DPL System Overview

Table of contents

This section outlines the structure and capabilities of the NVIDIA® BlueField® programmable pipeline, focusing on its implementation within the DOCA Pipeline Language (DPL) framework. The syntax from the P4-16 language enables users to describe the behavior of the packet processing pipeline, while DPL provides an NVIDIA platform-specific target architecture (TA) optimized for the BlueField networking platform's hardware (DPU or SuperNIC).

Introduction

The P4 Language is a powerful and flexible programming language designed to define advanced network behavior. DOCA Programming Language (DPL) is based on P4-16, reusing its syntax to provide a familiar programming interface. However, some of the semantics, programmability features and behavioral models are DOCA specific. Unlike traditional approaches that rely on APIs, DPL enables developers to specify packet-handling rules directly through code. These rules determine how packets are processed, including header parsing, matching logic, and actions such as forwarding, dropping, or modifying packets. DPL offers the flexibility to work with existing header formats or define custom headers. Developers can then use these header fields for matching and manipulation, providing extensive control over packet processing. Compared to implementing equivalent pipelines using C-based APIs, developing DPL programs is faster and more efficient. Modifications are simpler to make, and the DPL compiler automatically optimizes the pipeline, allowing developers to focus on correctness rather than performance.

When a DPL program is executed, runtime control is achieved using P4Runtime, an open-source protocol that allows dynamic updates to rules without recompiling the software stack. This enables real-time reconfiguration of packet processing pipelines. While any P4Runtime compliant SDN controller can be used with DPL, some vendor specific extensions will be included in the future.

The **DPL Solution** provides a comprehensive toolkit for compiling, executing, monitoring, and debugging DPL programs. It includes a library of commonly used header types and consists of three parts: a compiler, a runtime service and the developer tools. The service runs on NVIDIA BlueField DPUs, while the development tools can operate on any system, such as a laptop. Together, these components facilitate efficient development and deployment of DPL-based applications on BlueField platforms.

BlueField Network Configuration

BlueField-3 is a powerful cloud infrastructure processor that revolutionizes data center operations by offloading, accelerating, and isolating software-defined networking, storage, security, and management functions. In the context of processing packets in the data path, BlueField-3 leverages the NVIDIA® ConnectX® ASIC for high-speed networking

capabilities, enabling efficient packet processing and data transfer. The Arm subsystem within the BlueField-3 provides programmable processing power for network functions and management tasks, enhancing overall system flexibility and performance. Additionally, BlueField-3 works in tandem with the host CPU to offload networking and security tasks, freeing up CPU resources to focus on running applications, thereby optimizing data center performance and efficiency.

BlueField-3 supports two DOCA device types:

- Local device – this is an actual device exposed in the local system (BlueField or host) and can perform DOCA library processing jobs. This can be a PCIe physical function (PF) virtual function (VF) or scalable function (SF)

- Netdev representor – this is a representation of a local device. The represented local device is usually on the host (except for SFs) and the representor is always on the BlueField side (a proxy on the BlueField for the host-side device). The representors map each one of the host side physical and virtual functions where it:

    - Serves 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.

    - Serves as the channel to configure the embedded switch with rules to the corresponding represented function.

These representors are used as the virtual ports that connect to OVS and any other virtual switch running on the Arm cores.

When in embedded CPU function (ECPF) ownership mode (also called DPU mode), there are 2 representors for each of the BlueField'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 is 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>`

- SF representors - en3f0pf0sf1 `en<pcie_domain>f<pcie_function>pf<port_number>sf<function_number>`

Traffic from the x86 host server hosting the BF3 to an external host will go via BlueField's Arm running OVS-DOCA. See the section BlueField DPU Mode under DOCA Switching for further details.

> **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.

DPL Architecture

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, BlueField 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 BlueField DPU.

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 BlueField has loaded and completed NIC configuration.

2. All ICM (Interface Configuration Memory) is allocated by the ECPF and resides in BlueField's memory.

3. The ECPF controls and configures the NIC embedded switch which means that traffic to and from the host (BlueField) interface always lands on the Arm side.

