DOCA Pipeline Language Developer Tool

Table of contents

These pages describe the NVIDIA DOCA Pipeline Language (DPL) Developer Tools, which are used to inspect and debug DPL programs loaded onto the NVIDIA® BlueField® DPU.

For details on creating, compiling, and running DPL programs, refer to the DOCA Pipeline Language Services Guide page.

Introduction

Developers working with DPL programs require tools for validation and debugging. NVIDIA provides the following DPL developer tools:

- DPL Compiler – converts DPL source code into a binary blob that can be loaded by the DPL Runtime daemon on the DPU
- P4 Runtime Shell – control plane manager that implements the P4Runtime API
- DPL Nspect – command-line interface (CLI) tool for inspecting the current state of tables, keys, counters, and more
- DPL Debugger – user interface tool for debugging packet flow within a DPL program
- DPL Admin – CLI tool for dynamically modifying configuration items in the DPL Runtime daemon

These tools interact with the DPL Runtime daemon (`dpl_rtd`) running on BlueField.

The following pages outline the usage, commands, and options available for each tool.

Prerequisites

1. DPL Runtime Service
    1. The DPL Runtime Service container must be set up and running. This service manages the DPL Runtime daemon.
    2. `dpl_rtd` must be active on a BlueField configured to operate in DPU mode on the host machine.
2. DPL program compiled for debugging:
    1. Compile the DPL program using the DPL Compiler with the debug option enabled (e.g., `dplp4c.sh -g`).

2. Load the compiled binary into `dpl_rtd` using the P4Runtime controller.

> **Info**
>
> For more, refer to section Compiling DPL Applications.

Setup

The developer tools are included in the DPL Development Container, which is publicly available on NGC.

For installation instructions, refer to DPL Installation Guide.

DPL Nspect

This page describes the DPL Nspect tool, which is used to inspect DPL programs loaded onto the NVIDIA® BlueField® DPU.

Introduction

DPL Nspect is a CLI tool with several key use cases:

- Viewing the dynamic state of DPL program components, including tables, entries, keys, actions, and counters
- Recording a debug session and generating an archive that can be opened in the DPL Debugger
- Dumping the state of a DPL program into an archive for offline analysis

DPL Nspect Execution

Running `dpl_nspect.sh` without arguments displays the basic usage information. For a more detailed usage guide, use the `-h` or `--help` argument. This argument also provides command-specific usage details (e.g., `dpl_nspect.sh [command] -h`).

Connecting to a Host

DPL Nspect requires a connection to a local or remote host where the `dpl_nspect_server` is running. The host address and port can be configured using the environment variable. For example: `DPL_NSPECT_SERVER_ADDRESS=127.0.0.1:9560`.

Command Output

The output format of executed commands varies depending on the command. In some cases, the output can be modified using additional arguments to enable integration with automated scripts.

DPL Nspect Commands and Flags

The `dpl_nspect` offers various commands for interacting with the DPL program and BlueField:

- `system-info` – Displays BlueField system information, including hardware model, OS name and version, and OFED version
- `devices` – Lists all configured devices on BlueField, including their ID, name, virtual devices, and the loaded DPL program name

> **Note**
>
> Virtual devices are software abstractions representing real or emulated hardware components. They provide a standardized interface for applications to interact with BlueField resources, regardless of the underlying physical hardware. For example:
>
> - Virtual network interfaces (VNIs) – Logical representations of physical NICs, allowing multiple virtual networks to share a single physical interface
> - Virtual processing units (VPUs) – Software-defined processing units within BlueField that can be allocated to applications for tasks like packet processing or security offloading
> - Virtual storage devices – Software-defined representations of physical storage, offering flexibility in managing and allocating storage resources

- `counters` – Displays details and values of DPL counters, including the associated table, offset, and value
- `tables` – Lists the P4 tables of DPL programs loaded on BlueField, detailing their keys, actions, counters, and references to the program's source location

- `query` – Lists P4 table entries, showing keys, actions, and the priority of each key set
- `graph` – Generates a DPL pipeline graph using DOT language, which can be rendered with DOT-compatible visualization tools
- `hw-steering` – Displays a hardware steering rules dump, providing a low-level view of hardware entries
- `debug` – starts a debug session

Help Flags

To display details of `dpl_nspect` options usage use any of the following flags:

- `help`, `-h`, `--help` – Displays the help message and exits

Output Control Flags

The following optional flags can be used independently to modify command output format and behavior:

- `-ll`, `--low-level` (default: false) – Displays low-level internal information (DOCA HWS implementation)
- `-ni`, `--non-interactive` (default: false) – Disables interactive output mode. Interactive mode outputs data in chunks similar to the Linux `less` command, making it easier to analyze large outputs. Disabling this mode may be necessary when running `dpl_nspect` commands in automated scripts.
- `-nh`, `--no-hints` – Disables hints for additional information
- `--csv` (default: false) – Outputs command results in CSV format
- `--json` (default: false) – Outputs command results in JSON format
- Flags for the `dpl_nspect counters` command:

    - `--top-packets TOP_PACKETS` (default: false) – Displays the top entries with the highest packet counter values
    - `--top-bytes TOP_BYTES` (default: false) – Displays the top entries with the highest byte counter values
- Flags for the `dpl_nspect graph` counters command:
    - `--url` (default: false) – Encodes the output as a URL that can be directly pasted into a web browser for quick rendering using online visualization tools
    - `--type` – Specifies the type of graph to generate

Filter Control Flags

The following flags allow filtering of data based on specific criteria:

- `-d DEVICE_IDS, --device_ids DEVICE_IDS` (default: None) – Specifies a comma-separated list of device IDs to filter, e.g., `1000,2000`. If not specified, data is retrieved for all devices.
- `--table_ids TABLE_IDS` (default: `[]`) – Filters results by specific table IDs
- `--table_names TABLE_NAMES` (default: `[]`) – Filters results by specific table names
- `--counter-names COUNTER_NAMES` (default: `[]`) – Filters results by specific counter names
- `--indices INDICES` (default: `[]`) – Filters results by specific entry indices

Control Flags for Connecting to BlueField

- `-a ADDRESS`, `--address ADDRESS` – Specifies the `dpl_nspect_server` socket address in the format `[IPv4/IPv6][:port]`.

