DOCA Pipeline Language Model

Table of contents

This section outlines the DOCA Pipeline Language (DPL) services approach to packet processing programmability for the NVIDIA® BlueField®. It introduces the concept of a software development model based on a programming language, supported by a set of DOCA services. For in-depth details see DOCA Pipeline Language Services Guide section.

Introduction

DPL is derived from the P4-16 language specification. P4 is an open-source, domain-specific programming language (DSL) tailored for the programming and customization of network data planes. It is designed to offer a high-level abstraction for programmable packet processing devices, enabling the addition, modification, and extension of networking functionalities. For devices with fixed functions, P4 often serves as a documentation tool to comprehensively describe the data plane's fixed functional blocks.

Key Features of P4

- High-level abstraction – Simplifies the programming of complex network data planes by providing a clear and concise syntax
- Programmable packet processing – Facilitates the customization of how packets are processed and managed within the network
- Documentation of fixed functions – Offers a standardized method for documenting the fixed functional blocks of network devices

Role of the Compiler

The P4 compiler, known as p4c, plays a crucial role in the P4 ecosystem. It automatically generates the data plane program and a corresponding control plane interface. This interface acts as a contract, ensuring seamless communication and coordination between the data plane and the control plane. Benefits include:

- Automatic generation – streamlines the development process by automatically generating essential components and optimizing resource usage
- Custom pipeline behavior – enables developers to extend data plane functionality with tailored pipeline behaviors
- Dynamically loadable pipelines – developers can create and load different or updated pipelines, without needing to build and deploy an application from the ground up.

- Control plane integration – facilitates a matching control plane description via an open source API to manage the customized pipeline effectively

Focus on NVIDIA's DOCA Pipeline Language

The remainder of this document is dedicated to NVIDIA's implementation of the DOCA Pipeline Language (DPL). While its syntactic origins are based on P4-16, specific pipeline semantics follow the NVIDIA's DPU pipeline architecture. For example, P4 semantics imply a staged pipeline, with a feed-forward RMT type architecture. The NVIDIA DPU architecture is a run to completion dRMT architecture, with greater flexibility and capabilities.

DPL Highlights

The DPL programming model is a different paradigm from other constructs such as SDKs, APIs, libraries, drivers or utilities. DPL is a specialized programming language with a runtime system, designed to enable rapid development, testing and deployment of packet processing pipelines. It is provided as a out of the box, build your own solution under DOCA Services.

- DPL Services – provides a system level solution with a compiler, runtime agent and debugging tools, enabling rapid programming of the DPU pipeline

- Tailored for NVIDIA devices – specifically designed and optimized for programming network data planes on NVIDIA's hardware

- Advanced networking functionality – leverages the DPL language to enhance and extend networking capabilities on NVIDIA devices

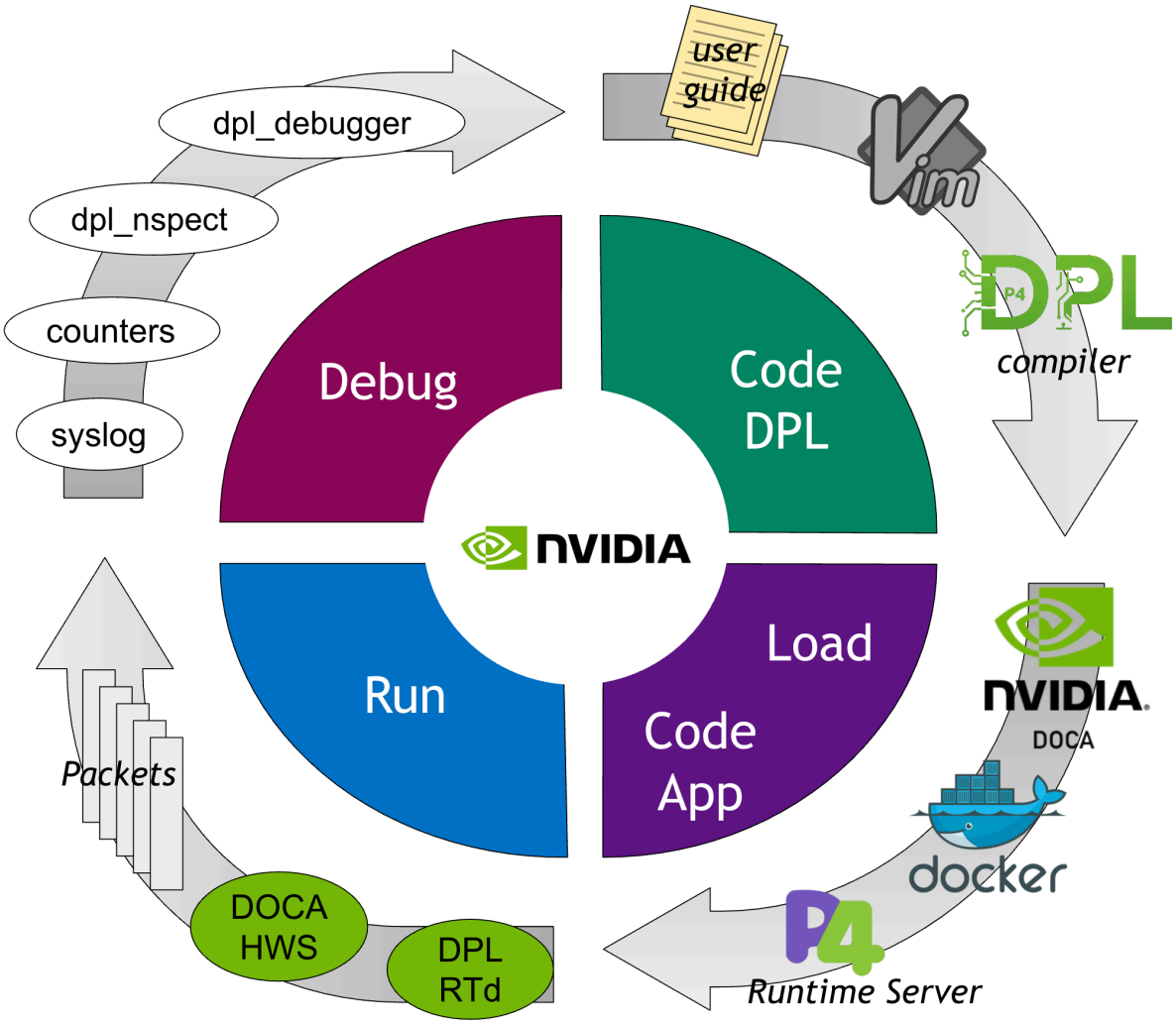
- Comprehensive documentation – provides detailed descriptions of Bluefield fixed functional blocks within the DPU data plane

