 -f 0x0 -m 0C:C4:7A:FF:22:93 -t 1500 -n 3 -s 1024
```

-  **Warning**

Note that the controller does not validate the MAC address (other than its length). The user must ensure MAC is valid and unique.

-  **Warning**

The maximum number of virtio-net queues is bound by the minimum of the following numbers:

- VIRTIO_NET_EMULATION_NUM_MSIX from the command `mlxconfig -d <mst_dev> q`
- `max_virtq` from the command `virtnet list`

This creates one hotplug virtio-net device with MAC address 0C:C4:7A:FF:22:93, MTU 1500, and 3 virtio queues with a depth of 1024 entries. This device is uniquely identified by its index. This index is used to query and update device attributes. If the device is created successfully, an output appears similar to the following:

```
{
```

```
"bdf": "85:00.0",  
"vuid": "VNETS1D0F0",  
"id": 3,  
"sf_rep_net_device": "en3f0pf0sf2000",  
"mac": "0C:C4:7A:FF:22:93"  
}
```

2. Add the representor port of the device to the OVS bridge and bring it up. Run:

```
$ ovs-vsctl add-port <bridge> en3f0pf0sf2000  
$ ip link set dev en3f0pf0sf2000 up
```

Once steps 1-2 are completed, the virtio-net device should be available from guest OS with the same PCIe bdf.

```
$ lspci | grep -i virtio  
85:00.0 Ethernet controller: Red Hat, Inc. Virtio network device (rev 01)
```

3. To query all the device configurations of virtio-net device that you created, run:

```
$ virtnet query -p 0
```

4. To list all the virtio-net devices, run:

```
$ virtnet list
```

5. To modify device attributes, for example, changing its MAC address, run:

```
$ virtnet modify -p 0 device -m 0C:C4:7A:FF:22:98
```

6. Once usage is complete, to hot-unplug a virtio-net device, run:

```
$ virtnet unplug -p 0
```

Creating Transitional Hotplug VirtIO-net PF Device

A transitional device is a virtio device which supports drivers conforming to virtio specification 1.x and legacy drivers operating under virtio specification 0.95 (i.e., legacy mode) so that servers with old Linux kernels can still utilize virtio-based technology.

1. Run the following command on the DPU:

```
$ mst start
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s \
VIRTIO_NET_EMULATION_PF_PCI_LAYOUT=1 \
VIRTIO_EMULATION_HOTPLUG_TRANS=1
```

2. Add the following parameters to the Linux boot arguments on the guest OS (host OS or VM) side:

```
virtio_pci.force_legacy=1 intel_iommu=off
```

Refer to the [known limitations](#) below.

3. Cold reboot the host system.
4. If virtio_pci is a kernel module rather than built-in from the guest OS, run the following command after both the host and DPU Oses are up:

```
modprobe -rv virtio_pci
modprobe -v virtio_pci force_legacy=1
```

5. To create a transitional hotplug virtio-net device. Run the following command on the DPU (with additional -l/--legacy):

```
$ virtnet hotplug -i mlx5_0 -f 0x0 -m 0C:C4:7A:FF:22:93 -t 1500 -n 3 -s 1024 -l
```

6. Proceed from step 2 of section "[Creating Modern Hotplug VirtIO-net PF Device](#)" for the rest of configuration.

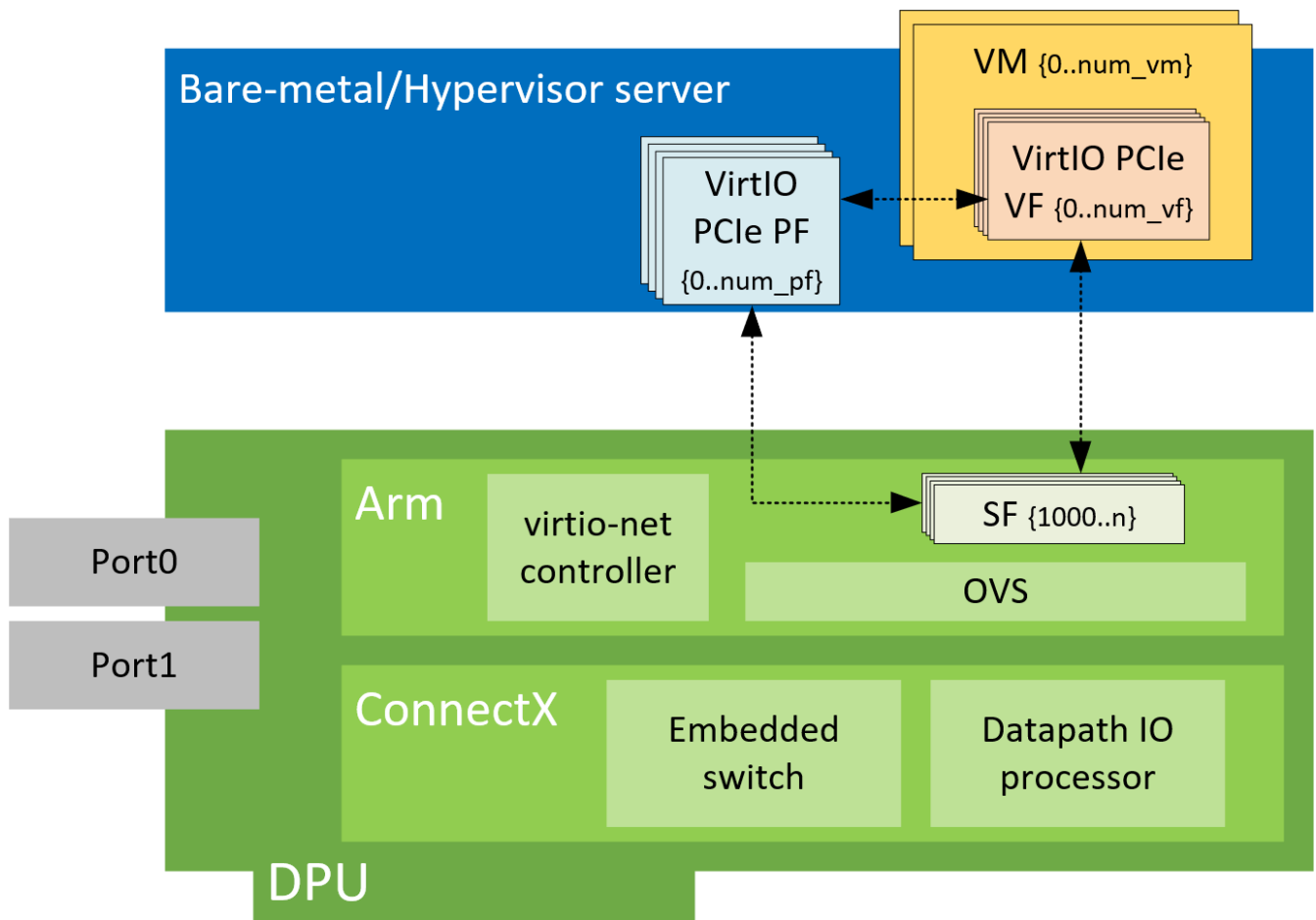
Warning

Known limitations:

- AMD CPU is not supported.
- Only kernel versions 3.10 and above are supported.
intel_iommu=off is not required for kernel 5.1 and above.
- An x86-64 system has only 64K I/O port space which is shared by all peripherals. The virtio transitional device uses I/O BAR. The hotplug device is under one PCIe bridge which is at the emulated PCIe switch downstream port. According to the PCIe specification, the granularity for the bridge I/O window is 4K bytes. If the system cannot satisfy the I/O resource demands by the emulated PCIe switch (depending on the port number of the PCIe switch), the I/O BAR allocation will fail. One hot-plug device requires one emulated PCIe switch port. Each emulated PCIe switch port takes 4K bytes of I/O space if the transitional virtio device is supported. Use `cat /proc/ioports` to check how many I/O port resources are allocated for the host bridge which contains the NIC. The number of supported hotplug transitional virtio device equals: $(\text{allocated I/O port space} - 4k) / 4k$.

Virtio-net SR-IOV VF Devices

This section covers managing virtio-net PCIe SR-IOV VF devices using virtio-net-controller.



Virtio-net SR-IOV VF Device Configuration

Warning

Virtio-net SR-IOV VF is only supported with statically configured PF, hot-plugged PF is not currently supported.

1. On the DPU, make sure virtio-net-controller service is enabled so that it starts automatically. Run:

```
systemctl status virtio-net-controller.service
```


2. On the host, enable SR-IOV. Please refer to [MLNX_OFED documentation](#) under Features Overview and Configuration > Virtualization > Single Root IO Virtualization (SR-IOV) > Setting Up SR-IOV for instructions on how to do that. Make sure the parameters `intel_iommu=on iommu=pt pci=realloc` exist in `grub.conf` file.
3. It is recommended to add `pci=assign-busses` to the boot command line when creating more than 127 VFs. Without this option, the following errors might appear from host and the virtio driver will not probe these devices.

```
pci 0000:84:00.0: [1af4:1041] type 7f class 0xffffffff
pci 0000:84:00.0: unknown header type 7f, ignoring device
```

4. Run the following command on the DPU if it is not already configured to DPU mode:

```
mst start && mlxconfig -d /dev/mst/mt41686_pciconf0 s
INTERNAL_CPU_MODEL=1
```

5. Add the following kernel boot parameters to the Linux boot arguments:

```
intel_iommu=on iommu=pt pci=realloc
```

6. Cold reboot the host system.
7. Apply the following configuration on the DPU in three steps to support up to 125 VFs per PF (500 VFs in total).

Note

The maximum number of VFs 504, so `VIRTIO_NET_EMULATION_NUM_PF * VIRTIO_NET_EMULATION_NUM_VF` must be equal to or less than the max value.

1.

```
$ mst start && mlxconfig -d /dev/mst/mt41686_pciconf0 s  
PF_BAR2_ENABLE=0 PER_PF_NUM_SF=1
```
2.

```
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s \  
PCI_SWITCH_EMULATION_ENABLE=0 \  
PCI_SWITCH_EMULATION_NUM_PORT=0 \  
VIRTIO_NET_EMULATION_ENABLE=1 \  
VIRTIO_NET_EMULATION_NUM_VF=126 \  
VIRTIO_NET_EMULATION_NUM_PF=4 \  
VIRTIO_NET_EMULATION_NUM_MSIX=4 \  
NUM_VF_MSIX=4 \  
SRIOV_EN=1 \  
PF_SF_BAR_SIZE=8 \  
PF_TOTAL_SF=508 \  
NUM_OF_VFS=0
```
3.

```
$ mlxconfig -d /dev/mst/mt41686_pciconf0.1 s PF_TOTAL_SF=1  
PF_SF_BAR_SIZE=8
```

8. Cold reboot the host system.

Creating Virtio-net SR-IOV VF Devices

1. On the host, make sure the static virtio network device presents. Run:

```
# lspci | grep -i virtio  
85:00.3 Network controller: Red Hat, Inc. Virtio network device
```

2. On the host, make sure virtio_pci and virtio_net are loaded. Run:

```
# lsmod | grep virtio
```

The net device should be created:

```
# ethtool -i p7p3
driver: virtio_net
version: 1.0.0
firmware-version:
expansion-rom-version:
bus-info: 0000:85:00.3
supports-statistics: no
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no
```

3. To create SR-IOV VF devices on the host, run:

```
# echo 2 > /sys/bus/pci/drivers/virtio-pci/0000\:85\:00.3/sriov_numvfs
```

Warning

When the number of VFs created is high, SR-IOV enablement may take several minutes.

2 VFs should be created from the host:

```
# lspci | grep -i virt
85:00.3 Network controller: Red Hat, Inc. Virtio network device
85:04.5 Network controller: Red Hat, Inc. Virtio network device
85:04.6 Network controller: Red Hat, Inc. Virtio network device
```

4. From the DPU virtio-net controller, run the following command to get VF information.

```
# virtnet list
```

```
{
  "vf_id": 0,
  "parent_pf_id": 0,
  "function_type": "VF",
  "vuid": "VNETS0D0F2VF1",
  "bdf": "83:00.6",
  "sf_num": 3000,
  "sf_parent_device": "mlx5_0",
  "sf_rep_net_device": "en3f0pf0sf3000",
  "sf_rep_net_ifindex": 19,
  "sf_rdma_device": "mlx5_7",
  "sf_vhca_id": "0x192",
  "msix_config_vector": "0x0",
  "num_msix": 10,
  "max_queues": 4,
  "max_queues_size": 256,
  "net_mac": "5A:94:07:04:F6:1C",
  "net_mtu": 1500
},
```

You may use the pci-bdf to match the PF/VF on the host to the information showing on DPU.

To query all the device configurations of the virtio-net device of that VF, run:

```
$ virtnet query -p 0 -v 0
```

Add the corresponding SF representor to the OVS bridge and bring it up. Run:

```
# ovs-vsctl add-port <bridge> en3f0pf0sf1004
# ip link set dev en3f0pf0sf1004 up
```

Now the VF is functional.

Warning

When port MTU (p0/p1 of the DPU) is changed after the controller is started, you must restart controller service. It is not recommended to use jumbo MTUs because that may lead to performance degradation.

5. To destroy SR-IOV VF devices on the host, run:

```
# echo 0 > /sys/bus/pci/drivers/virtio-pci/0000\:85\:00.3/sriov_numvfs
```

Warning

When the command returns from the host OS, it does not necessarily mean the controller finished its operations. Look at controller log from the DPU and make sure you see a log like below before removing virtio kernel modules or recreate VFs.

```
# virtio-net-controller[3544]: [INFO]
virtnet.c:617:virtnet_device_vfs_unload: PF(0): Unload (4)
VFs finished
```

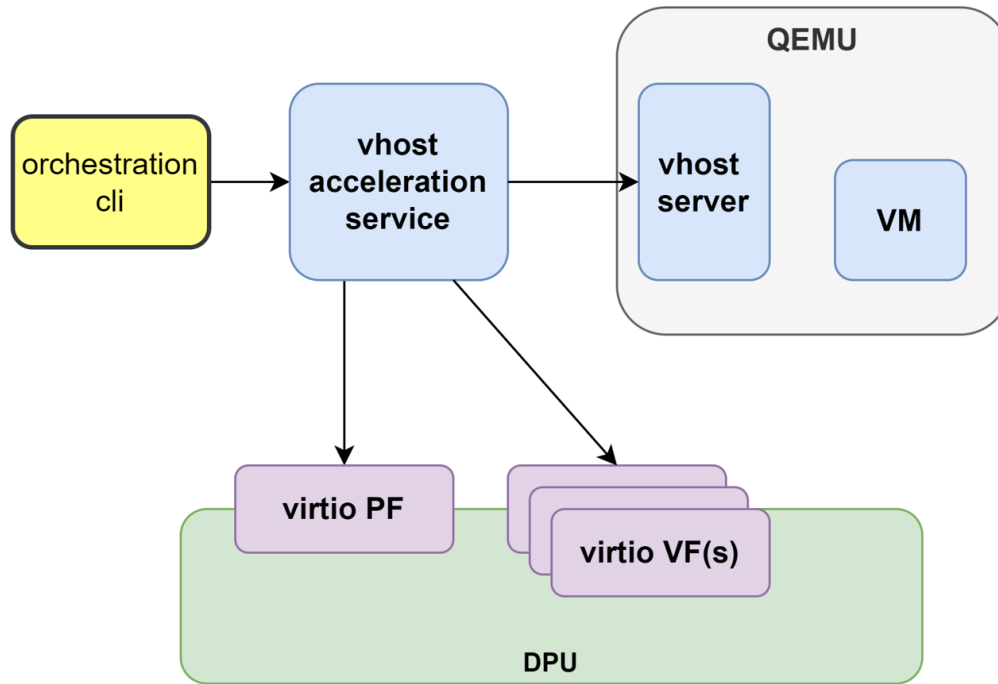
Once VFs are destroyed, created SFs from the DPU side are not destroyed but are saved into the SF pool to be reused later.

Transitional VirtIO-net VF Device Support

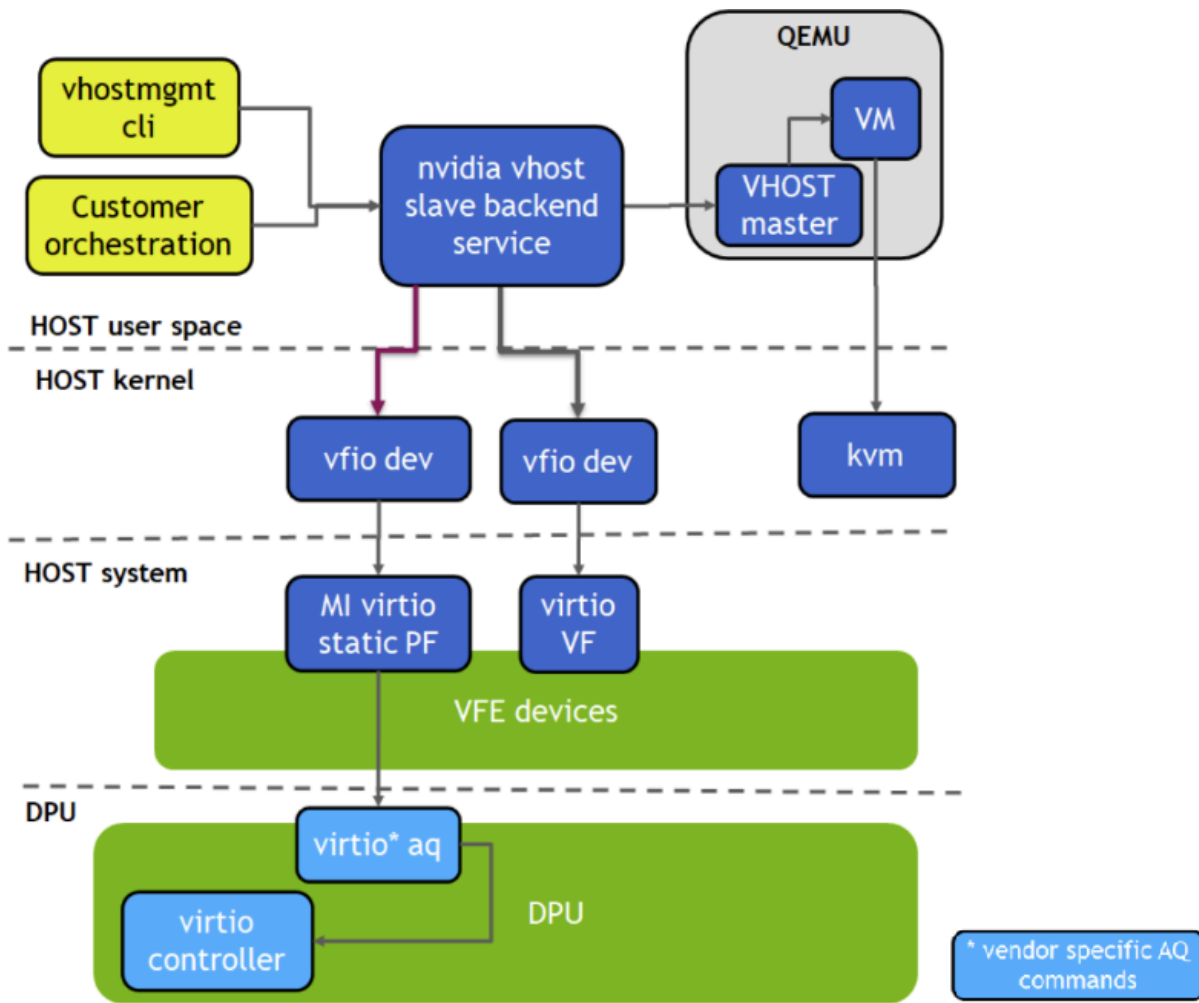
Transitional virtio-net VF devices are not currently supported.

Virtio VF PCIe Devices for vHost Acceleration

Virtio VF PCIe devices can be attached to the guest VM using vhost acceleration software stack. This enables performing live migration of guest VMs.



This section describes the steps to enable VM live migration using virtio VF PCIe devices along with vhost acceleration software.



Prerequisites

- Minimum hypervisor kernel version – Linux kernel 5.7 (for VFIO SR-IOV support)

Install vHost Acceleration Software Stack

Vhost acceleration software stack is built using open-source BSD licensed DPDK.

To install vhost acceleration software:

1. Clone the software source code.