> **Info**
>
> For IPv6 addresses, if a port is specified, the address must be enclosed in square brackets. Examples:
>
> - `ipv6:[2607:f8b0:400e:c00::ef]:443`
> - `ipv6:[::]:1234`

- This parameter is mandatory, unless the environment variable `DPL_NSPECT_SERVER_ADDRESS` is set with the address and port. For example:

```
DPL_NSPECT_SERVER_ADDRESS=127.0.0.1:9560
```

- `--timeout TIMEOUT` (default: `0`) – Specifies the connectivity timeout (optional)

Misc Flags

- `--version`, `-V` – Displays the program's version number and exits

DPL Nspect Commands

System-info

```
dpl_nspect system-info
```

DESCRIPTION	VALUE
DOCA Version	2.10.0

DESCRIPTION	VALUE
Debug Schema Version	Device ID: 1000, doctype: `/dplp4c/devtools/base/v6, target_doctype: /dplp4c/devtools/doca/v1`
Linux OS version	Ubuntu 22.04 LTS
OFED Version	`MLNX_OFED_LINUX-24.01-0.1.7.0`
DPL Runtime Service version	Not Found
RDMA Core Version	`99999P4.2410mlnx54-1.P4.109.gec00b3c78ef6`

Devices

```
dpl_nspect devices
```

ID	NAME	PCI ADDRESS	PORT ID	NAME	NUMBER	DPL PROGRAM	FW VERSION	HW MODEL
1000	br9	0000:65:00.0	0	enp101s0f0np0	50	alpha2_d2	32.38.0588	BlueField-3 integrated ConnectX-7 network controller
			4	eth3	5			
			1	eth0	2			
			2	eth1	3			
			3	eth2	4			

Counters

```
dpl_nspect counters
```

DEVICE: 1000, NV_COUNTER: alpha2_drop2.decap.decap_counter, SCOPE: alpha2_drop2, TYPE: Pkts/Bts, SIZE: 4, IN USE: 3			
IDX	P4	ENTRY OFFSET	VALUE
0	`decap_table`	1	(2, 140)
1	`decap_table`	2	(0, 0)
2	`decap_table`	0	(0, 0)

DEVICE: 1000, NV_COUNTER: alpha2_drop2.encap.encap_counter, SCOPE: alpha2_drop2, TYPE: Pkts/Bts, SIZE: 4, IN USE: 3			
IDX	P4	ENTRY OFFSET	VALUE
0	`encap_table`	0	(0, 0)
1	`encap_table`	2	(0, 0)
2	`encap_table`	1	(0, 0)

DEVICE: 1000, NV_DIRECT_COUNTER: alpha2_drop2.gtp.gtp_counter, SCOPE: alpha2_drop2, TYPE: Pkts/Bts, SIZE: 128, IN USE: 3			
IDX	P4	ENTRY OFFSET	VALUE
0	`gtp_table`	2	(0, 0)
1	`gtp_table`	1	(0, 0)
2	`gtp_table`	0	(0, 0)

The value is in the following format: `(number of packets, number of bytes)`

Tables

```
dpl_nspect tables
```

DEVICE: 1000						
ID	TABLE	KEYS	ACTIONS	COUNTER	ENTRIES	SOURCE
44963766	alpha2_drop2.decap.decap_table	std_meta.ingress_port headers.gre.$valid$ headers.vxlan.$valid$	alpha2_drop2.decap.send_to_port alpha2_drop2.decap.decap_gre alpha2_drop2.decap.decap_vxlan NoAction		4	alpha2_d2.p4:235
41521446	alpha2_drop2.encap.encap_table	std_meta.ingress_port headers.ipv4.src_addr	alpha2_drop2.encap.send_to_port alpha2_drop2.encap.encap_gre alpha2_drop2.encap.encap_vxlan NoAction		4	alpha2_d2.p4:173
34318161	alpha2_drop2.gtp.gtp_table	std_meta.ingress_port headers.gtpv1.teid	alpha2_drop2.gtp.send_to_port alpha2_drop2.gtp.drop	alpha2_drop2.gtp.gtp_counter	3	alpha2_d2.p4:102

Query

```
dpl_nspect query
```

DEVICE: 1000, TABLE: alpha2_drop2.gtp.gtp_table, ID: 34318161, SOURCE: alpha2_d2.p4:102					
PRIO	P4 KEY	VALUE	MASK	TYPE	P4 ACTION
0	`std_meta.ingress_port` `headers.gtpv1.teid`	0x0 0x1		exact exact	`alpha2_drop2.gtp.send_to_port(` `port=0x1)`
0	`std_meta.ingress_port` `headers.gtpv1.teid`	0x1 0x1		exact exact	`alpha2_drop2.gtp.send_to_port(` `port=0x0)`
0	`ALWAYS_HIT`				`alpha2_drop2.gtp.drop()`

DEVICE: 1000, TABLE: alpha2_drop2.encap.encap_table, ID: 41521446, SOURCE: alpha2_d2.p4:173					
PRIO	P4 KEY	VALUE	MASK	TYPE	P4 ACTION
3	`std_meta.ingress_port` `headers.ipv4.src_addr`	0x0 0x10000000	0xff000000	exact ternary	`alpha2_drop2.encap.send_to_port(` `port=0x4)`
2	`std_meta.ingress_port` `headers.ipv4.src_addr`	0x0 0x20000000	0xff000000	exact ternary	`alpha2_drop2.encap.encap_gre(` `port=0x2,` `src_mac=0x112233445566,` `dst_mac=0x10203040506,` `sip=0x1010101,` `dip=0x2020202,` `key=0xabcd)`
1	`std_meta.ingress_port`	0x0	0xff000000	exact	`alpha2_drop2.encap.encap_vxlan(`

DEVICE: 1000, TABLE: alpha2_drop2.encap.encap_table, ID: 41521446, SOURCE: alpha2_d2.p4:173					
	`headers.ipv4.src_addr`	0x20000000		ternary	`port=0x3,` `src_mac=0x112233445566,` `dst_mac=0x10203040506,` `sip=0x1010101,` `dip=0x2020202,` `vni=0xabcd)`
0	`ALWAYS_HIT`				`NoAction()`