The DPL programming guide serves as a comprehensive resource for developers looking to leverage the power of DPL for programming network data planes. By utilizing the DPL p4c compiler and the P4-16 specification, developers can enhance the functionality and efficiency of network devices, ensuring they meet the evolving demands of modern network infrastructures.

Prerequisites

The reader should be familiar with the basic concepts of P4 and DPL. Language specifications, runtime APIs and tutorials can be found at https://github.com/p4lang. The DPL compiler can be run on any Linux OS that supports Docker. The development components include:

- Host computer with Ubuntu 22.04 or greater and Docker, needed to install the DOCA dev container
- Server with root/hypervisor access to install the DPL Runtime Service package
- One or more BlueField-3 devices installed in the server



The diagram above shows the suggested workflow:

1. Coding

1. The DPL developer, utilizing the programming guide and sample applications, develops their DPL program remotely.

2. dplp4c is used to compile the DPL program, until it successfully produces a binary.

2. Loading

    1. The binary output is transferred to the BlueField system. Utilizing the P4Runtime API (e.g. via either the open source or a proprietary P4Runtime controller), the pipeline is sent from the remote machine to the DPL Service running on the BlueField DPU.

    2. User checks for P4Runtime error messages.

3. Running

    1. User checks the logs for any DPL Service error messages.

    2. Run the dpl_nspect tool to see that the P4 tables and the entries present in the hardware.

    3. Run the dpl_pipeline debugger tool to deep dive into the stages in the pipeline that packet processing occurred, and observe the state of the packet and its metadata.

This process is repeated until the DPL application is fully complete and verified by the developer.

Understanding the DPL Programming Model

P4, and hence DPL, is a domain-specific language designed for programming network data planes. It allows for a high degree of customization in packet processing, making it a powerful tool for developers. However, it's important to note that P4 programs (NVIDIA DPL and other vendor P4) are generally **not** compatible across different architectures but are instead expected to be compatible within the same target architecture family.

The BlueField programmable pipeline is a hybrid model that leverages the native functionality of BlueField's hardware. The pipeline consists of several key stages:

Parsing

The native parser is a foundational element of BlueField's packet processing pipeline. As the initial pipeline stage, it is tasked with identifying and parsing a sequence of packet

headers, progressing from one to the next until the entire stack is parsed. BlueField's hybrid architecture includes predefined protocol headers and standard transitions based on the "next header" type field, following IETF standards. Unique to DPL is the ability to reparse packets on demand at any stage in the pipeline, eliminating the need for packet reinjection and a final deparser stage.

Flex Parsing

Flex parsing introduces the ability to integrate custom protocol headers into the standard hardware packet parsing engine. This functionality is segmented into four primary components:

- Flex Arc In: Determines the transition from a native header to a Flex header.
- Flex Header: Defines the configurable header's characteristics, such as header length and the position of the next protocol header.
- Flex Sampler: Specifies which Flex sampler should extract bytes from the hardware, e.g., from header field data used in an `apply` block of a control or a field referenced in a P4 table key.
- Flex Arc Out: Sets the transition from the Flex header back to a native header or to another Flex header.

The DPL compiler is responsible for generating the flex parsing components based on the user's specifications for the parse node and transitions.

Operational Mode

The DPL parser functions in a hybrid mode, with a default native parser. This configuration enables the compiler to utilize symbols associated with native headers and fields within DPL constructs. The compiler also adjusts the native graph using flex parser primitives as needed. The flex parse graph is composed of nodes, arcs (transitions), and samplers, with nodes being either flex or fixed (native). The compiler generates the required primitives for flex parsing, which includes creating and destroying flex nodes, managing arcs, and creating samplers for flex node fields, eliminating the need to redefine and reimplement standard IETF protocols and headers.