```
git clone https://github.com/Mellanox/dpdk-vhost-vfe
```

Note

Latest release tag is vfe-1.0.

2. Build software:

```
apt-get install libev-dev  
yum install -y numactl-devel libev-devel  
meson build -Dexamples=vdpa  
ninja -C build
```

To install QEMU:

Note

Upstream QEMU later than 8.1 can be used or the following QEMU.

1. Clone QEMU sources.

```
git clone https://github.com/Mellanox/qemu -b stable-8.1-presetup
```

Note

Latest release tag is vfe-0.4.

2. Build QEMU.

```
mkdir bin
cd bin
../configure --target-list=x86_64-softmmu --enable-kvm
make -j24
```

Configure vHost and DPU System

1. Set the DPU nvconfig.

```
mlxconfig -d /dev/mst/mt41686_pciconf0 s \
VIRTIO_NET_EMULATION_ENABLE=1 VIRTIO_NET_EMULATION_NUM_PF=1
VIRTIO_NET_EMULATION_NUM_VF=16 \
VIRTIO_BLK_EMULATION_ENABLE=1 VIRTIO_BLK_EMULATION_NUM_PF=1
VIRTIO_BLK_EMULATION_NUM_VF=16 \
VIRTIO_NET_EMULATION_NUM_MSIX=64
VIRTIO_BLK_EMULATION_NUM_MSIX=64 NUM_VF_MSIX=64
```

2. Cold reboot the system after above configuration.

3. Setup the hypervisor system:

1. Configure hugepages and libvirt VM XML (see [OVS Hardware Offloads Configuration](#) for information on doing that).
2. Add a virtio-net interface and a virtio-blk interface in VM XML.

```
<qemu:commandline>
<qemu:arg value='-chardev'/>
<qemu:arg value='socket,id=char0,path=/tmp/vfe-net0,server=on'/>
<qemu:arg value='-netdev'/>
<qemu:arg value='type=vhost-user,id=vdpa,chardev=char0,queues=4'/>
<qemu:arg value='-device'/>
```

```
<qemu:arg value='virtio-net-  
pci,netdev=vdpa,mac=00:00:00:00:33:00,page-per-  
vq=on,rx_queue_size=1024,tx_queue_size=1024,mq=on'/>  
<qemu:arg value='-chardev'/>  
<qemu:arg value='socket,id=char1,path=/tmp/vfe-blk0,server=on'/>  
<qemu:arg value='-device'/>  
<qemu:arg value='vhost-user-blk-pci,chardev=char1,page-per-  
vq=on,num-queues=4,disable-legacy=on,disable-modern=off'/>  
</qemu:commandline>
```

4. Create block device on the DPU:

```
spdk_rpc.py bdev_null_create Null0 1024 512  
snap_rpc.py controller_virtio_blk_create --pf_id 0 --bdev_type spdk mlx5_0 --  
bdev Null0 --num_queues 1 --admin_q --force_in_order
```

5. On BlueField-3 SNAP:

```
spdk_rpc.py bdev_null_create Null0 1024 512  
snap_rpc.py virtio_blk_controller_create --pf_id 0 --bdev Null0 --num_queues 1 -  
-admin_q --force_in_order
```

Run vHost Acceleration Service

1. Bind the virtio PF devices to vfio-pci driver:

```
modprobe vfio vfio_pci  
echo 1 > /sys/module/vfio_pci/parameters/enable_sriov  
  
echo 0x1af4 0x1041 > /sys/bus/pci/drivers/vfio-pci/new_id  
echo 0x1af4 0x1042 > /sys/bus/pci/drivers/vfio-pci/new_id  
  
echo 0000:af:00.2 > /sys/bus/pci/drivers/vfio-pci/bind
```

```
echo 0000:af:00.3 > /sys/bus/pci/drivers/vfio-pci/bind
```

```
lspci -vvv -s 0000:af:00.3 | grep "Kernel driver"
```

```
Kernel driver in use: vfio-pci
```

```
lspci -vvv -s 0000:af:00.2 | grep "Kernel driver"
```

```
Kernel driver in use: vfio-pci
```

2. Enable SR-IOV and create a VF(s):

```
echo 1 > /sys/bus/pci/devices/0000:af:00.2/sriov_numvfs
```

```
echo 1 > /sys/bus/pci/devices/0000:af:00.3/sriov_numvfs
```

```
lspci | grep Virtio
```

```
af:00.2 Ethernet controller: Red Hat, Inc. Virtio network device
```

```
af:00.3 Non-Volatile memory controller: Red Hat, Inc. Virtio block device
```

```
af:04.5 Ethernet controller: Red Hat, Inc. Virtio network device
```

```
af:05.1 Non-Volatile memory controller: Red Hat, Inc. Virtio block device
```

3. Add a VF representor to the OVS bridge on the DPU:

```
virtnet query -p 0 -v 0 | grep sf_rep_net_device
```

```
"sf_rep_net_device": "en3f0pf0sf3000",
```

```
ovs-vsctl add-port ovsbr1 en3f0pf0sf3000
```

4. Run the vhost acceleration software service:

```
cd dpdk-vhost-vfe
```

```
sudo ./build/app/dpdk-vfe-vdpa -a 0000:00:00.0 --log-level=.,8 --vfio-vf-
```

```
token=cdc786f0-59d4-41d9-b554-fed36ff5e89f -- --client
```

5. Provision the virtio-net PF and VF.

```
cd dpdk-vhost-vfe
```

```
python ./app/vfe-vdpa/vhostmgmt mgmtpf -a 0000:af:00.2
# Wait on virtio-net-controller finishing handle PF FLR

# On DPU, change VF MAC address or other device options
virtnet modify -p 0 -v 0 device -m 00:00:00:00:33:00
python ./app/vfe-vdpa/vhostmgmt vf -a 0000:af:04.5 -v /tmp/vfe-net0
```

6. Provision the virtio-blk PF and VF.

```
cd dpdk-vhost-vfe

python ./app/vfe-vdpa/vhostmgmt mgmtpf -a 0000:af:00.3
# Wait on SNAP controller to finish handling PF FLR

# On DPU, the user must create a VF device controller before adding the VF
device to the
# vhostmgmt upon pf or vf device delete from vhostmgmt, or vhostmgmt
restart:
# For BlueField-3, the VF controller is automatically recreated
# For BlueField-2, the VF controller must be manually recreated
# Use snap_rpc.py controller_list to check for controller existence and create
controller if it's not there
snap_rpc.py controller_virtio_blk_create mlx5_0 --pf_id 0 --vf_id 0 --bdev_type
spdk --bdev Null0 --force_in_order
python ./app/vfe-vdpa/vhostmgmt vf -a 0000:af:05.1 -v /tmp/vfe-blk0
```

Warning

If the SR-IOV is disabled and reenabled, the user must re-provision the VFs.

Start the VM

```
virsh start <domain-name>
```

Simple Live Migration

Prepare two identical hosts and perform the provisioning of the virtio device to DPDK on both.

Boot the VM on one server:

```
virsh migrate --verbose --live --persistent gen-l-vrt-440-162-CentOS-7.4  
qemu+ssh://gen-l-vrt-439/system --unsafe
```

Remove Device

When finished with using the virtio device, use following commands to remove them from DPDK:

```
python ./app/vfe-vdpa/vhostmgmt vf -r 0000:af:04.5  
python ./app/vfe-vdpa/vhostmgmt mgmtpf -r 0000:af:00.2  
  
python ./app/vfe-vdpa/vhostmgmt vf -r 0000:af:05.1  
python ./app/vfe-vdpa/vhostmgmt mgmtpf -r 0000:af:00.3
```

NVIDIA BlueField Modes of Operation

This document describes the modes of operation available for NVIDIA® BlueField® DPU.

Introduction

The NVIDIA® BlueField® DPU has several modes of operation:

- [DPU mode](#), or embedded function (ECPF) ownership, where the embedded Arm system controls the NIC resources and data path
- [Zero-trust mode](#) which is an extension of the ECPF ownership with additional restrictions on the host side
- [NIC mode](#) where the DPU behaves exactly like an adapter card from the perspective of the external host

Note

The default mode of operation for BlueField DPU is DPU mode

The default mode of operation for BlueField SuperNIC is NIC mode

DPU Mode

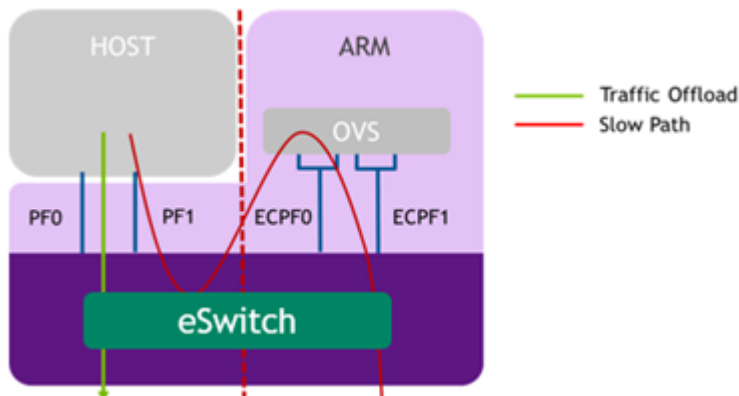
This mode, known also as embedded CPU function ownership (ECPF) mode, is the default mode for BlueField DPU.

In DPU mode, the NIC resources and functionality are owned and controlled by the embedded Arm subsystem. All network communication to the host flows through a virtual switch control plane hosted on the Arm cores, and only then proceeds to the host. While working in this mode, the DPU is the trusted function managed by the data center and host administrator—to load network drivers, reset an interface, bring an interface up and down, update the firmware, and change the mode of operation on the DPU device.

A network function is still exposed to the host, but it has limited privileges. In particular:

1. The driver on the host side can only be loaded after the driver on the DPU has loaded and completed NIC configuration.

2. All ICM (Interface Configuration Memory) is allocated by the ECPF and resides in the DPU's memory.
3. The ECPF controls and configures the NIC embedded switch which means that traffic to and from the host (DPU) interface always lands on the Arm side.



When the server and DPU are initiated, the networking to the host is blocked until the virtual switch on the DPU is loaded. Once it is loaded, traffic to the host is allowed by default.

There are two ways to pass traffic to the host interface: Either using representors to forward traffic to the host (every packet to/from the host would be handled also by the network interface on the embedded Arm side) or push rules to the embedded switch which allows and offloads this traffic.

In DPU mode, OpenSM must be run from the DPU side (not the host side). Also, management tools (e.g., sminfo, ibdev2netdev, ibnetdiscover) can only be run from the DPU side (not from the host side).

Zero-trust Mode

Zero-trust mode is a specialization of DPU mode which implements an additional layer of security where the host system administrator is prevented from accessing the DPU from the host. Once zero-trust mode is enabled, the data center administrator should control the DPU entirely through the Arm cores and/or BMC connection instead of through the host.

For security and isolation purposes, it is possible to restrict the host from performing operations that can compromise the DPU. The following operations can be restricted individually when changing the DPU host to zero-trust mode:

- Port ownership – the host cannot assign itself as port owner
- Hardware counters – the host does not have access to hardware counters
- Tracer functionality is blocked
- RShim interface is blocked
- Firmware flash is restricted

Enabling Zero-trust Mode

To enable host restriction:

1. Start the MST service.
2. Set zero-trust mode. From the Arm side, run:

```
$ sudo mlxprivhost -d /dev/mst/<device> r --disable_rshim --disable_tracer --  
disable_counter_rd --disable_port_owner
```

Note

If any `--disable_*` flags are used, users must perform BlueField system-level reset as explained in the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page.

Disabling Zero-trust Mode

To disable host restriction, set the mode to privileged. Run:

```
$ sudo mlxprivhost -d /dev/mst/<device> p
```


The configuration takes effect immediately.

Note

If host restriction has been applied using any `--disable_*` flags, users must perform BlueField system-level reset as explained in the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page.

NIC Mode

In this mode, the DPU behaves exactly like an adapter card from the perspective of the external host.

Note

The following instructions presume the DPU to operate in DPU mode. If the DPU is operating in zero-trust mode, please [return to DPU mode](#) before continuing.

Note

The following notes are relevant for updating the BFB Bundle in NIC mode:

- During BFB Bundle installation, Linux is expected to boot to upgrade NIC firmware and BMC software
- During the BFB Bundle installation, it is expected for the mlx5 driver to error messages on the x86 host. These prints may be

ignored as they are resolved by a mandatory, post-installation power cycle.

- It is mandatory to power cycle the host after the installation is complete for the changes to take effect
- As Linux is booting during BFB Bundle installation, it is expected for the mlx5 core driver to timeout on the BlueField Arm

NIC Mode for BlueField-3

Note

When BlueField-3 is configured to operate in NIC mode, Arm OS will not boot.

NIC mode for BlueField-3 saves power, improves device performance, and improves the host memory footprint.

Configuring NIC Mode on BlueField-3 from Linux

Enabling NIC Mode on BlueField-3 from Linux

Before moving to NIC mode, make sure you are operating in DPU mode by running:

```
host/dpu> sudo mlxconfig -d /dev/mst/mt41692_pciconf0 -e q
```

The output should have INTERNAL_CPU_MODEL= EMBEDDED_CPU(1) and EXP_ROM_UEFI_ARM_ENABLE = True (1) (default).

To enable NIC mode from DPU mode:

1. Run the following on the host or Arm:

```
host/dpu> sudo mlxconfig -d /dev/mst/mt41692_pciconf0 s  
INTERNAL_CPU_OFFLOAD_ENGINE=1
```

2. Perform a BlueField system-level reset, for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

Disabling NIC Mode on BlueField-3 from Linux

To return to DPU mode from NIC mode:

1. Run the following on the host:

```
host> sudo mlxconfig -d /dev/mst/mt41692_pciconf0 s  
INTERNAL_CPU_OFFLOAD_ENGINE=0
```

2. Perform a BlueField system-level reset for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

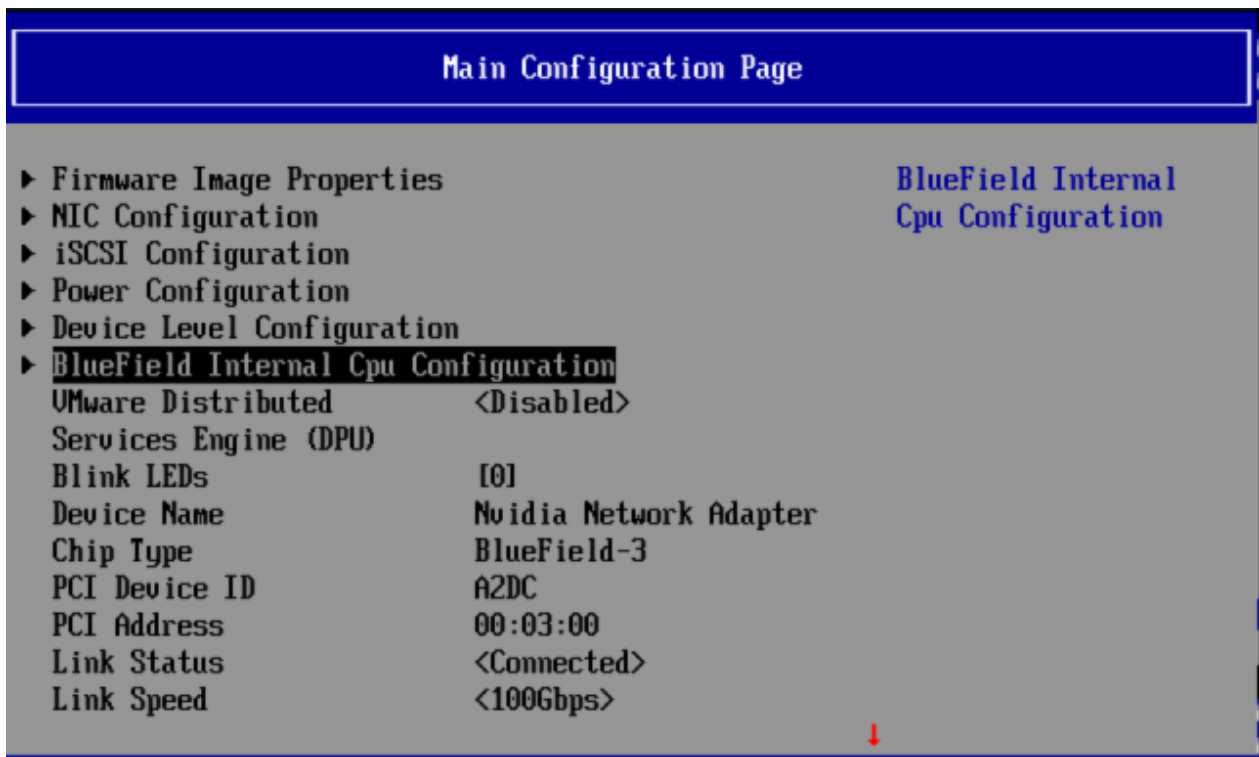
Configuring NIC Mode on BlueField-3 from Host BIOS HII UEFI Menu

Info

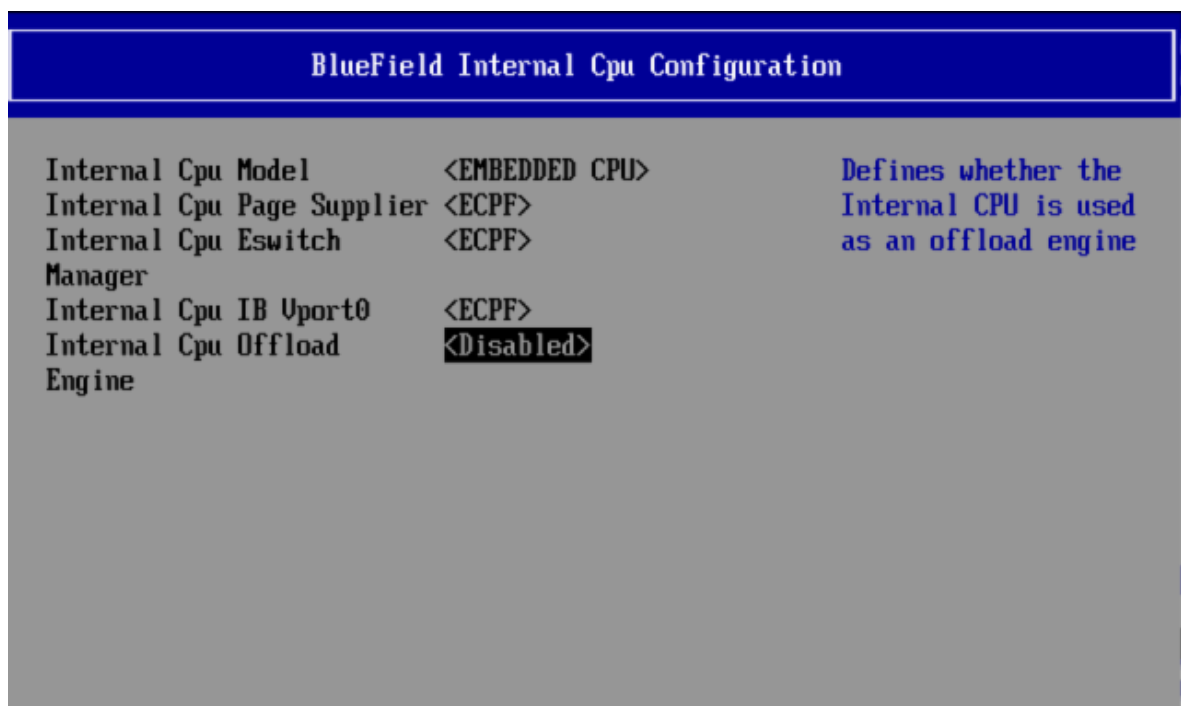
The screenshots in this section are examples only and may vary depending on the vendor of your specific host.

1. Select the network device that presents the uplink (i.e., select the device with the uplink MAC address).

2. Select "BlueField Internal Cpu Configuration".



- To enable NIC mode, set "Internal Cpu Offload Engine" to "Disabled".
- To switch back to DPU mode, set "Internal Cpu Offload Engine" to "Enabled".



Configuring NIC Mode on BlueField-3 from Arm UEFI

1. Access the Arm UEFI menu by pressing the Esc button twice.
2. Select "Device Manager".
3. Select "System Configuration".
4. Select "BlueField Modes".
5. Set the "NIC Mode" field to NicMode to enable NIC mode.

```
Internal CPU Model      <Embedded>
Host Privilege Level   <Privileged>
NIC Mode                <NicMode>
                        Enable/Disable NIC
                        Mode. Any change to
                        this value requires
                        powercycling the
                        system.
                        000000000000000000L:
                        0 DpuMode           0
                        0 NicMode          0
                        0 Unavailable       0
                        000000000000000000L
```

Info

Configuring Unavailable is inapplicable.

6. Exit "BlueField Modes" and "System Configuration" and make sure to save the settings. Exit the UEFI setup using the 'reset' option. The configuration is not yet applied and the DPU is expected to boot regularly, still in DPU Mode.
7. Perform a BlueField system-level reset, to change to NIC Mode. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

Configuring NIC Mode on BlueField-3 Using Redfish

Run the following from the BlueField BMC:

1. Get the current BIOS attributes:

```
sudo curl -k -u root:'<password>' -H 'content-type: application/json' -X GET  
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Bios/
```

2. Change BlueField mode from DpuMode to NicMode:

```
curl -k -u root:'<password>' -H 'content-type: application/json' -d '{ "Attributes":  
{ "NicMode": "NicMode" } }' -X PATCH  
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Bios/Settings
```

Info

To revert back to DPU mode, run:

```
curl -k -u root:'<password>' -H 'content-type:  
application/json' -d '{ "Attributes": { "NicMode":  
"DpuMode" } }' -X PATCH  
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Bios/Settings
```

3. Verify that the BMC has registered the new settings:

```
curl -k -u root:'<password>' -H 'content-type: application/json' -X GET  
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Bios/Settings
```

4. Issue a software reset then power cycle the host for the change to take effect.

5. Verify the mode is changed:

```
curl -k -u root:'<password>' -H 'content-type: application/json' -X GET
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Oem/Nvidia
```

Note

To retrieve the mode via BIOS attributes, another BlueField software reset is required before running the command:

```
curl -k -u root:'<password>' -H 'content-type: application/json' -X
GET https://<bmc_ip>/redfish/v1/Systems/Bluefield/Bios
```

Updating Firmware Components in BlueField-3 NIC Mode

Once in NIC mode, updating ATF and UEFI can be done using the standard *.bfb image:

```
# bfb-install --bfb <BlueField-BSP>.bfb --rshim rshim0
```

NIC Mode for BlueField-2

In this mode, the ECPFs on the Arm side are not functional but the user is still able to access the Arm system and update `mlxconfig` options.

Note

When NIC mode is enabled, the drivers and services on the Arm are no longer functional.

Configuring NIC Mode on BlueField-2 from Linux

Enabling NIC Mode on BlueField-2 from Linux

To enable NIC mode from DPU mode:

1. Run the following from the x86 host side:

```
$ mst start
$ mlxconfig -d /dev/mst/<device> s \
INTERNAL_CPU_PAGE_SUPPLIER=1 \
INTERNAL_CPU_ESWITCH_MANAGER=1 \
INTERNAL_CPU_IB_VPORT0=1 \
INTERNAL_CPU_OFFLOAD_ENGINE=1
```

Note

To restrict RShim PF (optional), make sure to configure INTERNAL_CPU_RSHIM=1 as part of the mlxconfig command.

2. Perform BlueField system-level reset to load the new configuration .

Info

Refer to the troubleshooting section of the guide for a step-by-step procedure.

Note

Multi-host is not supported when the DPU is operating in NIC mode.

Note

To obtain firmware BINs for BlueField-2 devices, please refer to the [BlueField-2 firmware download page](#).

Disabling NIC Mode on BlueField-2 from Linux

To change from NIC mode back to DPU mode:

1. Install and start the RShim driver on the host.
2. Disable NIC mode. Run:

```
$ mst start
$ mlxconfig -d /dev/mst/<device> s \
INTERNAL_CPU_PAGE_SUPPLIER=0 \
INTERNAL_CPU_ESWITCH_MANAGER=0 \
INTERNAL_CPU_IB_VPORT0=0 \
INTERNAL_CPU_OFFLOAD_ENGINE=0
```

Note

If INTERNAL_CPU_RSHIM=1, then make sure to configure INTERNAL_CPU_RSHIM=0 as part of the mlxconfig command.

3. Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

Configuring NIC Mode on BlueField-2 from Arm UEFI

Follow the same instructions in section "[Configuring NIC Mode on BlueField-3 from Arm UEFI](#)".

Configuring NIC Mode on BlueField-2 Using Redfish

Follow the same instructions in section "[Configuring NIC Mode on BlueField-3 Using Redfish](#)".

NVIDIA DOCA with OpenSSL

This guide provides instructions on using DOCA SHA for OpenSSL implementations.

Introduction

The `doca_sha_offload_engine` is an OpenSSL dynamic engine with the ability of offloading SHA calculation. It can offload the OpenSSL one-shot SHA-1, SHA-256, and SHA-512. It supports synchronous mode and asynchronous mode by leveraging the OpenSSL `async_jobs` library. For more information on the `async_jobs` library, please refer to [official OpenSSL documentation](#).

This engine is based on the `doca_sha` library and the OpenSSL dynamic engine interface API. For more information on the OpenSSL dynamic engine, please refer to [official OpenSSL documentation](#).

This engine can be called by an OpenSSL application through the OpenSSL high-level algorithm call interface, `EVP_Digest`. For more information on the `EVP_Digest`, please refer to

[official OpenSSL documentation.](#)

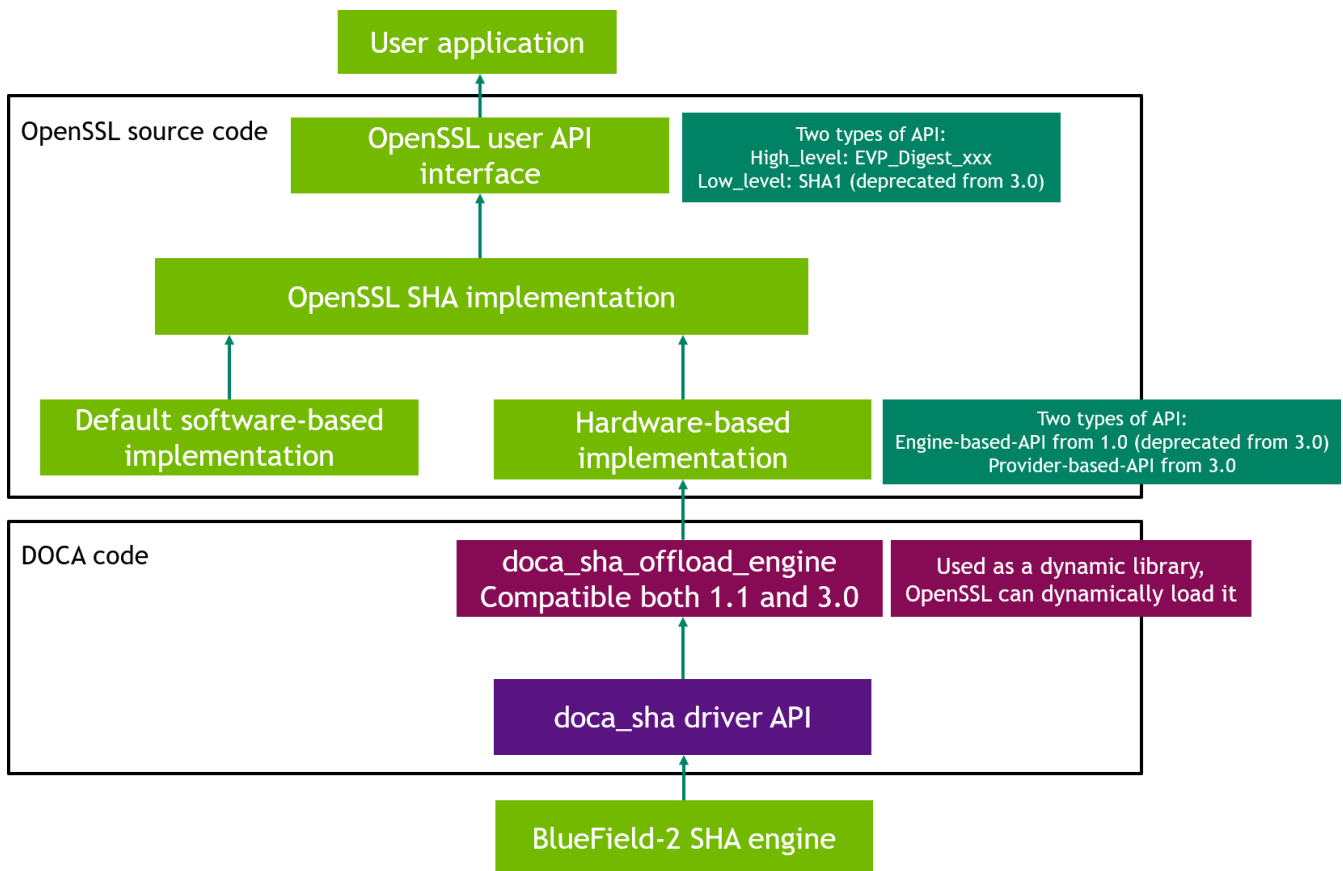
Prerequisites

- Hardware-based `doca_sha` engine which can be verified by calling `doca_sha_get_hardware_supported()`
- Installed OpenSSL version $\geq 1.1.1$

Architecture

The following diagram shows the software hierarchy of `doca_sha_offload_engine` and its location in the whole DOCA repository.

From the perspective of OpenSSL, this engine is an instantiation of the OpenSSL dynamic engine interface API by leveraging the `doca_sha` library.



Capabilities and Limitations

- Only one-shot OpenSSL SHA is supported

- The maximum message length $\leq 2\text{GB}$, the same as `doca_sha` library

OpenSSL Command Line Verification

Verify that the engine can be loaded:

```
$ openssl engine dynamic -pre NO_VCHECK:1 -pre
SO_PATH:${DOCA_DIR}/infrastructure/doca_sha_offload_engine/libdoca_sha_offload_e
-pre LOAD -vv -t -c
(dynamic) Dynamic engine loading support
[Success]:
SO_PATH:${DOCA_DIR}/infrastructure/doca_sha_offload_engine/libdoca_sha_offload_e
[Success]: LOAD
Loaded: (doca_sha_offload_engine) Openssl SHA offloading engine based on
doca_sha
[SHA1, SHA256, SHA512]
[ available ]
set_pci_addr: set the pci address of the doca_sha_engine
(input flags): STRING
```

- For SHA-1:

```
$ echo "hello world" | openssl dgst -sha1 -engine
{DOCA_DIR}/infrastructure/doca_sha_offload_engine/libdoca_sha_offload_engine
-engine_impl
```

- For SHA-256:

```
$ echo "hello world" | openssl dgst -sha256 -engine
{DOCA_DIR}/infrastructure/doca_sha_offload_engine/libdoca_sha_offload_engine
-engine_impl
```

- For SHA-512:

```
$ echo "hello world" | openssl dgst -sha512 -engine  
{DOCA_DIR}/infrastructure/doca_sha_offload_engine/libdoca_sha_offload_engine  
-engine_impl
```

OpenSSL Throughput Test

openssl-speed is the OpenSSL throughput benchmark tool. For more information, consult [official OpenSSL documentation](#). doca_sha_offload_engine throughput can also be measured using openssl-speed.

- SHA-1, each job 10000 bytes, using engine:

```
$ openssl speed -evp sha1 -bytes 10000 -elapsed --engine  
{DOCA_DIR}/infrastructure/doca_sha_offload_engine/libdoca_sha_offload_engine
```

- SHA-256, each job 10000 bytes, using engine, async_jobs=256:

```
$ openssl speed -evp sha256 -bytes 10000 -elapsed --engine  
{DOCA_DIR}/infrastructure/doca_sha_offload_engine/libdoca_sha_offload_engine  
-async_jobs 256
```

- SHA-512, each job 10000 bytes, using engine, async_jobs=256, threads=8:

```
$ openssl speed -evp sha512 -bytes 10000 -elapsed --engine  
{DOCA_DIR}/infrastructure/doca_sha_offload_engine/libdoca_sha_offload_engine  
-async_jobs 256 -multi 8
```

Using DOCA SHA Offload Engine in OpenSSL Application

More information on the dynamic engine usage can be found in the [official OpenSSL documentation](#).

1. To load the doca_sha_offload_engine (optionally, set engine PCIe address):

```
ENGINE *e;
```

```
const char *doca_engine_path =
"${DOCA_DIR}/infrastructure/doca_sha_offload_engine/libdoca_sha_offload_engine.so";
const char *default_doca_pci_addr = "03:00.0";
ENGINE_load_dynamic();
e = ENGINE_by_id(doca_engine_path);
ENGINE_ctrl_cmd_string(e, "set_pci_addr", doca_engine_pci_addr, 0);
ENGINE_init(e);
ENGINE_set_default_digests(e);
```

2. To perform SHA calculation by calling the OpenSSL high-level function EVP_XXX:

```
const EVP_MD *evp_md = EVP_sha1();
EVP_MD_CTX *mdctx = EVP_MD_CTX_create();
EVP_DigestInit_ex(mdctx, evp_md, e);
EVP_DigestUpdate(mdctx, msg, msg_len);
EVP_DigestFinal_ex(mdctx, digest, digest_len);
EVP_MD_CTX_destroy(mdctx);
```

3. To unload the engine:

```
ENGINE_unregister_digests(e);
ENGINE_finish(e);
ENGINE_free(e);
```

NVIDIA BlueField DPU Scalable Function User Guide

This document provides an overview and configuration of scalable functions (sub-functions, or SFs) for NVIDIA® BlueField® DPU.

Introduction

Scalable functions (SFs), or sub-functions, are very similar to virtual functions (VFs) which are part of a Single Root I/O Virtualization (SR-IOV) solution. I/O virtualization is one of the key features used in data centers today. It improves the performance of enterprise servers by giving virtual machines direct access to hardware I/O devices. The SR-IOV specification allows one PCI Express (PCIe) device to present itself to the host as multiple distinct "virtual" devices. This is done with a new PCIe capability structure added to a traditional PCIe function (i.e., a physical function or PF).

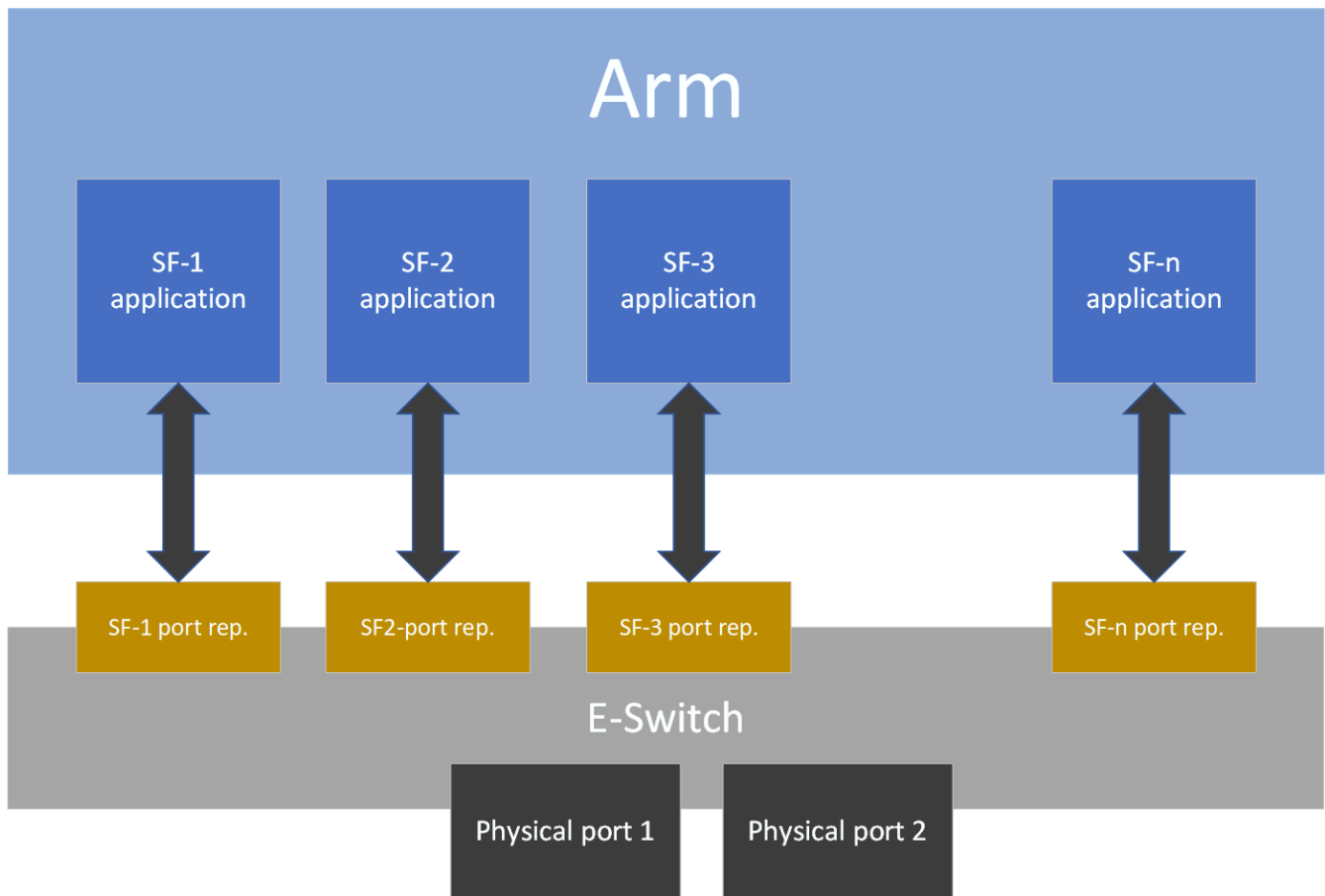
The PF provides control over the creation and allocation of new VFs. VFs share the device's underlying hardware and PCIe. A key feature of the SR-IOV specification is that VFs are very lightweight so that many of them can be implemented in a single device.

To utilize the capabilities of VF in the BlueField, SFs are used. SFs allow support for a larger number of functions than VFs, and more importantly, they allow running multiple services concurrently on the DPU.

An SF is a lightweight function which has a parent PCIe function on which it is deployed. The SF, therefore, has access to the capabilities and resources of its parent PCIe function and has its own function capabilities and its own resources. This means that an SF would also have its own dedicated queues (i.e., txq, rxq).

SFs co-exist with PCIe SR-IOV virtual functions (on the host) but also do not require enabling PCIe SR-IOV.

SFs support E-Switch representation offload like existing PF and VF representors. An SF shares PCIe-level resources with other SFs and/or with its parent PCIe function.



Prerequisites

Refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField related software.

- Make sure your firmware version is 20.30.1004 or higher
- To enable SF support on the device, change the PCIe address for each port:

```
$ mlxconfig -d 0000:03:00.0 s PF_BAR2_ENABLE=0 PER_PF_NUM_SF=1
PF_TOTAL_SF=236
PF_SF_BAR_SIZE=10
```

PF_BAR2_ENABLE: if this config is set, then all PFs and ECPFs have the same number of SFs. This should be off (deprecated).

If set. PF_TOTAL_SF and PF_SF_BAR_SIZE won't work.

PER_PF_NUM_SF: If this config is set, each PF and ECPF configure/control its own number of SFs.

THE ABOVE TWO CONFIGS AFFECTS BOTH BF AND HOST, TREAT WITH CARE!
Also, only one of them can be set. It is INVALID to set them both

PF_TOTAL_SF: maximum number of SFs we wish to configure for the given PF/ECPF.

PF_SF_BAR_SIZE: size of each SF at the BAR2. The size is in powers of 2 in KB.
For example: PF_SF_BAR_SIZE=10 means each SF is taking 1MB of the BAR.

PF_TOTAL_SF=14 means this PCI function can create up to 14 SFs.

In total: FW will allocate 14MB of BAR2.

Note

Perform a [BlueField system-level reset](#) for the mlxconfig settings to take effect.

SF Configuration

To use an SF, a 3-step setup sequence must be followed first:

1. Create.
2. Configure.
3. Deploy.



These steps can be performed using `mlxdevm` tool.

Info

When working on top of an upstream-based kernel, on which the `mlxdevm` tool is unavailable, please refer to the [Upstream Guide on Scalable Functions](#) for instructions on using the `devlink` tool which should be used instead.

Configuration Using `mlxdevm` Tool

1. Create the SF.

SFs are managed using the `mlxdevm` tool supplied with `iproute2` package. The tool is found at `/opt/mellanox/iproute2/sbin/mlxdevm`.

An SF is created using the `mlxdevm` tool. The SF is created by adding a port of `pcisf` flavor.

To create an SF port representor, run:

```
/opt/mellanox/iproute2/sbin/mlxdevm port add pci/<pci_address> flavour  
pcisf pfnnum <corresponding pfnnum> sfnum <sfnum>
```

Note

Each SF must have a unique number (<sfnum>).

For example:

```
/opt/mellanox/iproute2/sbin/mlxdevm port add pci/0000:03:00.0 flavour pcisf  
pfnnum 0 sfnum 4
```

Output example:

```
pci/0000:30:00.0/229409: type eth netdev eth0 flavour pcisf controller 0 pfnm
0 sfnm 4
function:
hw_addr 00:00:00:00:00:00 state inactive opstate detached roce true
max_uc_macs 128 trust off
```

The number 229409 is required to complete the following two steps (i.e., configuration and deployment).

pci/0000:03:00.0/229409 is called the SF index.

pci/<pci_address>/<sf_index> can be replaced with <representor_name>. For example:

```
pci/0000:03:00.0/229409 = en3f0pf0sf4
```

To see information about the created SF such as its MAC address, trust mode, or state (active/inactive), run the following command:

```
/opt/mellanox/iproute2/sbin/mlxdevm port show
```

Output example:

```
pci/0000:30:00.0/229409: type eth netdev en3f0pf0sf4 eth0 flavor pcisf
controller 0 pfnm 0 sfnm 4
function:
hw_addr 00:00:00:00:00:00 state inactive opstate detached roce true
max_uc_macs 128 trust off
```

2. Configure the SF.

A subfunction representor (SF port representor) is created but it is not deployed yet. Users should configure the hardware address (e.g., MAC address), set trust mode to on, and activate the SF before deploying it.

The following steps can be executed as separate commands (at any order) or combined as one:

- To configure the hardware address, run:

```
/opt/mellanox/iproute2/sbin/mlxdevm port function set  
pci/<pci_address>/<sf_index> hw_addr <MAC address>
```

- To set the trust mode to on, run:

```
/opt/mellanox/iproute2/sbin/mlxdevm port function set  
pci/<pci_address>/<sf_index> trust on
```

- To activate the created SF, run:

```
/opt/mellanox/iproute2/sbin/mlxdevm port function set  
pci/<pci_address>/<sf_index> state active
```

Alternatively, to configure the MAC address, set trust mode on, and set the state as active, run:

```
/opt/mellanox/iproute2/sbin/mlxdevm port function set  
pci/<pci_address>/<sf_index> hw_addr <mac_address> trust on state active
```

For example:

```
/opt/mellanox/iproute2/sbin/mlxdevm port function set pci/0000:03:00.0/229409  
hw_addr 00:00:00:00:04:0 trust on state active
```

Note

The SF capabilities above must be set before deploying the SF.

3. Deploy the SF.

To unbind the SF from the default config driver and bind the actual SF driver, run:

```
echo mlx5_core.sf.<next_serial> >
/sys/bus/auxiliary/drivers/mlx5_core.sf_cfg/unbind
echo mlx5_core.sf.<next_serial> > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind
```

For example:

```
echo mlx5_core.sf.4 > /sys/bus/auxiliary/drivers/mlx5_core.sf_cfg/unbind
echo mlx5_core.sf.4 > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind
```

Note

<next_serial> is a number produced by the firmware when creating the SF (this is the gvmi number of the SF). `mlxdevm` tool when creating the SF. To obtain it, refer to the [useful commands](#) provided below.

Useful commands:

- To see the available sub-functions, run:

```
$ devlink dev show
```

For example, if you run the command before creating, configuring, and deploying the SF (using the steps detailed earlier), the output would appear as follows:

```
pci/0000:03:00.0
pci/0000:03:00.1
auxiliary/mlx5_core.sf.2
auxiliary/mlx5_core.sf.3
```

After creating, configuring, and deploying the SF, the output would be:

```
pci/0000:03:00.0
pci/0000:03:00.1
auxiliary/mlx5_core.sf.2
auxiliary/mlx5_core.sf.3
auxiliary/mlx5_core.sf.4
```

Note that the `<next_serial>` number is 4 for the created SF.

- o To see the `sfnum` of each sub-function, run:

```
cat /sys/bus/auxiliary/devices/mlx5_core.sf.<next_serial>/sfnum
```

For example:

```
cat /sys/bus/auxiliary/devices/mlx5_core.sf.4/sfnum
```

Example output:

```
cat /sys/bus/auxiliary/devices/mlx5_core.sf.4/sfnum
4
```

- o To remove an SF, you must first make its state inactive and only then remove the SF representor.

To make the SF's state inactive, run:

```
/opt/mellanox/iproute2/sbin/mlxdevm port function set
pci/<pci_address>/<sf_index> state inactive
```

To delete the SF port representor, run:

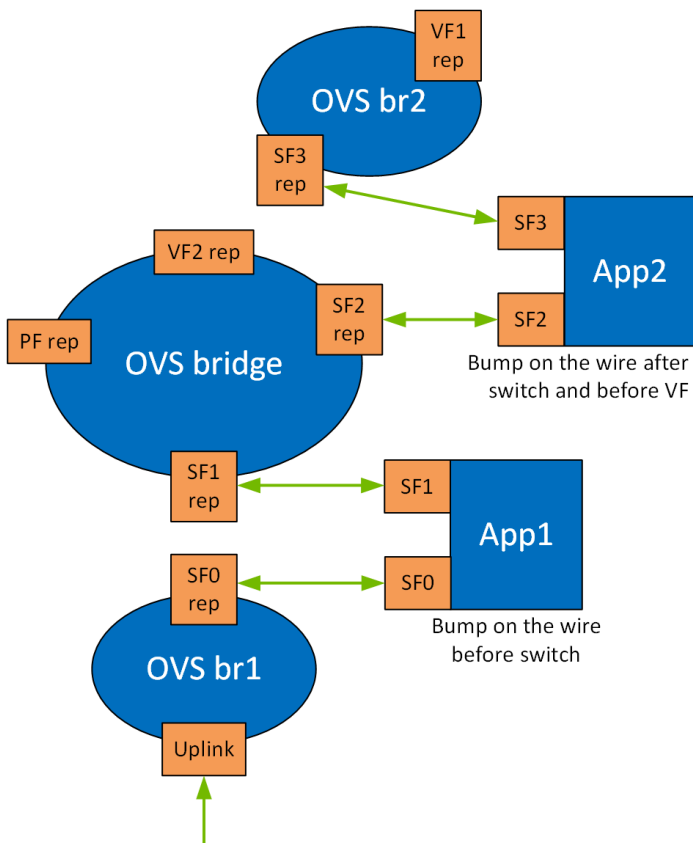
```
/opt/mellanox/iproute2/sbin/mlxdevm port del
```

```
pci/<pci_address>/<sf_index>
```

For example:

```
/opt/mellanox/iproute2/sbin/mlxdevm port function set  
pci/0000:03:00.0/229409 state inactive  
/opt/mellanox/iproute2/sbin/mlxdevm port del pci/0000:03:00.0/229409
```

4. Use the SF.



Running the application on the DPU requires OVS configuration. By creating SFs, an SF representor for the OVS is also created and named `en3f0pf*sf*`. Therefore, each representor needs to be connected to the correct OVS bridge.

Note

Two SFs related to the same PCIe are necessary for the configuration in the illustration.

The following example configures 2 SFs and adds their representors to the OVS.

1. Create, configure, and deploy the SFs. Run:

```
/opt/mellanox/iproute2/sbin/mlxdevm port add pci/0000:03:00.0 flavour pcisf pfnum 0 sfnum 4
/opt/mellanox/iproute2/sbin/mlxdevm port add pci/0000:03:00.0 flavour pcisf pfnum 0 sfnum 5
```

Using the command `mlxdevm port show`, you can see the SF indices of the created SFs.

```
/opt/mellanox/iproute2/sbin/mlxdevm port show
```

Output example:

```
pci/0000:30:00.0/229409: type eth netdev en3f0pf0sf4 flavour pcisf
controller 0 pfnum 0 sfnum 4
function:
hw_addr 00:00:00:00:00:00 state inactive opstate detached roce true
max_uc_macs 128 trust off
pci/0000:30:00.0/229410: type eth netdev en3f0pf0sf5 flavour pcisf
controller 0 pfnum 0 sfnum 5
function:
hw_addr 00:00:00:00:00:00 state inactive opstate detached roce true
max_uc_macs 128 trust off
```

2. Configure the MAC address, set trust mode on, and activate the created SFs:


```
/opt/mellanox/iproute2/sbin/mlxdevm port function set
pci/0000:03:00.0/229409 hw_addr 02:25:f2:8d:a2:4c trust on state active
/opt/mellanox/iproute2/sbin/mlxdevm port function set
pci/0000:03:00.0/229410 hw_addr 02:25:f2:8d:a2:5c trust on state active
```

Using `ifconfig`, you may see that there are 2 added network interfaces: `en3f0pf0sf4` and `en3f0pf0sf5` for the two respective SF port representors.

3. Delete existing OVS bridges (optional).

For example, run the following command to delete an OVS bridge called `ovsbr1`:

```
ovs-vsctl del-br ovsbr1
```

4. Create two bridges `sf_bridge1` and `sf_bridge2` and configure them as follows:

```
ovs-vsctl add-br sf_bridge1
ovs-vsctl add-br sf_bridge2
ovs-vsctl add-port sf_bridge1 p0
ovs-vsctl add-port sf_bridge2 pf0hpf
```

5. Add the port representors to the OVS bridges:

```
ovs-vsctl add-port sf_bridge1 en3f0pf0sf4
ovs-vsctl add-port sf_bridge2 en3f0pf0sf5
```

The OVS bridges after adding the SF representors:

```
Bridge sf_bridge1
Port p0
Interface p0
Port sf_bridge1
Interface sf_bridge1
type: internal
Port en3f0pf0sf4
```

```
Interface en3f0pf0sf4
Bridge sf_bridge2
Port sf_bridge2
Interface sf_bridge2
type: internal
Port en3f0pf0sf5
Interface en3f0pf0sf5
Port pf0hpf
Interface pf0hpf
ovs_version: "2.14.1"
```

(i) Note

The interface might be down by default. Remember to `ifconfig` the interface to "up" status.

(i) Note

When deleting the SF port representor, you must also de-attach it from the bridge it is connected to using the command `ovs-vsctl port-del en3f0pf0sf*`. Otherwise, the port representor will still be connected to the bridge but would not be recognizable.

To run the application, use the following command to initialize the SFs during runtime:

```
*Executable_binary* -a auxiliary:mlx5_core.sf.* -a auxiliary:mlx5_core.sf.*
```

For example:

```
doca_<app_name> -a auxiliary:mlx5_core.sf.4 -a auxiliary:mlx5_core.sf.5 --  
[application_flags]
```

NVIDIA TLS Offload Guide

This guide provides an overview and configuration steps of TLS hardware offloading via kernel-TLS, using hardware capabilities of NVIDIA® BlueField® DPU.

Introduction

Transport layer security (TLS) is a cryptographic protocol designed to provide communications security over a computer network. The protocol is widely used in applications such as email, instant messaging, and voice over IP (VoIP), but its use in securing HTTPS remains the most publicly visible.

The TLS protocol aims primarily to provide cryptography, including privacy (confidentiality), integrity, and authenticity using certificates, between two or more communicating computer applications. It runs in the application layer and is itself composed of two layers: the TLS record and the TLS handshake protocols.

TLS works over TCP and consists of 3 phases:

1. Handshake – establishment of a connection
2. Application – sending and receiving encrypted packets
3. Termination – connection termination

TLS Handshake

In the handshake phase, the client and server decide on which cipher suites they will use, and exchange keys and certificates according to the following flow:

1. Client hello, provides the server at a minimum with the following:
 - A key exchange algorithm, to determine how symmetric keys are exchanged

- An authentication or digital signature algorithm, which dictates how server authentication and client authentication (if required) are implemented
- A bulk encryption cipher, which is used to encrypt the data
- A hash/MAC (message authentication code) function, which determines how data integrity checks are carried out
- The version of the protocol it understands
- The cipher suites it is capable of working with
- A unique random number, which is important to guard against replay attacks

2. Server hello:

- Selects a cipher suite
- Generates its own random number
- Assigns a session ID to the TLS connection
- Sends enough information to complete a key exchange—most often, this means sending a certificate including an RSA public key

3. Client:

- Responsible for completing the key exchange using the information the server provided

At this point, the connection is secured, both sides have agreed on an encryption algorithm, a MAC algorithm, and respective keys.

kTLS

The Linux kernel provides TLS offload infrastructure. kTLS (kernel TLS) offloads TLS handling from the user-space to the kernel-space.

kTLS has 3 modes of operation:

- SW – all operation is handled in kernel (i.e., handshake, encryption, decryption)
- HW-offload (the focus of this guide) – handshake and error handling are performed in software. Packets are encrypted/decrypted in hardware. In this case, there is an additional offload from the kernel to the hardware.
- HW-record – all operations are handled by the hardware (driver and firmware) including the handshake. It also handles its own TCP session. This option is currently not supported.

Note

It is important to understand that Rx (receiving) and Tx (sending) can have two separate modes. For example, Rx can be dealt in SW mode but Tx in HW-offload mode (i.e., the hardware will only encrypt but not decrypt).

HW-offloading kTLS

In general, the TLS HW-offload performs best and provides optimal value on longer lived sessions, with relatively large packets. Scaling in terms of concurrent connections and connections per second is use-case dependent (e.g., the amount of active concurrent connections from the overall open concurrent connections is material).

It is necessary to learn the following terms before proceeding:

- The transport interface send (TIS) object is responsible for performing all transport-related operations of the transmit side. Messages from Send Queues (SQs) get segmented and transmitted by the TIS including all transport required implications. For example, in the case of a large send offload, the TIS is responsible for the segmentation. The NVIDIA® ConnectX® hardware uses a TIS object to save and access the TLS crypto information and state of an offloaded Tx kTLS connection.
- The transport interface receive (TIR) object is responsible for performing all transport-related operations on the receive side. TIR performs the packet

processing and reassembly and is also responsible for demultiplexing the packets into different receive queues (RQs).

- Both TIS and TIR hold the data encryption key (DEK).

kTLS Offload Flow in High Level

Note

The following flow does not include resync and errors.

1. Establishes a TLS connection with remote host (server or client) by handling a TLS handshake by kernel on current host.
2. Initializes the following state for each connection, Rx and Tx:
 - Crypto secrets (e.g., public key)
 - Crypto processing state
 - Record metadata (e.g., record sequence number, offset)
 - Expected TCP sequence number

Tx flow:

1. Packets belonging to device offloaded sockets arrive to the kernel and it does not encrypt them.
2. Kernel performs record framing and marks the packet with a connection identifier.
3. Kernel sends packets to the device driver for offloading.
4. Device checks that the sequence number matches the state in the TIS and performs encryption and authentication.

Rx flow:

1. When the connection is created, a HW steering rule is added to steer packets to their respective TIR.
2. Device receives the packet then validates and checks that sequence number of TCP matches the state in the TIR.
3. Performs decryption and authentication, and indicates in the CQE (completion queue entry).
4. Kernel understands that the packet is already decrypted so it does not decrypt it itself and passes it on to the user-space.

Resync and Error Handling

When the sequence number does not match expectations or if any other error occurs, the hardware gives control back to the SW which handles the problem.

See more about kTLS modes, resync, and error handling in the [Linux Kernel documentation](#).

Prerequisites

All commands in this section should be performed on host (not on BlueField) unless stated otherwise.

Checking Hardware Support for Crypto Acceleration

To check if the BlueField or ConnectX have crypto acceleration, run the following command from host:

```
host> mst start # turn on mst driver
host> flint -d <device under /dev/mst/ directory> dc | grep Crypto
```

The output should include Crypto Enabled. For example:

```
host> flint -d /dev/mst/mt41686_pciconf0 dc | grep Crypto
....
;;Description = NVIDIA BlueField-2 E-Series Eng. sample DPU; 200GbE single-port
QSFP56; PCIe Gen4 x16; Secure Boot Disabled; Crypto Enabled; 16GB on-board
DDR; 1GbE OOB management
....
```

Kernel Requirements

- Operating system must be either:
 - FreeBSD 13.0+.
 - A Linux distribution built on Linux kernel version 5.3 or later for Tx support and version 5.9 or later for Rx support. We recommend using the latest version when possible for the best available optimizations.

Note

TIS Pool optimization is added to Linux kernel version 6.0. Instead of creating TIS per new connection, unused TIS from previous connection, will be recycled. This will improve Tx connection rate. No further installations required beyond installing the kernel itself.

- Check the current kernel version on the host. Run:

```
host> uname -r
```

- The kernel must be configured to support TLS by setting the options `TLS_DEVICE` and `MLX5_TLS` to `y`. To check if TLS is configured, run:


```
host> cat /boot/config-$(uname -r) | grep TLS
```

Example output:

```
host> cat /boot/config-5.4.0-121-generic | grep TLS
...
CONFIG_TLS_DEVICE=y
CONFIG_MLX5_TLS=y
...
```

If the current kernel does not support one of the options, you can change the configurations and recompile, or build a new kernel .

Note

Follow the build instructions provided with the kernel provider.

Schematic flow for building a Linux kernel:

1. Enter the Linux kernel directory downloaded (usually in `/usr/src`):

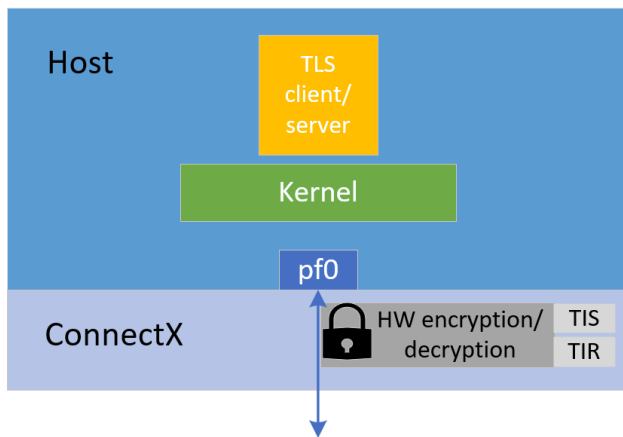
```
host> make menuconfig # Set TLS_DEVICE=y and MLX5_TLS=y in options.
Setting location in the menu can be found by pressing '/' and typing
'setting'.
host> make -j <num-of-cores> && make -j <num-of-cores>
modules_install && make -j <num of cores> install
```

2. Update the grub to the new configured kernel then reboot.

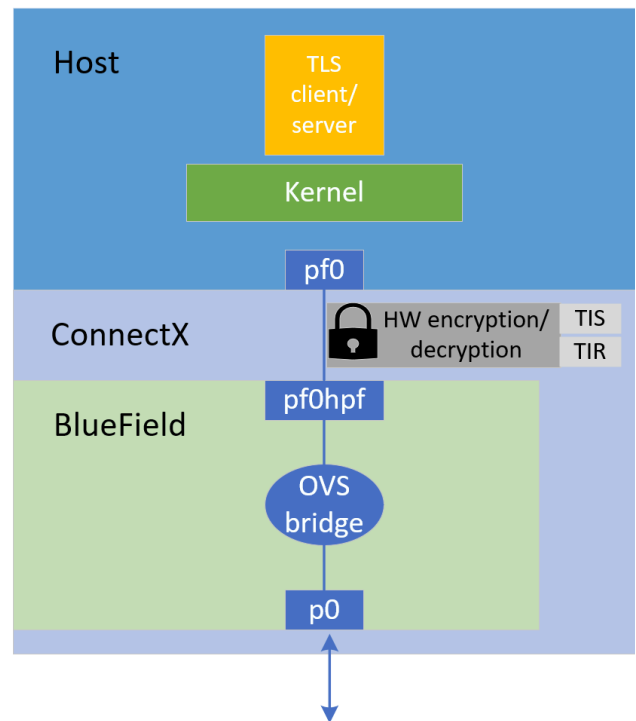
Configurations and Useful Commands

TLS Setup

Host + ConnectX Setup



Host + BlueField Setup



Finding NVIDIA Interfaces

```
host> mst start # if mst driver is not loaded.  
host> mst status -v
```

NVIDIA's netdev interfaces are found be under the NET column.

For example:

```
host> mst status -v  
....  
DEVICE_TYPE MST PCI RDMA NET NUMA  
BlueField2(rev:0) /dev/mst/mt41686_pciconf0.1 b1:00.1 mlx5_1 net-ens5f1 1  
  
BlueField2(rev:0) /dev/mst/mt41686_pciconf0 b1:00.0 mlx5_0 net-ens5f0 1
```

In this example, the interfaces `ens5f1` and `ens5f0` are NVIDIA's netdev interfaces.

Configuring TLS Offload

- To check if the offload option is on or off, run:

```
host> ethtool -k $iface | grep tls
```

Example output:

```
tls-hw-tx-offload: on  
tls-hw-rx-offload: off  
tls-hw-record: off [fixed]
```

Note

tls-hw-record is not required for the device as kTLS does not support "HW Record" mode.

- To turn Tx offload on or off:

```
host> ethtool -K $iface tls-hw-tx-offload <on | off>
```

- To turn Rx offload on or off:

```
host> ethtool -K $iface tls-hw-rx-offload <on | off>
```

Configuring OVS Bridge on BlueField

When the host is connected to a BlueField device, an OVS bridge must be configured on the BlueField so traffic passes bidirectionally from host to uplink. If no OVS bridge is configured, the host is isolated from the network (see [diagram](#) above).

Note

On BlueField image version 3.7.0 or higher the default OVS configuration can be used without additional modifications.

To configure the OVS bridge on BlueField, run the following commands on BlueField:

```
dpu> for br in $(ovs-vsctl list-br); do ovs-vsctl del-br $br; done # erasing existing bridges
dpu> ovs-vsctl add-br ovs-br0 && ovs-vsctl add-port ovs-br0 p0 && ovs-vsctl add-
port ovs-br0 pf0hpf
dpu> ovs-vsctl add-br ovs-br1 && ovs-vsctl add-port ovs-br1 p1 && ovs-vsctl add-
port ovs-br1 pf1hpf
dpu> ovs-vsctl set Open_vSwitch . other_config:hw-offload=true && systemctl restart
openvswitch-switch
```

Where p0/p1 are the uplink interfaces and pf0hpf/pf1hpf are the interfaces facing the host.

Common Use Cases

OpenSSL

OpenSSL is an all-around cryptography library that offers open-source application of the TLS protocol. It is the main library for using kTLS and other applications since Nginx depends on it as their base library.

Note

The kTLS and HW offloading do not depend on OpenSSL. Any program that can implement a TLS stack can be run instead. However, because of the vast use of OpenSSL, this guide addresses installation recommendations.

kTLS is supported only in OpenSSL version 3.0.0 or higher, and only on the [supported kernel versions](#). The supported OpenSSL version is available for download from distro packages, or it can be downloaded and compiled from the OpenSSL GitHub.

Warning

Many modules depend on OpenSSL. Changing the default version may cause problems. Adding `--prefix=/var/tmp/ssl --openssldir=/var/tmp/ssl` in the `./Configure` command below may prevent the built OpenSSL from becoming the default one used by the system. Make sure the directory of the OpenSSL you build manually is not located in any paths listed in the `PATH` environment variable.

1. Check the version of the default OpenSSL:

```
host> openssl version
```

2. Follow OpenSSL installation instructions from OpenSSL's supplied guides. During the configuration process, make sure to set the `enable-ktls` option before building it by running it from within the OpenSSL directory (works in version 3.0 and higher). For example:

```
host> ./Configure linux-$(uname -p) enable-ktls --prefix=/var/tmp/ssl --  
openssldir=/var/tmp/ssl # Add "threads" as well for multithread support
```

3. Check if kTLS is enabled in OpenSSL by running the following command from within the OpenSSL directory, and check whether `ktls` is listed under `Enabled features`:

```
host> perl configdata.pm --dump | less
```

If OpenSSL has been downloaded manually, the OpenSSL executable would be located in the `/<openssl-dir>/apps/` directory. For example, checking the version from within OpenSSL directory is done using the command `./apps/openssl version`.

Note

Installing a new OpenSSL requires recompiling user tools that were configured over OpenSSL (e.g., Nginx).

Note

In OpenSSL's master source code, there is a feature "Support for kTLS Zero-Copy sendfile() on Linux" ([Zero-Copy commit](#)). If the Zero-Copy option is set, `SSL_sendfile()` uses the Zero-Copy TX mode which means that the data itself is not copied from the user space to Kernel space. This gives a performance boost when used with kTLS hardware offload. Be aware that invalid TLS records may be transmitted if the file is changed while being sent.

Nginx

Nginx is a free and open-source software web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. Nginx can be configured to depend on OpenSSL library and therefore Nginx could have the great advantages of TLS HW-offload on ConnectX-6 Dx, ConnectX-7 or the DPU.

Prerequisites

Refer to the [OpenSSL](#) section for setting OpenSSL.

Configuration

1. Install dependencies. For Ubuntu distribution, for example:

```
host> apt install libpcre3 libpcre3-dev
```

2. Clone Nginx's repository and enter directory:

```
host> git clone https://github.com/nginx/nginx.git && cd nginx
```

3. Configure Nginx components to support kTLS:

```
host> ./auto/configure --with-openssl=/<insert_path_to_openssl_directory> --  
with-debug --with-http_ssl_module --with-openssl-opt="enable-ktls -  
DOPENSSL_LINUX_TLS -g3"
```

4. Build Nginx:

```
host> make -j <num of cores> && sudo make -j <num-of-cores> install
```

Note

If make fails with a deprecated openssl functions error, remove `-Werror` for `CFLAGS` in `objs/Makefile` and try again.

5. Add the following lines to the end of the `/usr/local/nginx/conf/nginx.conf` file (before the last closing bracket):

```
server {
listen 443 ssl default_server reuseport;
server_name localhost;
root /tmp/nginx/docs/html/;

include /etc/nginx/default.d/*.conf;
ssl_certificate /usr/local/nginx/conf/cert.pem;
ssl_certificate_key /usr/local/nginx/conf/key.pem;
ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256;
ssl_protocols TLSv1.2;

location / {
index index.html;
}

error_page 404 /404.html;
location = /40x.html {
}

error_page 500 502 503 504 /50x.html;
location = /50x.html {
}
}
```

6. Notice that the key and certificate of the Nginx server should be located in `/usr/local/nginx/conf/`. Therefore, after creating a key and certificate (as mentioned in section "[Adding Certificate and Key](#)") they should be copied to the aforementioned directory:

```
host> cp key.pem /usr/local/nginx/conf/ && cp cert.pem /usr/local/nginx/conf/
```

7. To run Nginx:

```
host> cd nginx && objs/nginx
```


This command starts Nginx Server in the background.

Stopping Nginx

```
host> pkill nginx
```

Wrk - Client

A simple client for requesting Nginx's server is "wrk". It can be installed by running the following:

```
host> git clone https://github.com/wg/wrk.git && cd wrk/ && make -j <num-of-cores>
```

Using Wrk

The following is an example of using the wrk client to request the page index.html from the Nginx server in address 4.4.4.4 (run within wrk's directory):

```
host> taskset -c 0 ./wrk -t1 -c10 -d30s https://4.4.4.4:443/index.html
```

Note

Testing the kTLS offload (with or without hardware offload) is in the same manner as mentioned in section "Testing kTLS". TBD

Testing Offload via OpenSSL

This chapter demonstrates how to test the kTLS hardware offload.

Note

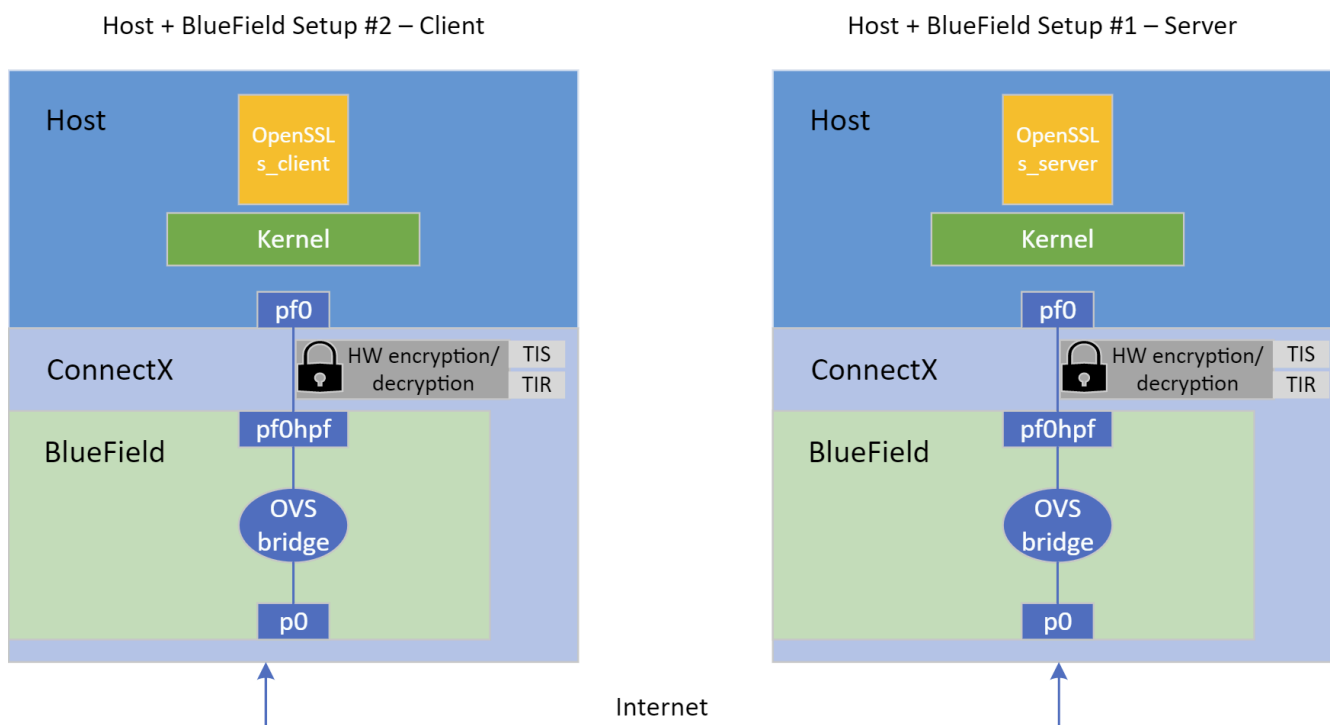
Make sure to refer to section "[OpenSSL](#)" before proceeding.

TLS Testing Setup

For testing purposes, a server and a client are required. The testing section only tests a single setup of a host and BlueField-2 or a host ConnectX which will participate either as a server or as a client. Setting a back-to-back setup of the same kind and installing the same OpenSSL version can help avoid misconfigurations. Nevertheless, it is required to have the same OpenSSL version on both the client and server.

Make sure the desired kTLS is configured as detailed in section "[Configuring TLS Offload](#)". To test hardware offload, make sure `tls-hw-tx-offload` and/or `tls-hw-rx-offload` are on. To test kTLS software mode, make sure to turn them off.

In addition, make sure both hosts (server and client) can communicate bidirectionally through ConnectX or BlueField. One can set the interface that supports the offload (on the host) with an IP, in same subnet. Make sure that when using BlueField, an OVS bridge is set on BlueField as shown in "[Configuring OVS Bridge on BlueField](#)".



Adding Certificate and Key

The server side should create a certificate and key. The client can also use a certificate, but it is not necessary for this test case. Run the following command in the installed OpenSSL directory and fill in all the requested details:

```
host> openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes
```

The following files are created:

- key.pem – private-key file used to generate the CSR and, later, to secure and verify connections using the certificate
- cert.pem – certificate signing request (CSR) file used to order your SSL certificate and, later, to encrypt messages that only its corresponding private key can decrypt

Note

The server side should be run before client side so that client's request are answered by server.

Running Server Side

The following example works on OpenSSL version 3.1.0:

```
host> openssl s_server -key key.pem -cert cert.pem -tls1_2 -cipher ECDHE-RSA-  
AES128-GCM-SHA256 -accept 443 -ktls
```

Note

Notice the `-ktls` flag.

Note

Refer to official OpenSSL documentation on `s_server` for more information.

In this example, the key and certificate are provided, the cipher suite and TLS version are configured, and the server listens to port 443 and is instructed to use kTLS.

Running Client Side

The following example works on OpenSSL version 3.1.0:

```
host> openssl s_client -connect 4.4.4.4:443 -tls1_2
```

Where 4.4.4.4 is the IP of the remote server.

Note

Refer to official OpenSSL documentation on `s_client` for more information.

Testing kTLS

After the connection is established (handshake is done), a prompt will open and the user, both on the client and server side, can send a message to other side in a chat-like manner. Messages should appear on the other side once they are received.

The following example checks kTLS hardware offload on the tested setup by tracking Rx and Tx TLS on device counters:

```
host> ethtool -S $iface | grep -i 'tx_tls_encrypted\|rx_tls_decrypted' # ($iface is the interface that offloads)
```

To check kTLS over kernel counters:

```
host> cat /proc/net/tls_stat
```

Output example:

Note

The comments are not part of the output and are added as explanation.

```
host> cat /proc/net/tls_stat
TlsCurrTxSw 0 # Current Tx connections opened in SW mode
TlsCurrRxSw 0 # Current Rx connections opened in SW mode
TlsCurrTxDevice 0 # Current Tx connections opened in HW-offload mode
TlsCurrRxDevice 0 # Current Rx connections opened in HW-offload mode
TlsTxSw 2323828 # Accumulated number of Tx connections opened in SW mode
TlsRxSw 1 # Accumulated number of Rx connections opened in SW mode
TlsTxDevice 12203652 # Accumulated number of Tx connections opened in HW-offload mode
TlsRxDevice 0 # Accumulated number of Rx connections opened in HW-offload mode
TlsDecryptError 0 # Failed record decryption (e.g., due to incorrect authentication tag)
TlsRxDeviceResync 0 # Rx resyncs sent to HW's handling cryptography
TlsDecryptRetry 0 # All Rx records re-decrypted due to TLS_RX_EXPECT_NO_PAD misprediction
TlsRxNoPadViolation 0 # Data Rx records re-decrypted due to TLS_RX_EXPECT_NO_PAD
misprediction
```

Note

More information about the kernel counters can be found in the [Statistics](#) section of the Kernel TLS documentation.

Optimizations over kTLS

XLIO

The NVIDIA accelerated IO (XLIO) software library boosts the performance of TCP/IP applications based on Nginx (e.g., CDN, DoH) and storage solutions as part of SPDK. XLIO

is a user-space software library that exposes standard socket APIs **with kernel-bypass** architecture, enabling a hardware-based direct copy between an application's user-space memory and the network interface. In particular, XLIO can boost the performance of applications that use the kTLS hardware offload as OpenSSL and Nginx. Read more about XLIO in the [NVIDIA XLIO Documentation](#) and XLIO TLS HW-offload over kTLS in the [TLS HW Offload](#) section.

Note

Even though XLIO is a kernel-bypass library, the kernel must support kTLS for the bypass to work properly.

Performance Tuning Options

TLS offload performance is related to how fast data can be pumped through the offload engine. In the case of user space applications, certain system configurations can be tuned to optimize its performance.

The following are items that can be tuned for optimal performance, mainly focusing on dedicating the server's work to the NUMA, or non-uniform memory access, cores:

Note

Non-uniform memory access (NUMA) cores are cores with a dedicated memory for each of them, granting cores fast access to their own memory and slower access to others'. This architecture is best for scenarios when it is not necessary to share memory between cores.

1. Add NUMA cores of the NIC to the `isolcpus` kernel boot arguments for each server so that the kernel scheduler does not interrupt the core's running user thread. The

following are examples of adding commands:

1. Identify the NIC NUMA node (see NUMA column):

```
host> mst status -v
DEVICE_TYPE MST PCI RDMA NET NUMA
ConnectX6DX(rev:0) /dev/mst/mt4125_pciconf0 41:00.0 mlx5_0 net-
enp65s0f0np0 1
```

2. Identify the cores of the NIC NUMA node using the NUMA node number acquired from the previous output:

```
host> lscpu | grep "NUMA node1"
NUMA node1 CPU(s): 1,3,5,7,9,11,13,15,17,19,21,23
```

3. Add the NIC NUMA cores to a grub file (e.g., /etc/default/grub) by adding the line `GRUB_CMDLINE_LINUX_DEFAULT="isolcpus=<NUMA-cores-from-previous-output>"`. For example:

```
GRUB_CMDLINE_LINUX_DEFAULT="isolcpus=1,3,5,7,9,11,13,15,17,19,21,23"
```

4. Update grub:

```
host> sudo update-grub
```

5. Reboot and check that the configuration has been applied:

```
host> cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-5.10.12 root=UUID=1879326c-711f-4f95-a974-
d732af14ef04 ro department=general user_notifier=dovd osi_string None
BOOTIF=01-90-b1-1c-14-02-44 quiet splash
isolcpus=1,3,5,7,9,11,13,15,17,19,21,23
```

2. Disable irqbalance service:

Note

Interrupt request, or IRQ, determines what hardware interrupts arrive to each core.

```
host> service irqbalance stop
```

3. Run `set_irq_affinity.sh` to redistribute IRQs to various cores.

Note

The script is within MLNX_OFED's sources:

1. You can find it in [MLNX_OFED downloads](#).
2. Under "Download" select the correct version and download the "SOURCES" .tgz file.
3. Extract the .tgz.
4. Under SOURCES, extract the `mlnx_tools`.

You should find both files `set_irq_affinity.sh` and its helper file `common_irq_affinity.sh` under the `sbin` directory.

```
host> ./set_irq_affinity.sh <ConnectX_or_BlueField_network_interface>
```

4. Set the interface RSS to the number of cores to use:

```
host> ethtool -X <ConnectX_or_BlueField_network_interface> equal  
<number_of_isolcpus_cores>
```

5. Set the interface queues for number of cores to use:

```
host> ethtool -L <ConnectX_or_BlueField_network_interface> combined  
<number_of_isolcpus_cores>
```

6. Pin the application with taskset to the isolcpus cores used. For example:

```
host> taskset -c 1,3,5,7,9,11,13,15,17,19,21,23 openssl s_server -key key.pem -  
cert cert.pem -tls1_2 -cipher ECDHE-RSA-AES128-GCM-SHA256 -accept 443 -  
ktls
```

Additional Reading

- [Linux kernel TLS documentation](#)
- [Linux kernel TLS offload documentation](#)
- [Autonomous NIC offloads](#) research paper

NVIDIA DOCA Troubleshooting Guide

This guide provides troubleshooting information for common issues and misconfigurations encountered when using DOCA for NVIDIA® BlueField® DPU.

DOCA Infrastructure

RShim Troubleshooting and How-Tos

Another backend already attached

Several generations of BlueField DPUs are equipped with a USB interface in which RShim can be routed, via USB cable, to an external host running Linux and the RShim driver.

In this case, typically following a system reboot, the RShim over USB prevails and the DPU host reports RShim status as "another backend already attached". This is correct behavior, since

there can only be one RShim backend active at any given time. However, this means that the DPU host does not own RShim access.

To reclaim RShim ownership safely:

1. Stop the RShim driver on the remote Linux. Run:

```
systemctl stop rshim  
systemctl disable rshim
```

2. Restart RShim on the DPU host. Run:

```
systemctl enable rshim  
systemctl start rshim
```

The "another backend already attached" scenario can also be attributed to the RShim backend being owned by the BMC in DPUs with integrated BMC. This is elaborated on further down on this page.

RShim driver not loading

Verify whether your DPU features an integrated BMC or not. Run:

```
# sudo sudo lspci -s $(sudo lspci -d 15b3: | head -1 | awk '{print $1}') -vv | grep  
"Product Name"
```

Example output for DPU **with integrated BMC**:

```
Product Name: BlueField-2 DPU 25GbE Dual-Port SFP56, integrated BMC, Crypto  
and Secure Boot Enabled, 16GB on-board DDR, 1GbE OOB management, Tall  
Bracket, FHHL
```

If your DPU has an integrated BMC, refer to [RShim driver not loading on host with integrated BMC](#).

If your DPU does not have an integrated BMC, refer to [RShim driver not loading on host on DPU without integrated BMC](#).

RShim driver not loading on DPU with integrated BMC

RShim driver not loading on host

1. Access the BMC via the RJ45 management port of the DPU.
2. Delete RShim on the BMC:

```
systemctl stop rshim  
systemctl disable rshim
```

3. Enable RShim on the host:

```
systemctl enable rshim  
systemctl start rshim
```

4. Restart RShim service. Run:

```
sudo systemctl restart rshim
```

If RShim service does not launch automatically, run:

```
sudo systemctl status rshim
```

This command is expected to display "active (running)".

5. Display the current setting. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME  
DEV_NAME pcie-0000:04:00.2
```

This output indicates that the RShim service is ready to use.

RShim driver not loading on BMC

1. Verify that the RShim service is not running on host. Run:

```
systemctl status rshim
```

If the output is active, then it may be presumed that the host has ownership of the RShim.

2. Delete RShim on the host. Run:

```
systemctl stop rshim  
systemctl disable rshim
```

3. Enable RShim on the BMC. Run:

```
systemctl enable rshim  
systemctl start rshim
```

4. Display the current setting. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME  
DEV_NAME usb-1.0
```

This output indicates that the RShim service is ready to use.

RShim driver not loading on host on DPU without integrated BMC

1. Download the suitable DEB/RPM for RShim (management interface for DPU from the host) driver.
2. Reinstall RShim package on the host.

- For Ubuntu/Debian, run:

```
sudo dpkg --force-all -i rshim-<version>.deb
```

- For RHEL/CentOS, run:

```
sudo rpm -Uhv rshim-<version>.rpm
```

3. Restart RShim service. Run:

```
sudo systemctl restart rshim
```

If RShim service does not launch automatically, run:

```
sudo systemctl status rshim
```

This command is expected to display "active (running)".

4. Display the current setting. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME  
DEV_NAME pcie-0000:04:00.2
```

This output indicates that the RShim service is ready to use.

Change ownership of RShim from NIC BMC to host

1. Verify that your card has BMC. Run the following on the host:

```
# sudo sudo lspci -s $(sudo lspci -d 15b3: | head -1 | awk '{print $1}') -vvv | grep  
"Product Name"
```

```
Product Name: BlueField-2 DPU 25GbE Dual-Port SFP56, integrated BMC,  
Crypto and Secure Boot Enabled, 16GB on-board DDR, 1GbE OOB  
management, Tall Bracket, FHHL
```

The product name is supposed to show "integrated BMC".

2. Access the BMC via the RJ45 management port of the DPU.
3. Delete RShim on the BMC:

```
systemctl stop rshim  
systemctl disable rshim
```

4. Enable RShim on the host:

```
systemctl enable rshim  
systemctl start rshim
```

5. Restart RShim service. Run:

```
sudo systemctl restart rshim
```

If RShim service does not launch automatically, run:

```
sudo systemctl status rshim
```

This command is expected to display "active (running)".

6. Display the current setting. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME  
DEV_NAME pcie-0000:04:00.2
```

This output indicates that the RShim service is ready to use.

Connectivity Troubleshooting

Connection (ssh, screen console) to the DPU is lost

The UART cable in the Accessories Kit (OPN: MBF20-DKIT) can be used to connect to the DPU console and identify the stage at which BlueField is hanging.

Follow this procedure:

1. Connect the UART cable to a USB socket, and find it in your USB devices.

```
sudo lsusb  
Bus 002 Device 003: ID 0403:6001 Future Technology Devices International,  
Ltd FT232 Serial (UART) IC
```

Note

For more information on the UART connectivity, please refer to the [DPU's hardware user guide](#) under Supported Interfaces > Interfaces Detailed Description > NC-SI Management Interface.

Info

It is good practice to connect the other end of the NC-SI cable to a different host than the one on which the BlueField DPU is installed.

2. Install the minicom application.

OS	Command
CentOS/RHEL	<pre>sudo yum install minicom -y</pre>
Ubuntu/Debian	<pre>sudo apt-get install minicom</pre>

3. Open the minicom application.

```
sudo minicom -s -c on
```

4. Go to "Serial port setup".

5. Enter "F" to change "Hardware Flow control" to NO.

6. Enter "A" and change to /dev/ttyUSB0 and press Enter.

7. Press ESC.

8. Type on "Save setup as dfl".

9. Exit minicom by pressing Ctrl + a + z.

Driver not loading in host server

What this looks like in dmsg:

```
[275604.216789] mlx5_core 0000:af:00.1: 63.008 Gb/s available PCIe bandwidth,
limited by 8 GT/s x8 link at 0000:ae:00.0 (capable of 126.024 Gb/s with 16 GT/s x8
link)
[275624.187596] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW
initialization, timeout abort in 100s
[275644.152994] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW
initialization, timeout abort in 79s
[275664.118404] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW
initialization, timeout abort in 59s
```

```
[275684.083806] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW
initialization, timeout abort in 39s
[275704.049211] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW
initialization, timeout abort in 19s
[275723.954752] mlx5_core 0000:af:00.1: mlx5_function_setup:1237:(pid 943):
Firmware over 120000 MS in pre-initializing state, aborting
[275723.968261] mlx5_core 0000:af:00.1: init_one:1813:(pid 943): mlx5_load_one
failed with error code -16
[275723.978578] mlx5_core: probe of 0000:af:00.1 failed with error -16
```


The driver on the host server is dependent on the Arm side. If the driver on Arm is up, then the driver on the host server will also be up.

Please verify that:

- The driver is loaded in the BlueField DPU
- The Arm is booted into OS
- The Arm is not in UEFI Boot Menu
- The Arm is not hanged

Then:

1. Perform graceful shutdown.
2. Power cycle on the host server.
3. If the problem persists, reset nvconfig (sudo mlxconfig -d /dev/mst/<device> -y reset) and power cycle the host.

 **Note**

If your DPU is VPI capable, please be aware that this configuration will reset the link type on the network ports to IB. To change the network port's link type to Ethernet, run:

```
sudo mlxconfig -d <device> s LINK_TYPE_P1=2  
LINK_TYPE_P2=2
```

4. If this problem still persists, please make sure to install the latest bfb image and then restart the driver in host server. Please refer to [this page](#) for more information.

No connectivity between network interfaces of source host to destination device

Verify that the bridge is configured properly on the Arm side.

The following is an example for default configuration:

```
$ sudo ovs-vsctl show  
f6740bfb-0312-4cd8-88c0-a9680430924f  
Bridge ovsbr1  
Port pf0sf0  
Interface pf0sf0  
Port p0  
Interface p0  
Port pf0hpf  
Interface pf0hpf  
Port ovsbr1  
Interface ovsbr1  
type: internal  
Bridge ovsbr2  
Port p1  
Interface p1  
Port pf1sf0  
Interface pf1sf0  
Port pf1hpf  
Interface pf1hpf  
Port ovsbr2
```

```
Interface ovsbr2
type: internal
ovs_version: "2.14.1"
```

If no bridge configuration exists, refer to "[Virtual Switch on DPU](#)".

Uplink in Arm down while uplink in host server up

Please check that the cables are connected properly into the network ports of the DPU and the peer device.

Performance Degradation

Degradation in performance indicates that openvswitch may not be offloaded.

Verify offload state. Run:

```
# ovs-vsctl get Open_vSwitch . other_config:hw-offload
```

- If hw-offload = true – Fast Pass is configured (desired result)
- If hw-offload = false – Slow Pass is configured

If hw-offload = false :

- For RHEL/CentOS, run:

```
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true;
# systemctl restart openvswitch;
# systemctl enable openvswitch;
```

- For Ubuntu/Debian, run:

```
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true;
# /etc/init.d/openvswitch-switch restart
```

SR-IOV Troubleshooting

Unable to create VFs

1. Please make sure that SR-IOV is enabled in BIOS.
2. Verify SRIOV_EN is true and NUM_OF_VFS bigger than 1. Run:

```
# mlxconfig -d /dev/mst/mt41686_pciconf0 -e q |grep -i
"SRIOV_EN\|num_of_vf"
Configurations: Default Current Next Boot
* NUM_OF_VFS 16 16 16
* SRIOV_EN True(1) True(1) True(1)
```

3. Verify that GRUB_CMDLINE_LINUX="iommu=pt intel_iommu=on pci=assign-busses".

No traffic between VF to external host

1. Please verify creation of representors for VFs inside the Bluefield DPU. Run:

```
# /opt/mellanox/iproute2/sbin/rdma link |grep -i up
...
link mlx5_0/2 state ACTIVE physical_state LINK_UP netdev pf0vf0
...
```

2. Make sure the representors of the VFs are added to the bridge. Run:

```
# ovs-vsctl add-port <bridge_name> pf0vf0
```

3. Verify VF configuration. Run:

```
$ ovs-vsctl show
bb993992-7930-4dd2-bc14-73514854b024
Bridge ovsbr1
Port pf0vf0
Interface pf0vf0
type: internal
Port pf0hpf
Interface pf0hpf
Port pf0sf0
Interface pf0sf0
Port p0
Interface p0
Bridge ovsbr2
Port ovsbr2
Interface ovsbr2
type: internal
Port pf1sf0
Interface pf1sf0
Port p1
Interface p1
Port pf1hpf
Interface pf1hpf
ovs_version: "2.14.1"
```

eSwitch Troubleshooting

Unable to configure legacy mode

To set devlink to "Legacy" mode in BlueField, run:

```
# devlink dev eswitch set pci/0000:03:00.0 mode legacy
```

```
# devlink dev eswitch set pci/0000:03:00.1 mode legacy
```

Please verify that:

- No virtual functions are open. To verify if VFs are configured, run:

```
# /opt/mellanox/iproute2/sbin/rdma link | grep -i up  
link mlx5_0/2 state ACTIVE physical_state LINK_UP netdev pf0vf0  
link mlx5_1/2 state ACTIVE physical_state LINK_UP netdev pf1vf0
```

If any VFs are configured, destroy them by running:

```
# echo 0 > /sys/class/infiniband/mlx5_0/device/mlx5_num_vfs  
# echo 0 > /sys/class/infiniband/mlx5_1/device/mlx5_num_vfs
```

- If any SFs are configured, delete them by running:

```
/sbin/mlnx-sf -a delete --sfindex <SF-Index>
```

Note

You may retrieve the <SF-Index> of the currently installed SFs by running:

```
# mlnx-sf -a show  
  
SF Index: pci/0000:03:00.0/229408  
Parent PCI dev: 0000:03:00.0  
Representor netdev: en3f0pf0sf0  
Function HWADDR: 02:61:f6:21:32:8c  
Auxiliary device: mlx5_core.sf.2  
netdev: enp3s0f0s0  
RDMA dev: mlx5_2
```

```
SF Index: pci/0000:03:00.1/294944
Parent PCI dev: 0000:03:00.1
Representor netdev: en3f1pf1sf0
Function HWADDR: 02:30:13:6a:2d:2c
Auxiliary device: mlx5_core.sf.3
netdev: enp3s0f1s0
RDMA dev: mlx5_3
```

Pay attention to the SF Index values. For example:

```
/sbin/mlnx-sf -a delete --sfindex pci/0000:03:00.0/229408
/sbin/mlnx-sf -a delete --sfindex pci/0000:03:00.1/294944
```

If the error "Error: mlx5_core: Can't change mode when flows are configured" is encountered while trying to configure legacy mode, please make sure that

1. Any configured SFs are deleted (see above for commands).
2. Shut down the links of all interfaces, delete any ip xfrm rules, delete any configured OVS flows, and stop openvswitch service. Run:

```
ip link set dev p0 down
ip link set dev p1 down
ip link set dev pf0hpf down
ip link set dev pf1hpf down
ip link set dev vxlan_sys_4789 down

ip x s f ;
ip x p f ;

tc filter del dev p0 ingress
tc filter del dev p1 ingress
tc qdisc show dev p0
tc qdisc show dev p1
tc qdisc del dev p0 ingress
```



```
tc qdisc del dev p1 ingress
tc qdisc show dev p0
tc qdisc show dev p1

systemctl stop openvswitch-switch
```

DPU appears as two interfaces

What this looks like:

```
# sudo /opt/mellanox/iproute2/sbin/rdma link
link mlx5_0/1 state ACTIVE physical_state LINK_UP netdev p0
link mlx5_1/1 state ACTIVE physical_state LINK_UP netdev p1
```

- Check if you are working in legacy mode.

```
# devlink dev eswitch show pci/0000:03:00.<0|1>
```

If the following line is printed, this means that you are working in legacy mode:

```
pci/0000:03:00.<0|1>: mode legacy inline-mode none encap enable
```

Please configure the DPU to work in switchdev mode. Run:

```
devlink dev eswitch set pci/0000:03:00.<0|1> mode switchdev
```

- Check if you are working in separated mode:

```
# mlxconfig -d /dev/mst/mt41686_pciconf0 q | grep -i cpu
* INTERNAL_CPU_MODEL SEPERATED_HOST(0)
```

Please configure the DPU to work in embedded mode. Run:

```
# mlxconfig -d /dev/mst/mt41686_pciconf0 s INTERNAL_CPU_MODEL=1
```

DOCA Applications

This chapter deals with troubleshooting issues related to DOCA applications.

EAL Initialization Failure

EAL initialization failure is a common error that may appear while running various DPDK-related applications.

Error

The error looks like this:

```
[DOCA][ERR][NUTILS]: EAL initialization failed
```

There may be many causes for this error. Some of them are as follows:

- The application requires huge pages and none were allocated
- The application requires root privileges to run and it was run without elevated privileges

Solution

The following solutions are respective to the possible causes listed above:

- Allocate huge pages. For example, run (on the host or the DPU, depending on where you are running the application):

```
sudo echo 2048 > /sys/kernel/mm/hugepages/hugepages-  
2048kB/nr_hugepages
```

- Run the application using sudo (or as root):

```
sudo <run_command>
```

Ring Memory Issue

This is a common memory issue when running application on the host.

Error

The error looks as follows:

```
RING: Cannot reserve memory  
[13:00:57:290147][DOCA][ERR][UFLTR::Core:156]: DPI init failed
```

The most common cause for this error is lack of memory (i.e., not enough huge pages per worker threads).

Solution

Possible solutions:

- Recommended: Increase the amount of allocated huge pages. Instructions for allocating huge pages can be found [here](#).

Note

For an SFT application with 64 cores, it is recommended to increase the allocation from 2048 to 8192.

- Alternatively, one can also limit the number of cores used by the application:
 - `-c <core-mask>` – Set the hexadecimal bitmask of the cores to run on.
 - `-l <core-list>` – list of cores to run on.
- For example:

```
./doca_<app_name> -a 3b:00.3 -a 3b:00.4 -l 0-64 -- -l 60
```

DOCA Apps Using DPDK in Parallel Issue

When running two DOCA apps in parallel that use DPDK, the first app runs but the second one fails.

Error

The following error is received:

```
Failed to start URL Filter with output: EAL: Detected 16 lcore(s)
EAL: Detected 1 NUMA nodes
EAL: RTE Version: 'MLNX_DPDK 20.11.4.0.3' EAL: Detected shared linkage of DPDK
EAL: Cannot create lock on '/var/run/dpdk/rte/config'. Is another primary process
running?
EAL: FATAL: Cannot init config
EAL: Cannot init config
[15:01:57:246339][DOCA][ERR][NUTILS]: EAL initialization failed
```

The cause of the error is that the second application is using `/var/run/dpdk/rte/config` when the first application is already using it.

Solution

To run two applications in parallel, the second application needs to be run with DPDK EAL option `--file-prefix <name>`.

In this example, after running the first application (without adding the `eal` option), to run the second with the EAL option. Run:

```
./doca_<app_name> --file-prefix second -a 0000:01:00.6,sft_en=1 -a 0000:01:00.7,sft_en=1  
-v -c 0xff -- -l 60
```

Failure to Set Huge Pages

When trying to configure the huge pages from an unprivileged user account, a permission error is raised.

Error

Configuring the huge pages results in the following error:

```
$ sudo echo 600 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages  
-bash: /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages: Permission  
denied
```

Solution

Using `sudo` with `echo` works differently than users usually expect. Instead, the command should be as follows:

```
$ echo '600' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-  
2048kB/nr_hugepages
```

DOCA Libraries

This chapter deals with troubleshooting issues related to DOCA libraries.

DOCA Flow Error

When trying to add new entry to the pipe, an error is received.

Error

The error happens after trying to add new entry function. The error message would look similar to the following:

```
mlx5_common: Failed to create TIR using DevX
mlx5_net: Port 0 cannot create DevX TIR.
[10:26:39:622581][DOCA][ERR][dpdk_engine]: create pipe entry fail on index:1,
error=Port 0 create flow fail, type 1 message: cannot get hash queue, type=8
```

The issue here seems to be caused by SF/ports configuration.

Solution

To fix the issue, apply the following commands on the DPU:

```
dpu# /opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode
legacy
dpu# /opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode
legacy
dpu# echo none > /sys/class/net/p0/compat/devlink/encap
dpu# echo none > /sys/class/net/p1/compat/devlink/encap
dpu# /opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.0 mode
switchdev
dpu# /opt/mellanox/iproute2/sbin/devlink dev eswitch set pci/0000:03:00.1 mode
switchdev
```

DOCA SDK Compilation

This chapter deals with troubleshooting issues related to compiling DOCA-based programs to use the DOCA SDK (e.g., missing dependencies).

Meson Complains About Missing Dependencies

As part of DOCA's installation, a basic set of environment variables are defined so that projects (such as DOCA applications) could easily compile against the DOCA SDK, and to allow users easy access to the various DOCA tools. In addition, the set of DOCA applications sometimes rely on various 3rd party dependencies, some of which require specific environment variables so to be correctly found by the compilation environment (meson).

Error

There are multiple forms this error may appear in, such as:

- DOCA libraries are missing:

```
Dependency doca found: NO (tried pkgconfig and cmake)
meson.build:13:1: ERROR: Dependency "doca" not found, tried pkgconfig and
cmake
```

- DPDK definitions are missing:

```
Dependency libdpdk found: NO (tried pkgconfig and cmake)
meson.build:41:1: ERROR: Dependency "libdpdk" not found, tried pkgconfig and
cmake
```

- mpicc is missing for DPA All to All application:

```
=====
Skipped Applications
=====
* dpa_all_to_all: Missing mpicc
```

Solution

All the dependencies mentioned above are installed as part of DOCA's installation, and yet it is recommended to check that the packages themselves were installed correctly. The packages that install each dependency define the environment variables needed by it, and apply these settings per user login session:

- If DOCA was just installed (on the host or DPU), user session restart is required to apply these definitions (i.e., log off and log in).
- It is important to compile DOCA using the **same** logged in user. Logging as ubuntu and using `sudo su`, or compiling using `sudo`, will not work.

If restarting the user session is not possible (e.g., automated non-interactive session), the following is a list of the needed environment variables:

Note

All the following examples use the required environment variables for the DPU. For the host, the values should be adjusted accordingly (aarch64 is for the DPU and x86 is for the host): aarch64-linux-gnu
x86_64-linux-gnu.

Tip

It is recommended to define all of the following settings so as to not have to remember which DOCA application requires which module (whether DPDK, FlexIO, etc).

DOCA Libraries & Tools:

- For Ubuntu:


```
export
PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/opt/mellanox/doca/lib/aarch64-
linux-gnu/pkgconfig
export PATH=${PATH}:/opt/mellanox/doca/tools
```

- For CentOS:

```
export
PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/opt/mellanox/doca/lib64/pkgconfig
export PATH=${PATH}:/opt/mellanox/doca/tools
```

DOCA Applications:

- For Ubuntu and CentOS

```
export PATH=${PATH}:/usr/mpi/gcc/openmpi-4.1.7a1/bin
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/mpi/gcc/openmpi-
4.1.7a1/lib
```

DPDK:

- For Ubuntu:

```
export
PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/opt/mellanox/dpdk/lib/aarch64-
linux-gnu/pkgconfig
```

- For CentOS:

```
export
PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/opt/mellanox/dpdk/lib64/pkgconfig
```

FlexIO:

- For Ubuntu:

```
export
PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/opt/mellanox/flexio/lib/pkgconfig
```

- For CentOS:

```
export
PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/opt/mellanox/flexio/lib/pkgconfig
```

CollectX:

- For Ubuntu and CentOS:

```
export
PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/opt/mellanox/collectx/lib/aarch64-
linux-gnu/pkgconfig
```

Meson Complains About Permissions

Our guides for compiling the reference samples and applications of DOCA's SDK are using the [meson build system](#).

Error

A permission error is encountered when trying to **reuse** a build directory from a previous build:

```
ubuntu@localhost:/opt/mellanox/doca/samples/doca_flow/flow_acl$ meson
/tmp/build
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/mesonbuild/mesonmain.py", line 146, in run
return options.run_func(options)
File "/usr/lib/python3/dist-packages/mesonbuild/msetup.py", line 294, in run
app.generate()
```

```
File "/usr/lib/python3/dist-packages/mesonbuild/msetup.py", line 181, in generate
mlog.initialize(env.get_log_dir(), self.options.fatal_warnings)
File "/usr/lib/python3/dist-packages/mesonbuild/mlog.py", line 103, in initialize
log_file = open(os.path.join(logdir, log_fname), 'w', encoding='utf-8')
PermissionError: [Errno 13] Permission denied: '/tmp/build/meson-logs/meson-log.txt'
```

Solution

Per the meson [build instructions](#), the user can choose any write-accessible directory to be used as the build directory, using the following syntax:

```
meson <build-dir>
```

When reusing a build directory, it is best to ensure that the existing directory was created by a user with the same permissions, and only then do one of the following:

- Removing the old build directory:

```
rm -rf /tmp/build
```

- Reconfiguring the build directory:

```
meson --reconfigure /tmp/build
```

The above error is an indication that the build directory was created by a different user, and that our user doesn't have permissions to use it. In such cases, it is best to choose a **different** build directory, in a directory that our user has write-access to. For example:

```
meson /tmp/build2
```

Static Compilation on CentOS: Undefined References to C++

When statically compiling against the DOCA SDK on RHEL 7.x machines, there could be a conflict between the libstdc++ version available out-of-the-box and the one used when building DOCA's SDK libraries.

Error

There are multiple forms this error may appear in, such as:

```
$ cc test.o -o test_out `pkg-config --libs --static doca`  
/opt/mellanox/doca/lib64/libdoca_common.a(doca_common_core_src_doca_dev.cpp.o)  
In function `doca_devinfo_rep_list_create':  
(.text.experimental+0x2193): undefined reference to  
`__cxa_throw_bad_array_new_length'  
/opt/mellanox/doca/lib64/libdoca_common.a(doca_common_core_src_doca_dev.cpp.o): In function  
`doca_devinfo_rep_list_create':  
(.text.experimental+0x2198): undefined reference to  
`__cxa_throw_bad_array_new_length' collect2: error: ld returned 1 exit status
```

Solution

Upgrading the devtoolset on the machine to the one used when building the DOCA SDK resolves the undefined references issue:

```
$ sudo yum install epel-release  
$ sudo yum install centos-release-scl-rh  
$ sudo yum install devtoolset-8  
# This will enable the use of devtoolset-8 to the *current* bash session  
$ source /opt/rh/devtoolset-8/enable
```

Static Compilation on CentOS: Unresolved Symbols

When statically compiling against the DOCA SDK on RHEL 7.x machines, a [known issue](#) in the default pkg-config version (0.27) causes a linking error.

Error

There are multiple forms this error may appear in. For example:

```
$ cc test.o -o test_out 'pkg-config --libs --static doca' ...  
/opt/mellanox/dpdk/lib64/librte_net_mlx5.a(net_mlx5_mlx5_sft.c.o): In function  
'mlx5_sft_start':  
mlx5_sft.c:(.text+0x1827): undefined reference to 'mlx5_malloc' ...
```

Solution

Use an updated version of `pkg-config` or `pkgconf` instead when building applications (as is recommended in [DPDK's compilation instructions](#)).

Cross-compiling DOCA and CUDA

This chapter deals with troubleshooting issues related to DOCA-CUDA cross-compilation.

Application Build Error

When trying to build with meson, an architecture-related error is received.

Error

The error may happen when trying to build DOCA or DOCA-CUDA applications.

```
cc1: error: unknown value 'corei7' for -march
```

It indicates that some dependency (usually `libdpdk`) is not taken from the host machine (i.e., the machine the executable file should be running on). This dependency should be taken from the Arm dependencies directories (the path is specified in the cross file) but is skipped if the host's `PKG_CONFIG_PATH` environment variable is used instead.

Solution

Make sure that the cross file contains the following PKG_CONFIG related definitions:

```
[built-in options]
pkg_config_path = " [properties]
pkg_config_libdir = ... // Some content here
```

In addition, verify that `pkg_config_libdir` properly points to all `pkgconfig`-related directories under your cross-build root directory, and that the dependency reported in the error is not missing.

DOCA Services (Containers)

This section deals with troubleshooting issues related to DOCA-based containers.

YAML Syntax Error #1

When deploying the container using the respective YAML file, the pod fails to start.

Error

The error may happen after modifying a service's YAML file, or after copying an example YAML file from one of the guides.