DEVICE: 1000, TABLE: alpha2_drop2.decap.decap_table, ID: 44963766, SOURCE: alpha2_d2.p4:235					
PRIO	P4 KEY	VALUE	MASK	TYPE	P4 ACTION
0	`std_meta.ingress_port` `headers.gre.$valid$` `headers.vxlan.$valid$`	0x3 0x0 0x1		exact exact exact	`alpha2_drop2.decap.decap_vxlan(` `port=0x0,` `src_mac=0xaabbccddeeff,` `dst_mac=0x112233445566)`
0	`std_meta.ingress_port` `headers.gre.$valid$` `headers.vxlan.$valid$`	0x4 0x0 0x0		exact exact exact	`alpha2_drop2.decap.send_to_port(` `port=0x0)`
0	`std_meta.ingress_port` `headers.gre.$valid$` `headers.vxlan.$valid$`	0x2 0x1 0x0		exact exact exact	`alpha2_drop2.decap.decap_gre(` `port=0x0)`
0	`ALWAYS_HIT`				`NoAction()`

Different Output Option Flags

- `--csv`

```
dpl_nspect --csv tables
```

```
"DEVICE: 1000"
"ID","TABLE","KEYS","ACTIONS","COUNTER","ENTRIES","SOURCE"
"44963766","alpha2_drop2.decap.decap_table","std_meta.ingress_port
headers.gre.$valid$
headers.vxlan.$valid$","alpha2_drop2.decap.send_to_port
alpha2_drop2.decap.decap_gre
alpha2_drop2.decap.decap_vxlan
NoAction","","4","alpha2_d2.p4:235"
"41521446","alpha2_drop2.encap.encap_table","std_meta.ingress_port
headers.ipv4.src_addr","alpha2_drop2.encap.send_to_port
alpha2_drop2.encap.encap_gre
alpha2_drop2.encap.encap_vxlan
NoAction","","4","alpha2_d2.p4:173"
"34318161","alpha2_drop2.gtp.gtp_table","std_meta.ingress_port
headers.gtpv1.teid","alpha2_drop2.gtp.send_to_port
alpha2_drop2.gtp.drop","alpha2_drop2.gtp.gtp_counter","3","alpha2_d2
```

- `--json`

```
dpl_nspect --json tables
```

```
[
    [
        {
            "DEVICE": 1000
        },
        {
```

```
        "ID": 44963766,
        "TABLE": "alpha2_drop2.decap.decap_table",
        "KEYS": [
            "std_meta.ingress_port",
            "headers.gre.$valid$",
            "headers.vxlan.$valid$"
        ],
        "ACTIONS": [
            "alpha2_drop2.decap.send_to_port",
            "alpha2_drop2.decap.decap_gre",
            "alpha2_drop2.decap.decap_vxlan",
            "NoAction"
        ],
        "COUNTER": "",
        "ENTRIES": 4,
        "SOURCE": {
            "filename": "/tmp/p4src/alpha2_d2.p4",
            "line": 235,
            "column": 11
        }
    },
    {
        "ID": 41521446,
        "TABLE": "alpha2_drop2.encap.encap_table",
        "KEYS": [
            "std_meta.ingress_port",
            "headers.ipv4.src_addr"
        ],
        "ACTIONS": [
            "alpha2_drop2.encap.send_to_port",
            "alpha2_drop2.encap.encap_gre",
            "alpha2_drop2.encap.encap_vxlan",
            "NoAction"
        ],
        "COUNTER": "",
        "ENTRIES": 4,
```

```
                "SOURCE": {
                    "filename": "/tmp/p4src/alpha2_d2.p4",
                    "line": 173,
                    "column": 11
                }
            },
            {
                "ID": 34318161,
                "TABLE": "alpha2_drop2.gtp.gtp_table",
                "KEYS": [
                    "std_meta.ingress_port",
                    "headers.gtpv1.teid"
                ],
                "ACTIONS": [
                    "alpha2_drop2.gtp.send_to_port",
                    "alpha2_drop2.gtp.drop"
                ],
                "COUNTER": "alpha2_drop2.gtp.gtp_counter",
                "ENTRIES": 3,
                "SOURCE": {
                    "filename": "/tmp/p4src/alpha2_d2.p4",
                    "line": 102,
                    "column": 11
                }
            }
        ]
    ]
```


Graph (requires the --low-level flag)