Programmable Flow Tables

Following the parser is the programmable Match-Action part of BlueField's pipeline. This stage is where packet processing decisions are made based on predefined rules and actions, and is often referred to as "Steering". This component allows for flexible and customizable handling of network traffic, enabling efficient data forwarding, filtering, and manipulation. Since DPL tables can be populated by P4Runtime APIs, this documentation also refers to these flow tables as P4 tables.

Key Features

1. Match fields – define criteria for matching incoming packets, such as source/destination MAC addresses, IP addresses, VLAN tags, and protocol headers.

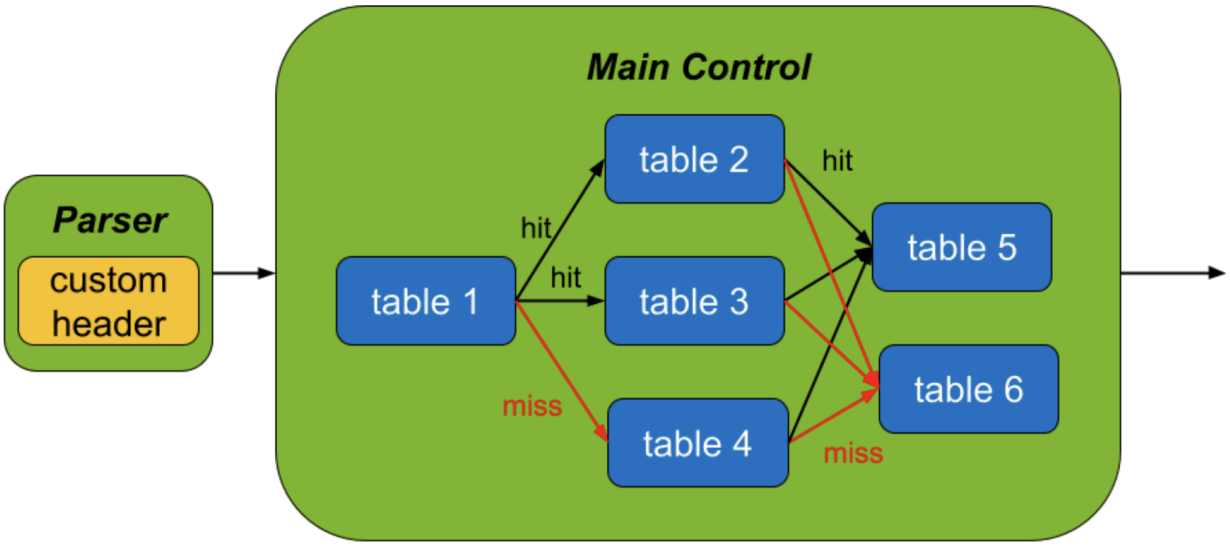
2. Tables – store rules for packet processing based on match criteria, allowing for efficient lookup and decision-making.

3. Actions – specify actions to be taken upon matching specific criteria, such as forwarding packets to a specific port, modifying packet headers, dropping packets, or triggering additional processing.

4. Programmability – enables users to define and modify match-action rules dynamically, adapting to changing network requirements and protocols.

5. Efficient processing – optimizes packet handling by executing actions directly in hardware, reducing latency and improving overall network performance.

6. Integration with P4Runtime – DPL defines custom match-action behaviors, providing a high level of flexibility and control over packet processing logic, that can be managed by a standard P4Runtime SDN controller.



Forwarding Database

The final stage within the embedded switch (eswitch) involves forwarding the packet to its destination port via the forwarding database (FDB). It is a critical component that plays a key role in network communication and packet processing. This database is responsible for storing and managing the MAC addresses of devices connected to the network, enabling efficient and accurate forwarding of data packets within the network infrastructure. By maintaining a record of MAC addresses and their corresponding port locations, the FDB facilitates the proper routing of data packets to their intended destinations.

NVIDIA BlueField DPU Pipeline Behavior

BlueField's flexibility allows developers to customize the packet processing pipeline to meet specific requirements. By extending the parser and utilizing the Match-Action control block, developers can tailor the pipeline's behavior to implement the custom functionality on BlueField's hardware. The DPL compiler will then take full advantage of BlueField's hardware capabilities, ensuring efficient and effective data plane operations. In particular, the execution model is immediate, and no actions are deferred to the end of the pipeline. Notably, BlueField's TA does not include a deparser control. This is because BlueField reparses the packet after any extern action that rewrites the packet headers, mid pipeline. For example, an encapsulation action will immediately be visible to the next table, and the standard metadata will be updated as a result of the reparse. Operations on user metadata and packet fields are immediately visible.

DPL Services

Instead of a programmer's SDK, or driver level APIs, the DPL solution provides a set of services and tools to address the problem programming a DPU pipeline. See the DOCA Pipeline Language Services Guide for details on how to install, develop applications and deploy them using DPL.