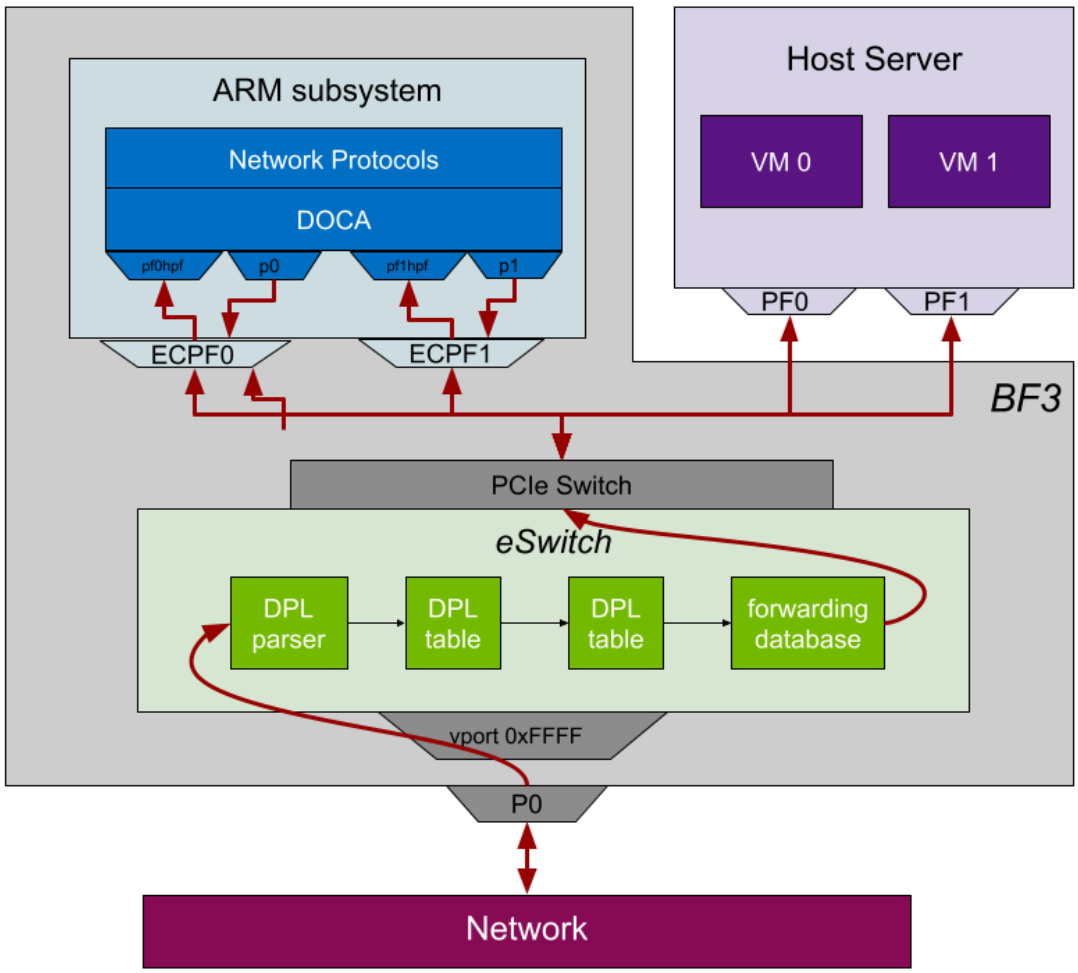
A deployed DPL program executes on the eSwitch component of the BlueField hardware. Basic DPL processing blocks consist of:

- A programmable packet parser based on a native parse graph

- A pipeline of match/action flow tables

- Actions on packets based on the outcome of a lookup. Actions include packet modification, encapsulation, payload encryption, forwarding actions and packet

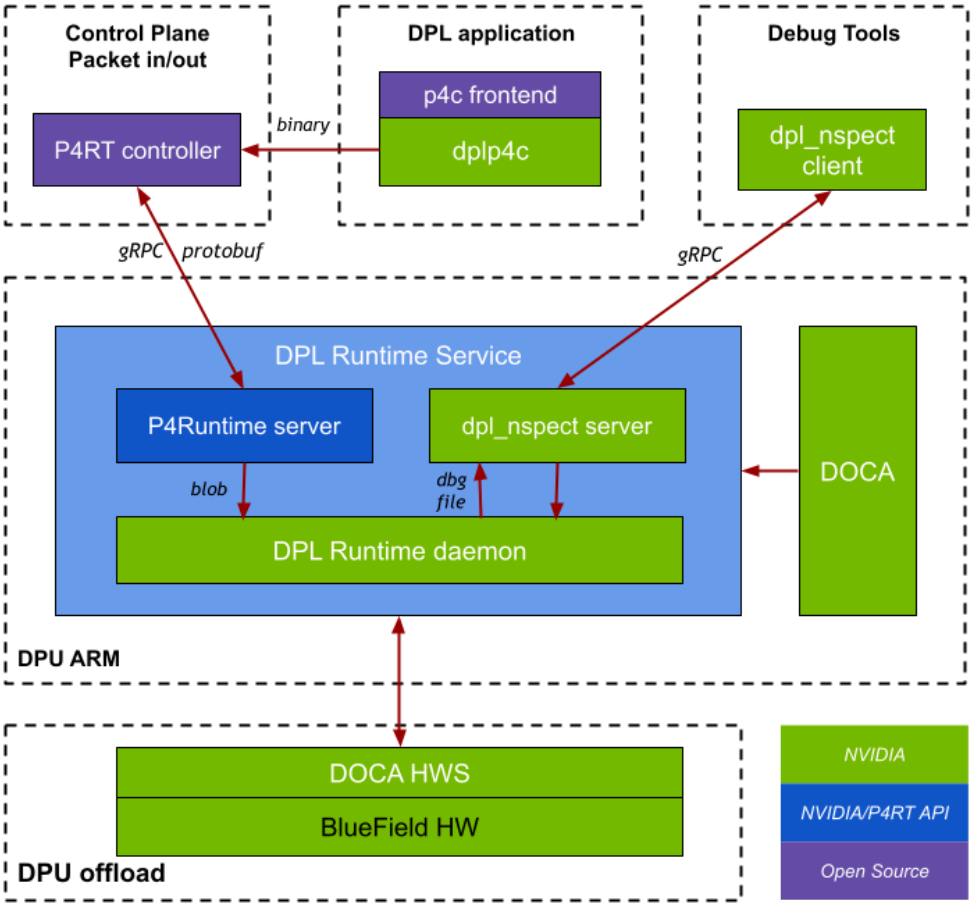
duplication.