- No extra flags

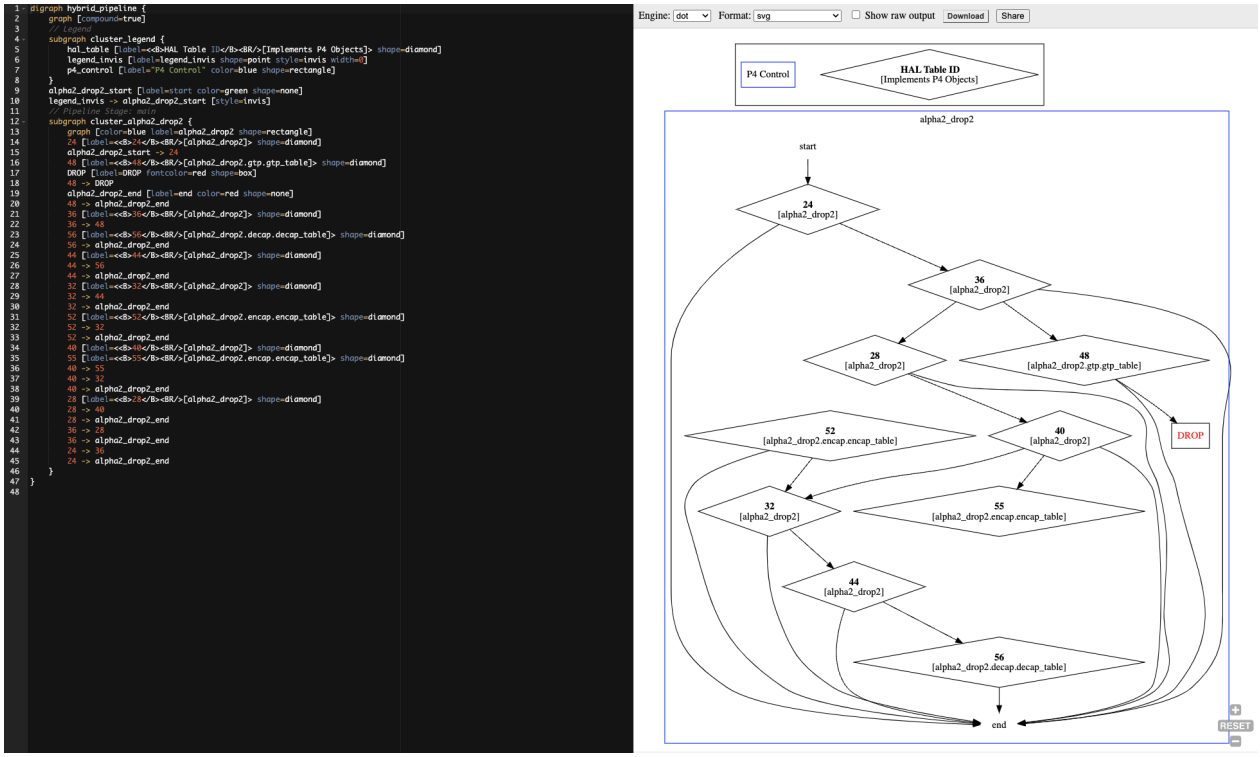
```
dpl_nspect -ll graph --type pipeline_low_level
```

```
Device 1000:
digraph hybrid_pipeline {
        graph [compound=true]
        // Legend
        subgraph cluster_legend {
                hal_table [label=<<B>HAL Table ID</B><BR/>
[Implements P4 Objects]> shape=diamond]
                legend_invis [label=legend_invis shape=point
style=invis width=0]
                p4_control [label="P4 Control" color=blue
shape=rectangle]
        }
        alpha2_drop2_start [label=start color=green
shape=none]
        legend_invis -> alpha2_drop2_start [style=invis]
        // Pipeline Stage: main
        subgraph cluster_alpha2_drop2 {
                graph [color=blue label=alpha2_drop2
shape=rectangle]
                24 [label=<<B>24</B><BR/>[alpha2_drop2]>
shape=diamond]
                alpha2_drop2_start -> 24
                48 [label=<<B>48</B><BR/>
[alpha2_drop2.gtp.gtp_table]> shape=diamond]
                DROP [label=DROP fontcolor=red shape=box]
                48 -> DROP
                alpha2_drop2_end [label=end color=red
shape=none]
                48 -> alpha2_drop2_end
                36 [label=<<B>36</B><BR/>[alpha2_drop2]>
shape=diamond]
```

```
                36 -> 48
                56 [label=<<B>56</B><BR/>
[alpha2_drop2.decap.decap_table]> shape=diamond]
                56 -> alpha2_drop2_end
                44 [label=<<B>44</B><BR/>[alpha2_drop2]>
shape=diamond]
                44 -> 56
                44 -> alpha2_drop2_end
                32 [label=<<B>32</B><BR/>[alpha2_drop2]>
shape=diamond]
                32 -> 44
                32 -> alpha2_drop2_end
                52 [label=<<B>52</B><BR/>
[alpha2_drop2.encap.encap_table]> shape=diamond]
                52 -> 32
                52 -> alpha2_drop2_end
                40 [label=<<B>40</B><BR/>[alpha2_drop2]>
shape=diamond]
                55 [label=<<B>55</B><BR/>
[alpha2_drop2.encap.encap_table]> shape=diamond]
                40 -> 55
                40 -> 32
                40 -> alpha2_drop2_end
                28 [label=<<B>28</B><BR/>[alpha2_drop2]>
shape=diamond]
                28 -> 40
                28 -> alpha2_drop2_end
                36 -> 28
                36 -> alpha2_drop2_end
                24 -> 36
                24 -> alpha2_drop2_end
        }
}
```

- `--url` – generates an HTTP URL link to the program's control flow graph, in dot format, rendered using https://dreampuf.github.io/GraphvizOnline/.

```
dpl_nspect -ll graph --url --type pipeline_low_level
```