```
$ crictl pods
POD ID CREATED STATE NAME NAMESPACE ATTEMPT RUNTIME
$ journalctl -u kubelet
...
Oct 06 12:10:08 dpu-name kubelet[3260]: E1006 12:10:08.552306 3260 file.go:108] "Unable to
process watch event" err="can't process config file \"/etc/kubelet.d/file_name.yaml": invalid pod:
[metadata.name: Invalid value: \"-dpu-name\": a lowercase RFC 1123 subdomain must consist of lower
case alphanumeric characters, '-' or '.', and must start and end with an alphanumeric character (e.g.
'example.com', regex used for validation is '[a-z0-9]([-a-z0-9]*[a-z0-9])?(\\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*'
spec.containers: Required value]"
```

...

This indicates that some of the fields in the YAML file fail to comply with RFC 1123.

Solution

Both the pod name and container name have a strict alphabet (RFC 1123) restrictions. This means that users can only use dash ("-") and not underscore ("_") as the latter is an illegal character and cannot be used in the pod/container name. However, for the container's image name, use underscore ("_") instead of dash ("-") to help differentiate the two.

YAML Syntax Error #2

When deploying the container using the respective YAML file, the pod fails to start.

Error

The error may happen after modifying a service's YAML file, or after copying an example YAML file from one of the guides.

Note

This error can occur when there is a whitespace issue if the YAML file has been copied from one of the guides causing a formatting mistake. It is important to ensure that the space characters used in the files are indeed spaces (" ") and not some other whitespace character.

```
$ crictl pods
```

```
POD ID CREATED STATE NAME NAMESPACE ATTEMPT RUNTIME
```

```
$ journalctl -u kubelet
...
Oct 04 12:35:58 dpu-name kubelet[3046]: E1004 12:35:58.744406 3046 file.go:187] "Could not
process manifest file" err="/etc/kubelet.d/file_name.yaml: couldn't parse as pod(yaml: line 48: did not
find expected '-' indicator), please check config file" path="/etc/kubelet.d/file_name.yaml"
...
```

This indicates that there is a probable indentation issue in line 48 or in the line above it.

Solution

Go over the file and make sure that the file only uses spaces (" ") for indentations (2 per indent). Using any other number of spaces causes undefined behavior.

Missing Huge Pages

When deploying the container using the respective YAML file, the pod fails to start.

Error

```
$ crictl pods
POD ID CREATED STATE NAME NAMESPACE ATTEMPT RUNTIME
$ journalctl -u kubelet
...
Oct 04 12:39:41 dpu-name kubelet[3046]: I1004 12:39:41.643621 3046 predicate.go:103]
"Failed to admit pod, unexpected error while attempting to recover from admission failure"
pod="default/file_name" err="preemption: error finding a set of pods to preempt: no set of running
pods found to reclaim resources: [(res: hugepages-2Mi, q: 1021313024), ]"
...
```

This error indicates that the service expected 1GB (1021313024 bytes) of huge pages of size 2MB per page, and could not find them.

Solution

1. Remove the YAML file of the service from the deployment directory (/etc/kubelet.d).
2. Allocate huge pages as described in the service's prerequisites steps:
 1. Make sure that the huge pages are allocated as required per the desired container.
 2. Both the amount and size of the pages are important and must match precisely.
3. Restart the container infrastructure daemons:

```
sudo systemctl restart kubelet.service  
sudo systemctl restart containerd.service
```

4. Once the above operations are completed successfully, the container could be deployed (YAML can be copied to /etc/kubelet.d).

Failed to Reserve Sandbox Name

After rebooting the DPU, the respective pods start. However, the containers repeatedly fail to spawn and their "attempt" counter does not increment.

Error

```
$ crictl pods  
POD ID CREATED STATE NAME NAMESPACE ATTEMPT RUNTIME  
bee147792a85b Less than a second ago Ready doca-hbn-service-my-dpu default 0  
(default)  
ea66ee46e75a5 Less than a second ago Ready doca-telemetry-service-my-dpu  
default 0 (default)
```

```

$ crictl ps -a
CONTAINER IMAGE CREATED STATE NAME ATTEMPT POD ID POD
6a35c025a3590 ce4c0cafd583e Less than a second ago Exited init-sfs 0
bee147792a85b doca-hbn-service-my-dpu
9048f4c7b8f3c 095a5833a3f80 Less than a second ago Running doca-telemetry-
service 0 ea66ee46e75a5 doca-telemetry-service-my-dpu
059d0aa8a3199 095a5833a3f80 Less than a second ago Exited init-telemetry-
service 0 ea66ee46e75a5 doca-telemetry-service-my-dpu
bcfbe536271ea ce4c0cafd583e 33 seconds ago Running init-sfs 1 bee147792a85b
doca-hbn-service-my-dpu

$ journalctl -u containerd
...
"2023-11-28T08:43:42.408173348+02:00" level=error msg="RunPodSandbox for
&PodSandboxMetadata{Name:doca-hbn-service-my-
dpu,Uid:823b1ad0e241a10475edde26e905856b,Namespace:default,Attempt:0,} failed, error"
error="failed to reserve sandbox name \"doca-hbn-service-my-
dpu_default_823b1ad0e241a10475edde26e905856b_0\": name \"doca-hbn-service-my-
dpu_default_823b1ad0e241a10475edde26e905856b_0\" is reserved for
\"bee147792a85bc23a3629a9fcd0a5f388794f6b67ef552c959d4d5e49d04f5b2\""
...

```

This error indicates that there has been some collision with prior instances of the doca-hbn-service container, probably pre-reboot.

Note

This issue indicates irregularities in the time of the machine, and usually that the DPU's time pre-reboot was later than the time post-reboot. This leads to bugs in the recovery of the container infrastructure daemons. It is of utmost importance that the time of the system does not jump backwards.

Solution

1. Remove all YAML files from the deployment directory (/etc/kubelet.d).

2. Stop all pods:

```
sudo crictl stop $(crictl pods | tail -n +2 | awk '{ print $1 }')
```

3. Clear all containers:

```
sudo ctr -n k8s.io container rm $(ctr -n k8s.io container ls | tail -n +2 | awk '{ print $1 }')
```

4. Make sure the system's time is correct, and adjust it if needed:

```
date
```

5. Restart the container infrastructure daemons:

```
sudo systemctl restart kubelet.service  
sudo systemctl restart containerd.service
```

6. Once the above operations are completed successfully, the container could be deployed (YAML can be copied to /etc/kubelet.d).

NVIDIA DOCA Virtual Functions User Guide

This guide provides an overview and configuration of virtual functions for NVIDIA® BlueField® and demonstrates a use case for running the DOCA applications over x86 host.

Introduction

Single root IO virtualization (SR-IOV) is a technology that allows a physical PCIe device to present itself multiple times through the PCIe bus. This technology enables multiple virtual instances of the device with separate resources. NVIDIA adapters are able to expose virtual instances or functions (VFs) for each port individually. These virtual functions can then be provisioned separately.

Each VF can be seen as an additional device connected to the physical interface or function (PF). It shares the same resources with the PF, and its number of ports equals those of the PF.

SR-IOV is commonly used in conjunction with an SR-IOV-enabled hypervisor to provide virtual machines direct hardware access to network resources, thereby increasing its performance.

There are several benefits to running applications on the host. For example, one may want to utilize a strong and high-resource host machine, or to start DOCA integration on the host before offloading it to the BlueField DPU.

The configuration in this document allows the entire application to run on the host's memory, while utilizing the HW accelerators on BlueField.

When VFs are enabled on the host, VF representors are visible on the Arm side which can be bridged to corresponding PF representors (e.g., the uplink representor and the host representor). This allows the application to only scan traffic forwarded to the VFs as configured by the user and to behave as a simple "bump-on-the-wire". DOCA installed on the host allows access to the hardware capabilities of the BlueField DPU without comprising features which use HW offload/steering elements embedded inside the eSwitch.

Prerequisites

To run all the reference applications over the host, you must install the host DOCA package. Refer to the [NVIDIA DOCA Installation Guide for Linux](#) for more information on host installation.

VFs must be configured as trusted for the hardware jump action to work as intended. The following steps configure "trusted" mode for VFs:

1. Delete all existing VFs

1. To delete all VFs on a PF run the following on the host:

```
$ echo 0 > /sys/class/net/<physical_function>/device/sriov_numvfs
```

For example:

```
$ echo 0 > /sys/class/net/ens1f0/device/sriov_numvfs
```

2. Delete all existing SFs.

Info

Refer to [NVIDIA BlueField DPU Scalable Function User Guide](#) for instructions on deleting SFs.

3. Stop the main driver on the host:

```
/etc/init.d/openibd stop
```

4. Before creating the VFs, set them to "trusted" mode on the device by running the following commands on the DPU side.

1. Setting VFs on port 0:

```
$ mlxreg -d /dev/mst/mt41686_pciconf0 --reg_id 0xc007 --reg_len 0x40 --  
indexes "0x0.0:32=0x80000000" --yes --set "0x4.0:32=0x1"
```

2. Setting VFs on port 1:

```
$ mlxreg -d /dev/mst/mt41686_pciconf0.1 --reg_id 0xc007 --reg_len 0x40 --  
indexes "0x0.0:32=0x80000000" --yes --set "0x4.0:32=0x1"
```

(i) Note

These commands set trusted mode for all created VFs/SFs after their execution on the DPU.

(i) Note

Setting trusted mode should be performed once per reboot.

5. Restart the main driver on the host by running the following command:

```
/etc/init.d/openibd restart
```

VF Creation

1. Make sure mst driver is running:

```
host $ mst status
```

If it is not loaded, run:

```
host $ mst start
```

2. Enable SR-IOV. Run:

```
host $ mlxconfig -y -d /dev/mst/mt41686_pciconf0 s SRIOV_EN=1
```

3. Set number of VFs. Run:

```
host $ mlxconfig -y -d /dev/mst/mt41686_pciconf0 s NUM_OF_VFS=X
```

Note

Perform a [BlueField system reboot](#) for the mlxconfig settings to take effect.

```
host $ echo X > /sys/class/net/<physical_function>/device/sriov_numvfs
```

For example:

```
host $ mlxconfig -y -d /dev/mst/mt41686_pciconf0 s NUM_OF_VFS=2
host $ reboot
host $ echo 2 > /sys/class/net/ens1f0/device/sriov_numvfs
```

After enabling VF, the representor appears on the DPU. The function itself is seen at the x86 side.

4. To verify that the VFs have been created. Run:

```
$ lspci | grep Virtual
b1:00.3 Ethernet controller: Mellanox Technologies ConnectX Family mlx5Gen
Virtual Function (rev 01)
b1:00.4 Ethernet controller: Mellanox Technologies ConnectX Family mlx5Gen
Virtual Function (rev 01)
b1:01.3 Ethernet controller: Mellanox Technologies ConnectX Family mlx5Gen
Virtual Function (rev 01)
```

Note

2 new virtual Ethernet devices are created in this example.

Running DOCA Application on Host

(i) Note

Allocate the required amount of VFs as explained previously.

(i) Note

Allocate any other resources as specified by the application (e.g., huge pages).

The following is the CLI example for running a reference application over the host using VF:

```
./opt/mellanox/doca/applications/<app_name>/bin/doca_<app_name> -a "pci address VF0" -a "pci address VF1" -c 0xff -- [application flags]
```

The following is an example with specific PCIe addresses for the VFs:

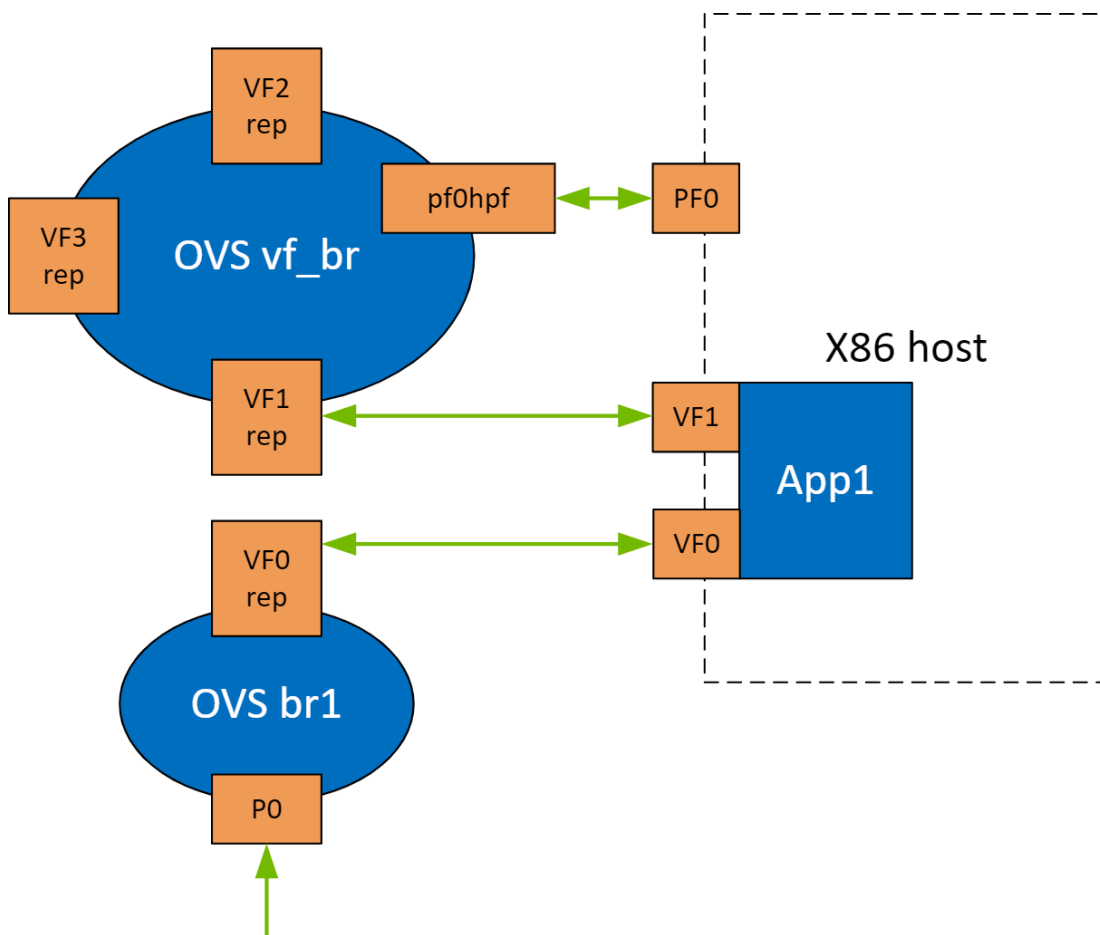
```
./opt/mellanox/doca/applications/<app_name>/bin/doca_<app_name> -a b1:00.3 -a b1:00.4 -c 0xff -- -l 60
```

(i) Note

By default, a DPDK application initializes all the cores of the device. This is usually unnecessary and may even cause unforeseeable issues. It is recommended to limit the number of cores, especially when using an AMD-based system, to 16 cores using the `-c` flag when running DPDK.

Topology Example

The following is a topology example for running the application over the host.



Configure the OVS on BlueField as follows:

```
Bridge ovsbr1
Port ovsbr1
Interface ovsbr1
```

```
type: internal
Port pf0hpf
Interface pf0hpf
Port pf0vf1
Interface pf0vf1
Bridge vf_br
Port p0
Interface p0
Port vf_br
Interface vf_br
type: internal
Port pf0vf0
Interface pf0vf0
```

When enabling a new VF over the host, VF representors are created on the Arm side. The first OVS bridge connects the uplink connection (p0) to the new VF representor (pf0vf0), and the second bridge connects the second VF representor (pf0vf1) to the host representors (pf0hpf). On the host, the 2 PCIe addresses of the newly created function must be initialized when running the applications.

When traffic is received (e.g., from the uplink), the following occurs:

1. Traffic is received over p0.
2. Traffic is forwarded to pf0vf0.
3. Application "listens" to pf0vf0 and pf0vf1 and can, therefore, acquire the traffic from pf0vf0, inspect it, and forward to pf0vf1.
4. Traffic is forwarded from pf0vf1 to pf0hpf.

VF Creation on Adapter Card

Note

Supported only for NVIDIA® ConnectX®-6 Dx based adapter cards and higher.

The following steps are required only when running DOCA applications on an adapter card.

1. Set trust level for all VFs. Run:

```
host# mlxreg -d /dev/mst/mt4125_pciconf0 --reg_name VHCA_TRUST_LEVEL --yes --set "all_vhca=0x1,trust_level=0x1" --indexes "vhca_id=0x0,all_vhca=0x0"
```

2. Create X VFs (X being the required number of VFs) and run the following to turn on trusted mode for the created VFs:

```
echo ON | tee /sys/class/net/enp1s0f0np0/device/sriov/X/trust
```

For example, if you are creating 2 VFs, the following commands should be used:

```
echo ON | tee /sys/class/net/enp1s0f0np0/device/sriov/0/trust  
echo ON | tee /sys/class/net/enp1s0f0np0/device/sriov/1/trust
```

3. Create a VF representor using the following command, replace the PCIe address with the PCIe address of the created VF:

```
echo 0000:17:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind  
echo 0000:17:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
```

Archives

This section contains the following pages:

- [NVIDIA DOCA LTS Versions](#)
- [NVIDIA DOCA Documentation Archives](#)

NVIDIA DOCA LTS Versions

Documentation for DOCA long term support (LTS) releases.

Introduction

DOCA LTS releases are stable and verified DOCA versions. LTS updates include bug fixes and security vulnerability fixes but not ongoing features and enhancements.

LTS Documentation

Follow these links to navigate to the relevant LTS release or specific update:

- [DOCA 2.5.0 LTS base version](#)
 - [2.5.1 LTS update](#)
- [DOCA 1.5.0 LTS base version](#)
 - [1.5.1 LTS update](#)
 - [1.5.2 LTS update](#)
 - [1.5.3 LTS update](#)

NVIDIA DOCA Documentation Archives

Archived documentation of previous DOCA software releases.

- [DOCA v2.6.0 documentation](#)
- [DOCA v2.5.1 documentation](#)
- [DOCA v2.5.0 documentation](#)
- [DOCA v2.2.1 documentation](#)
- [DOCA v2.2.0 documentation](#)
- [DOCA v2.0.2 documentation](#)
- [DOCA v1.5.3 documentation](#)
- [DOCA v1.5.2 documentation](#)
- [DOCA v1.5.1 documentation](#)
- [DOCA v1.5.0 documentation](#)
- [DOCA v1.4.0 documentation](#)
- [DOCA v1.3.0 documentation](#)
- [DOCA v1.2.1 documentation](#)
- [DOCA v1.2.0 documentation](#)
- [DOCA v1.1.1 documentation](#)
- [DOCA v1.1.0 documentation](#)
- [DOCA v1.0.0 documentation](#)

© Copyright 2024, NVIDIA. PDF Generated on 06/07/2024