- `pf0hpf` and `pf1hpf` are the representors facing the x86 host
- `pf0` and `pf1` are the representors facing the network

DPL Components

The DPL solution on the BlueField DPU consists of multiple components working in tandem. These components are outlined below and shown in the following diagram:



DPL Compiler

The NVIDIA DPL Compiler (`dplp4c`) is the central component of the DPL ecosystem . The compiler is typically used from a remote host CPU equipped with Ubuntu 22.04, rather than locally on BlueField. Its core functions include processing and analyzing DPL programs, with the objective of crafting an optimized pipeline compatible with BlueField hardware capabilities. This process culminates in the generation of two critical output files:

- P4info file – This protobuf text file is designed to be target-agnostic, facilitating the integration with P4Runtime controllers by supporting P4Runtime APIs (version 1.3).
- Device configuration file – This binary *.dplconfig file is tailored specifically to the BlueField target device. It contains essential data structures that bridge the gap between P4Runtime (P4RT) commands and DOCA HWS driver objects, ensuring seamless translation and execution of DPL programs on BlueField hardware.

An optional debug information file is produced that is used by the DPL debug tools. For detailed specifications on the P4 language, please refer to the P4-16 v 1.2.4 documentation.

P4Runtime Controller

The P4Runtime Controller is a dynamic management tool designed for networks utilizing P4-programmable devices, enabling network operators to configure, control, and monitor these devices in real-time. This user-owned component must comply with the P4Runtime Specification 1.3.0, ensuring standardized communication and functionality across different network environments. The controller provides the ability to dynamically update packet processing rules, deploy custom network functions, and adapt to changing network conditions.

DPL Nspect Client

DPL Nspect is a dedicated debugging tool tailored for diagnosing and troubleshooting DPL applications on BlueField devices. This standalone utility assists developers in understanding how their DPL program's concepts are implemented on actual hardware. It simplifies various debugging tasks, such as examining table contents and analyzing debug packets to trace issues back to their source. DPL Nspect can be installed on any host machine that has network access to the BlueField's management IP interface, making it a versatile and essential tool for developers working with DPL-programmed BlueField devices.

DPL Runtime Service

The DPL Runtime service process fulfills various roles within BlueField's system:

- Manages the gRPC server and protobuf messages from the P4Runtime controller
- Processes compiler-generated P4info files and binary artifacts
- Responds to controller requests and provides BlueField status updates
- Handles P4 packet-in and packet-out data streams

- Bridges platform-independent P4Runtime APIs with platform-dependent functions using DOCA HWS Driver APIs
- Loads compiler binary artifacts to realize the DPL pipeline in BlueField hardware
- Maintains a database resource mapping between P4Runtime and DOCA HWS Driver APIs along with DPL parse graph information
- Manages the gRPC server for the DPL developer tools

P4Runtime Server

The P4Runtime Server enables communication between the controller(s) and the data plane, allowing for dynamic updates to packet processing rules, querying device configurations, and managing P4 entities declared in the P4Info metadata. The P4 Runtime Server is implemented as a gRPC server, which binds an implementation of auto-generated client and server stubs based on the p4runtime.proto Protobuf file. It listens on TCP port 9559 by default and allows for program-independent control of P4 targets, ensuring a target-independent and protocol-independent approach to runtime control. The software component responsible for implementing the server functionality on BlueField is the DPL Runtime service. The server is the northbound component that processes the P4 Runtime messages by deserializing the messages and dispatching them to the dpl_rtd for execution.

DPL Nspect Server

The DPL Nspect server is an internal component of the DPL Runtime Service, and is provides services to the DPL Nspect and DPL Debugger tools. The server implements a dedicated gRPC/protobuf-based server for the client to receive monitoring and debug information about the running DPL program.

DOCA HWS Driver

The DOCA HWS Driver API provides the essential building blocks for creating the packet processing pipeline in hardware. It provides APIs for constructing flow tables with match criteria, actions, and flow control logic. The compiler utilizes these APIs to build user-defined pipelines efficiently. The DOCA HWS driver serves as the abstraction layer for programmable frameworks such as the DPL compiler and the underlying firmware and BlueField hardware.

eSwitch Management

As a programmability solution, it is the developer's choice as to how the eSwitch administrator manages bridges, virtual ports, interfaces and global pipeline configuration. While Open vSwitch (OVS), a popular open-source virtual switch implementation (e.g. DOCA OVS) provides a flexible, multi-layer virtual networking switch, the user can configure interfaces via DPDK or DOCA as required by the application.

cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.