- Generate the program's parser graph in dot format:

```
dpl_nspect -ll graph --type parser
```

```
Device 1000:
// Parser packet_parser
digraph packet_parser {
graph [concentrate=true splines=compound]
// Legend
subgraph cluster_legend {
fixed_state [label="Fixed State" style=filled]
```

```
legend_invis [label=legend_invis shape=point style=invis
width=0]
flex_state [label="Flex State"]
default_transition_note [label="All states have an hidden transition to
accept\nEdges are annotated with their match value." shape=none]
legend_invis -> default_transition_note [style=invis]
}
start [label=start color=green shape=point]
default_transition_note -> start [style=invis]
nv_parse_ethernet [label=ethernet id=nv_parse_ethernet
style=filled]
start -> nv_parse_ethernet
nv_parse_vlan [label=vlan id=nv_parse_vlan style=filled]
nv_parse_ethernet -> nv_parse_vlan [label="0x8100" color=gray
id="nv_parse_ethernet-nv_parse_vlan"]
nv_parse_ipv4 [label=ipv4 id=nv_parse_ipv4 style=filled]
nv_parse_ethernet -> nv_parse_ipv4 [label="0x800" color=gray
id="nv_parse_ethernet-nv_parse_ipv4"]
nv_parse_ipv6 [label=ipv6 id=nv_parse_ipv6 style=filled]
nv_parse_ethernet -> nv_parse_ipv6 [label="0x86dd" color=gray
id="nv_parse_ethernet-nv_parse_ipv6"]
nv_parse_mpls [label=mpls id=nv_parse_mpls style=filled]
nv_parse_ethernet -> nv_parse_mpls [label="0x8847" color=gray
id="nv_parse_ethernet-nv_parse_mpls"]
nv_parse_geneve [label=geneve id=nv_parse_geneve
style=filled]
nv_parse_inner_ipv4 [label=inner_ipv4 id=nv_parse_inner_ipv4
style=filled]
nv_parse_geneve -> nv_parse_inner_ipv4 [label="0x800"
color=gray id="nv_parse_geneve-nv_parse_inner_ipv4"]
nv_parse_inner_ipv6 [label=inner_ipv6 id=nv_parse_inner_ipv6
style=filled]
nv_parse_geneve -> nv_parse_inner_ipv6 [label="0x86dd"
color=gray id="nv_parse_geneve-nv_parse_inner_ipv6"]
nv_parse_inner_mpls [label=inner_mpls id=nv_parse_inner_mpls
style=filled]
```

```
nv_parse_geneve -> nv_parse_inner_mpls [label="0x8847"
color=gray id="nv_parse_geneve-nv_parse_inner_mpls"]
nv_parse_inner_ethernet [label=inner_ethernet
id=nv_parse_inner_ethernet style=filled]
nv_parse_geneve -> nv_parse_inner_ethernet [label="0x6558"
color=gray id="nv_parse_geneve-nv_parse_inner_ethernet"]
nv_parse_gre [label=gre id=nv_parse_gre style=filled]
nv_parse_gre -> nv_parse_inner_ipv4 [label="0x800" color=gray
id="nv_parse_gre-nv_parse_inner_ipv4"]
nv_parse_gre -> nv_parse_inner_ipv6 [label="0x86dd" color=gray
id="nv_parse_gre-nv_parse_inner_ipv6"]
nv_parse_gre -> nv_parse_inner_mpls [label="0x8847" color=gray
id="nv_parse_gre-nv_parse_inner_mpls"]
nv_parse_nvgre_vsid [label=nvgre_vsid id=nv_parse_nvgre_vsid
style=filled]
nv_parse_gre -> nv_parse_nvgre_vsid [label="0x6558" color=gray
id="nv_parse_gre-nv_parse_nvgre_vsid"]
nv_parse_icmp [label=icmp id=nv_parse_icmp style=filled]
nv_parse_icmpv6 [label=icmpv6 id=nv_parse_icmpv6
style=filled]
nv_parse_inner_vlan [label=inner_vlan id=nv_parse_inner_vlan
style=filled]
nv_parse_inner_ethernet -> nv_parse_inner_vlan [label="0x8100"
color=gray id="nv_parse_inner_ethernet-nv_parse_inner_vlan"]
nv_parse_inner_ethernet -> nv_parse_inner_ipv4 [label="0x800"
color=gray id="nv_parse_inner_ethernet-nv_parse_inner_ipv4"]
nv_parse_inner_ethernet -> nv_parse_inner_ipv6 [label="0x86dd"
color=gray id="nv_parse_inner_ethernet-nv_parse_inner_ipv6"]
nv_parse_inner_ethernet -> nv_parse_inner_mpls [label="0x8847"
color=gray id="nv_parse_inner_ethernet-nv_parse_inner_mpls"]
nv_parse_inner_icmp [label=inner_icmp id=nv_parse_inner_icmp
style=filled]
nv_parse_inner_icmpv6 [label=inner_icmpv6
id=nv_parse_inner_icmpv6 style=filled]
nv_parse_inner_ipv4 -> nv_parse_inner_icmp [label="0x1"
color=gray id="nv_parse_inner_ipv4-nv_parse_inner_icmp"]
```

```
nv_parse_inner_tcp [label=inner_tcp id=nv_parse_inner_tcp
style=filled]
nv_parse_inner_ipv4 -> nv_parse_inner_tcp [label="0x6"
color=gray id="nv_parse_inner_ipv4-nv_parse_inner_tcp"]
nv_parse_inner_udp [label=inner_udp id=nv_parse_inner_udp
style=filled]
nv_parse_inner_ipv4 -> nv_parse_inner_udp [label="0x11"
color=gray id="nv_parse_inner_ipv4-nv_parse_inner_udp"]
nv_parse_inner_ipv6 -> nv_parse_inner_icmpv6 [label="0x3a"
color=gray id="nv_parse_inner_ipv6-nv_parse_inner_icmpv6"]
nv_parse_inner_ipv6 -> nv_parse_inner_tcp [label="0x6"
color=gray id="nv_parse_inner_ipv6-nv_parse_inner_tcp"]
nv_parse_inner_ipv6 -> nv_parse_inner_udp [label="0x11"
color=gray id="nv_parse_inner_ipv6-nv_parse_inner_udp"]
nv_parse_inner_mpls1 [label=inner_mpls1
id=nv_parse_inner_mpls1 style=filled]
nv_parse_inner_mpls -> nv_parse_inner_mpls1 [label="0x0"
color=gray id="nv_parse_inner_mpls-nv_parse_inner_mpls1"]
nv_parse_inner_mpls_next_header
[label=nv_parse_inner_mpls_next_header
id=nv_parse_inner_mpls_next_header style=filled]
nv_parse_inner_mpls -> nv_parse_inner_mpls_next_header
[label="0x1" color=gray id="nv_parse_inner_mpls-nv_parse_inner_mpls_next_header"]
nv_parse_inner_mpls2 [label=inner_mpls2
id=nv_parse_inner_mpls2 style=filled]
nv_parse_inner_mpls1 -> nv_parse_inner_mpls2 [label="0x0"
color=gray id="nv_parse_inner_mpls1-nv_parse_inner_mpls2"]
nv_parse_inner_mpls1 -> nv_parse_inner_mpls_next_header
[label="0x1" color=gray id="nv_parse_inner_mpls1-
nv_parse_inner_mpls_next_header"]
nv_parse_inner_mpls3 [label=inner_mpls3
id=nv_parse_inner_mpls3 style=filled]
nv_parse_inner_mpls2 -> nv_parse_inner_mpls3 [label="0x0"
color=gray id="nv_parse_inner_mpls2-nv_parse_inner_mpls3"]
nv_parse_inner_mpls2 -> nv_parse_inner_mpls_next_header
[label="0x1" color=gray id="nv_parse_inner_mpls2-
```

```
nv_parse_inner_mpls_next_header"]
nv_parse_inner_mpls4 [label=inner_mpls4
id=nv_parse_inner_mpls4 style=filled]
nv_parse_inner_mpls3 -> nv_parse_inner_mpls4 [label="0x0"
color=gray id="nv_parse_inner_mpls3-nv_parse_inner_mpls4"]
nv_parse_inner_mpls3 -> nv_parse_inner_mpls_next_header
[label="0x1" color=gray id="nv_parse_inner_mpls3-
nv_parse_inner_mpls_next_header"]
nv_parse_inner_mpls4 -> nv_parse_inner_mpls_next_header
[label="0x1" color=gray id="nv_parse_inner_mpls4-
nv_parse_inner_mpls_next_header"]
nv_parse_inner_mpls_next_header -> nv_parse_inner_ipv4
[label="0x4" color=gray id="nv_parse_inner_mpls_next_header-nv_parse_inner_ipv4"]
nv_parse_inner_mpls_next_header -> nv_parse_inner_ipv6
[label="0x6" color=gray id="nv_parse_inner_mpls_next_header-nv_parse_inner_ipv6"]
nv_parse_inner_vlan -> nv_parse_inner_ipv4 [label="0x800"
color=gray id="nv_parse_inner_vlan-nv_parse_inner_ipv4"]
nv_parse_inner_vlan -> nv_parse_inner_ipv6 [label="0x86dd"
color=gray id="nv_parse_inner_vlan-nv_parse_inner_ipv6"]
nv_parse_inner_vlan -> nv_parse_inner_mpls [label="0x8847"
color=gray id="nv_parse_inner_vlan-nv_parse_inner_mpls"]
nv_parse_ipsec_esp [label=esp id=nv_parse_ipsec_esp
style=filled]
nv_parse_ipv4 -> nv_parse_inner_ipv4 [label="0x4" color=gray
id="nv_parse_ipv4-nv_parse_inner_ipv4"]
nv_parse_ipv4 -> nv_parse_inner_ipv6 [label="0x29" color=gray
id="nv_parse_ipv4-nv_parse_inner_ipv6"]
nv_parse_ipv4 -> nv_parse_icmp [label="0x1" color=gray
id="nv_parse_ipv4-nv_parse_icmp"]
nv_parse_tcp [label=tcp id=nv_parse_tcp style=filled]
nv_parse_ipv4 -> nv_parse_tcp [label="0x6" color=gray
id="nv_parse_ipv4-nv_parse_tcp"]
nv_parse_udp [label=udp id=nv_parse_udp style=filled]
nv_parse_ipv4 -> nv_parse_udp [label="0x11" color=gray
id="nv_parse_ipv4-nv_parse_udp"]
```

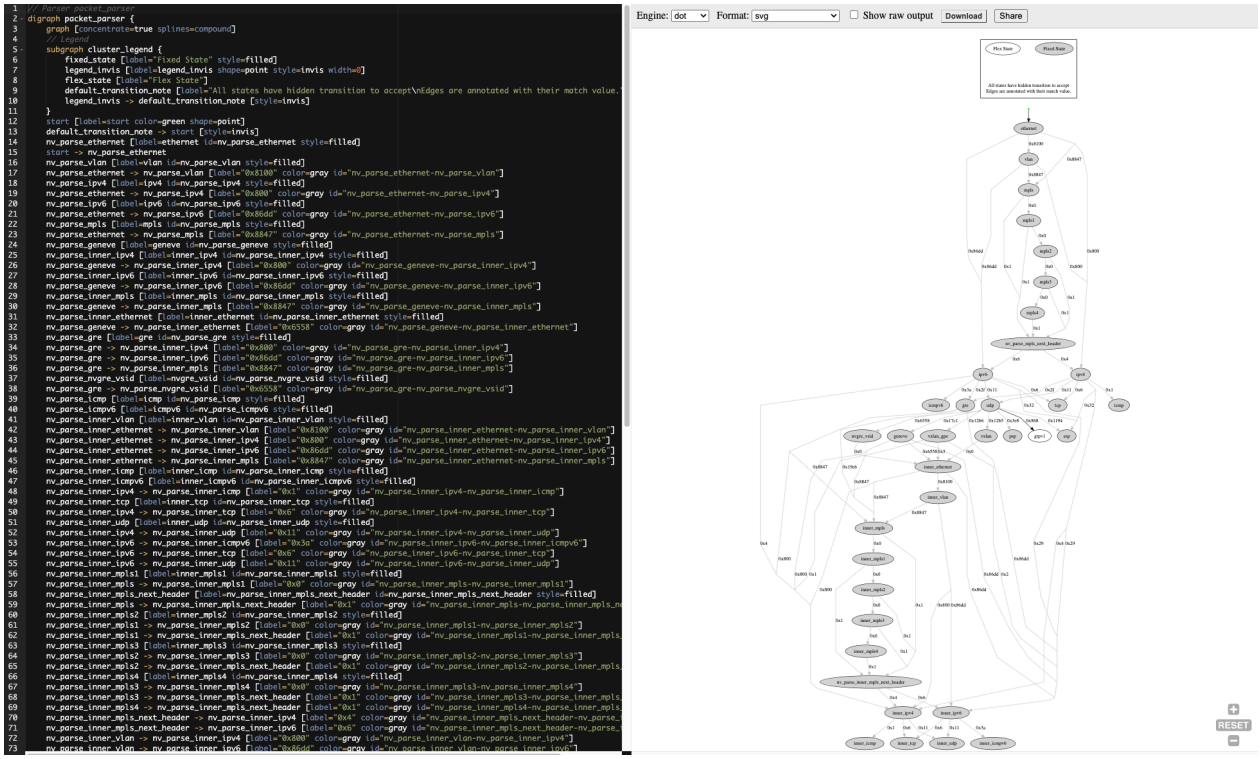
```
nv_parse_ipv4 -> nv_parse_gre [label="0x2f" color=gray
id="nv_parse_ipv4-nv_parse_gre"]
nv_parse_ipv4 -> nv_parse_ipsec_esp [label="0x32" color=gray
id="nv_parse_ipv4-nv_parse_ipsec_esp"]
nv_parse_ipv6 -> nv_parse_inner_ipv4 [label="0x4" color=gray
id="nv_parse_ipv6-nv_parse_inner_ipv4"]
nv_parse_ipv6 -> nv_parse_inner_ipv6 [label="0x29" color=gray
id="nv_parse_ipv6-nv_parse_inner_ipv6"]
nv_parse_ipv6 -> nv_parse_icmpv6 [label="0x3a" color=gray
id="nv_parse_ipv6-nv_parse_icmpv6"]
nv_parse_ipv6 -> nv_parse_tcp [label="0x6" color=gray
id="nv_parse_ipv6-nv_parse_tcp"]
nv_parse_ipv6 -> nv_parse_udp [label="0x11" color=gray
id="nv_parse_ipv6-nv_parse_udp"]
nv_parse_ipv6 -> nv_parse_gre [label="0x2f" color=gray
id="nv_parse_ipv6-nv_parse_gre"]
nv_parse_ipv6 -> nv_parse_ipsec_esp [label="0x32" color=gray
id="nv_parse_ipv6-nv_parse_ipsec_esp"]
nv_parse_mpls1 [label=mpls1 id=nv_parse_mpls1 style=filled]
nv_parse_mpls -> nv_parse_mpls1 [label="0x0" color=gray
id="nv_parse_mpls-nv_parse_mpls1"]
nv_parse_mpls_next_header [label=nv_parse_mpls_next_header
id=nv_parse_mpls_next_header style=filled]
nv_parse_mpls -> nv_parse_mpls_next_header [label="0x1"
color=gray id="nv_parse_mpls-nv_parse_mpls_next_header"]
nv_parse_mpls2 [label=mpls2 id=nv_parse_mpls2 style=filled]
nv_parse_mpls1 -> nv_parse_mpls2 [label="0x0" color=gray
id="nv_parse_mpls1-nv_parse_mpls2"]
nv_parse_mpls1 -> nv_parse_mpls_next_header [label="0x1"
color=gray id="nv_parse_mpls1-nv_parse_mpls_next_header"]
nv_parse_mpls3 [label=mpls3 id=nv_parse_mpls3 style=filled]
nv_parse_mpls2 -> nv_parse_mpls3 [label="0x0" color=gray
id="nv_parse_mpls2-nv_parse_mpls3"]
nv_parse_mpls2 -> nv_parse_mpls_next_header [label="0x1"
color=gray id="nv_parse_mpls2-nv_parse_mpls_next_header"]
nv_parse_mpls4 [label=mpls4 id=nv_parse_mpls4 style=filled]
```

```
nv_parse_mpls3 -> nv_parse_mpls4 [label="0x0" color=gray
id="nv_parse_mpls3-nv_parse_mpls4"]
nv_parse_mpls3 -> nv_parse_mpls_next_header [label="0x1"
color=gray id="nv_parse_mpls3-nv_parse_mpls_next_header"]
nv_parse_mpls4 -> nv_parse_mpls_next_header [label="0x1"
color=gray id="nv_parse_mpls4-nv_parse_mpls_next_header"]
nv_parse_mpls_next_header -> nv_parse_ipv4 [label="0x4"
color=gray id="nv_parse_mpls_next_header-nv_parse_ipv4"]
nv_parse_mpls_next_header -> nv_parse_ipv6 [label="0x6"
color=gray id="nv_parse_mpls_next_header-nv_parse_ipv6"]
nv_parse_nvgre_vsid -> nv_parse_inner_ethernet [label="0x0"
color=gray id="nv_parse_nvgre_vsid-nv_parse_inner_ethernet"]
nv_parse_psp [label=psp id=nv_parse_psp style=filled]
parse_gtp [label=gtpv1 id=parse_gtp style=""]
nv_parse_udp -> parse_gtp [label="0x868" color=black
id="nv_parse_udp-parse_gtp"]
nv_parse_vxlan [label=vxlan id=nv_parse_vxlan style=filled]
nv_parse_udp -> nv_parse_vxlan [label="0x12b5" color=gray
id="nv_parse_udp-nv_parse_vxlan"]
nv_parse_vxlan_gpe [label=vxlan_gpe id=nv_parse_vxlan_gpe
style=filled]
nv_parse_udp -> nv_parse_vxlan_gpe [label="0x12b6" color=gray
id="nv_parse_udp-nv_parse_vxlan_gpe"]
nv_parse_udp -> nv_parse_geneve [label="0x17c1" color=gray
id="nv_parse_udp-nv_parse_geneve"]
nv_parse_udp -> nv_parse_inner_mpls [label="0x19eb" color=gray
id="nv_parse_udp-nv_parse_inner_mpls"]
nv_parse_udp -> nv_parse_ipsec_esp [label="0x1194" color=gray
id="nv_parse_udp-nv_parse_ipsec_esp"]
nv_parse_udp -> nv_parse_psp [label="0x3e8" color=gray
id="nv_parse_udp-nv_parse_psp"]
nv_parse_vlan -> nv_parse_ipv4 [label="0x800" color=gray
id="nv_parse_vlan-nv_parse_ipv4"]
nv_parse_vlan -> nv_parse_ipv6 [label="0x86dd" color=gray
id="nv_parse_vlan-nv_parse_ipv6"]
```

```
nv_parse_vlan -> nv_parse_mpls [label="0x8847" color=gray
id="nv_parse_vlan-nv_parse_mpls"]
nv_parse_vxlan -> nv_parse_inner_ethernet [label="0x0"
color=gray id="nv_parse_vxlan-nv_parse_inner_ethernet"]
nv_parse_vxlan_gpe -> nv_parse_inner_ipv4 [label="0x1"
color=gray id="nv_parse_vxlan_gpe-nv_parse_inner_ipv4"]
nv_parse_vxlan_gpe -> nv_parse_inner_ipv6 [label="0x2"
color=gray id="nv_parse_vxlan_gpe-nv_parse_inner_ipv6"]
nv_parse_vxlan_gpe -> nv_parse_inner_ethernet [label="0x3"
color=gray id="nv_parse_vxlan_gpe-nv_parse_inner_ethernet"]
}
```

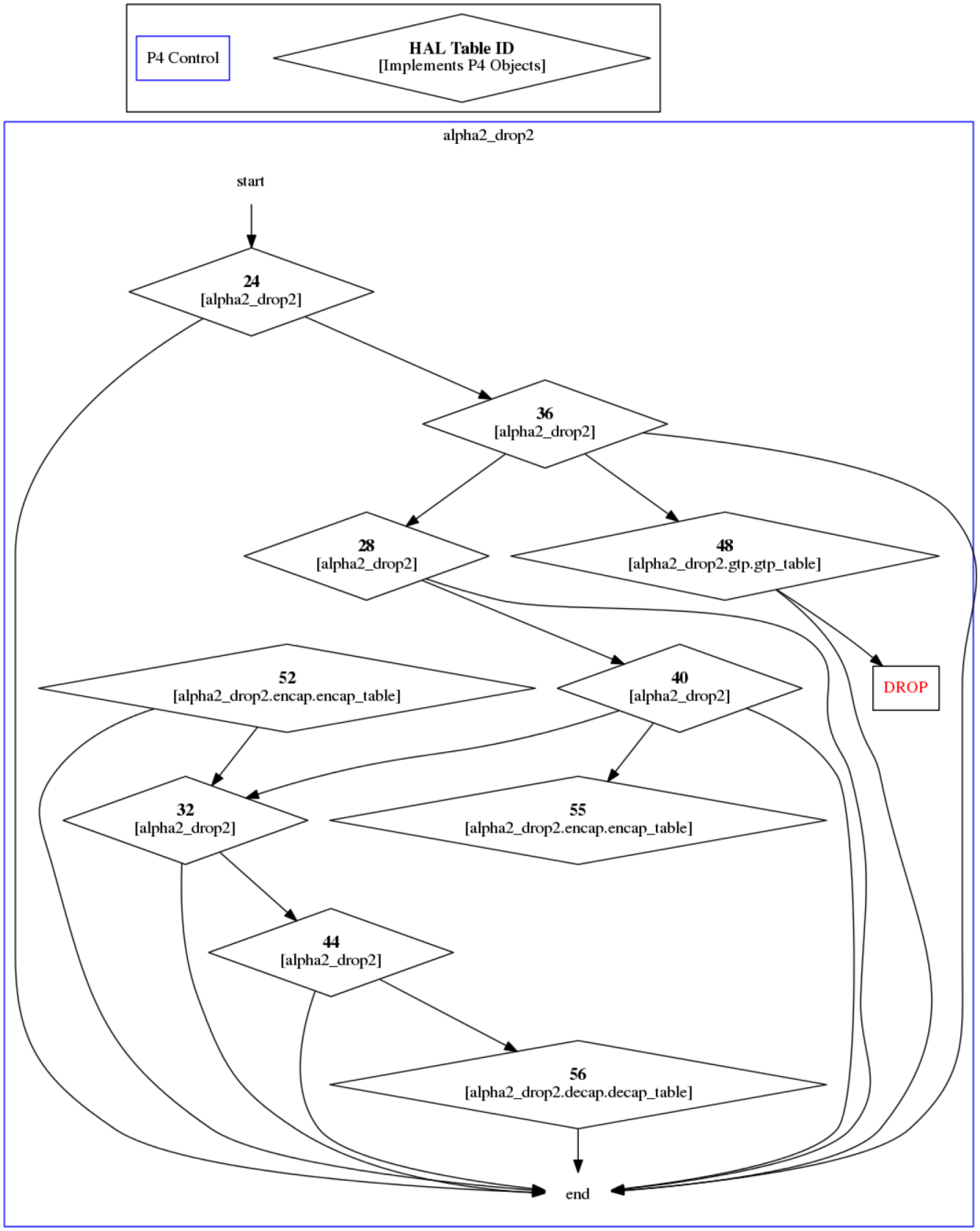
- `--url --type parser`

```
dpl_nspect -ll graph --url --type parser
```



Illustration

It is possible to render the graph to get the following picture:



Commands and Flags Summary

Some command flags are universally applicable to all commands, but others are applicable to only specific commands.

The following table outlines commands and applicable flags:

	system-info	devices	counters	tables	query	graph	hw-steering	debug
--address	✔	✔	✔	✔	✔	✔	✔	✔
--time-out	✔	✔	✔	✔	✔	✔	✔	✔
--non-interactive	✔	✔	✔	✔	✔	✔	✔	✔
--no-hints	✔	✔	✔	✔	✔	✔	✔	✔
--csv	✔	✔	✔	✔	✔	✔	✔	✔
--json	✔	✔	✔	✔	✔	✔	✔	✔
--device-ids	✔	✔	✔	✔	✔	✔	✔	✔
--low-level	✔	✔	✔	✔	✔	✔	✔	✔
--table-ids			✔	✔	✔			
--table_names			✔	✔	✔			
--counter-names			✔					
--indices			✔					
--top-packets			✔					
--top-bytes			✔					
--url						✔		
--type						✔		
--output-file								✔
--debug-device-id								✔

DPL Debugger

This page describes the DPL Debugger tools, which are used to debug pipeline packets processed by DPL programs on the NVIDIA® BlueField® DPU.

Introduction

The DPL Debugger is a GUI application that can run a live debug session or open a pre-recorded debug session file captured using the `dpl_nspect debug` command.

A debug session visualizes the flow of network packets through a DPL program's pipeline, allowing developers to trace each packet's path in detail. For more information, refer to the documentation on the nv_send_debug_pkt extern function.

A debug session visualizes the flow of network packets through a DPL program's pipeline, allowing developers to trace packet details throughout its course. For more information, refer to the documentation on the nv_send_debug_pkt extern function.

> **Note**
>
> The current version of the debugger only displays debug packets received by the DPU (Rx path) or packets arriving from the primary wire port (P0) in the network-to-host direction.

Executing the DPL Debugger

Running `dpl_debugger.sh` without any arguments launches the GUI application.

To open and debug a pre-recorded session specify the session archive when executing the program:

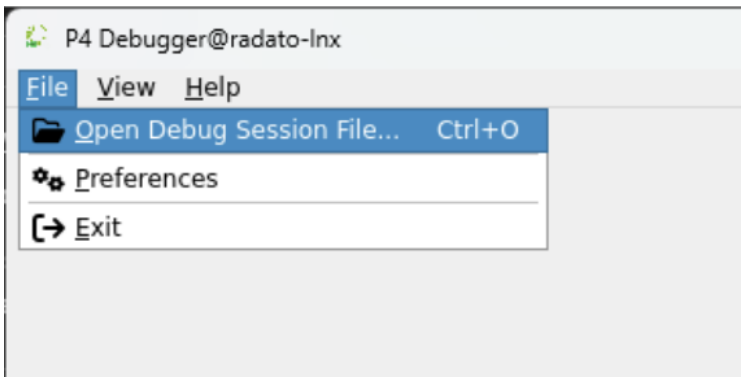
```
dpl_debugger.sh my-debug-archive.tar.gz
```

Using the DPL Debugger

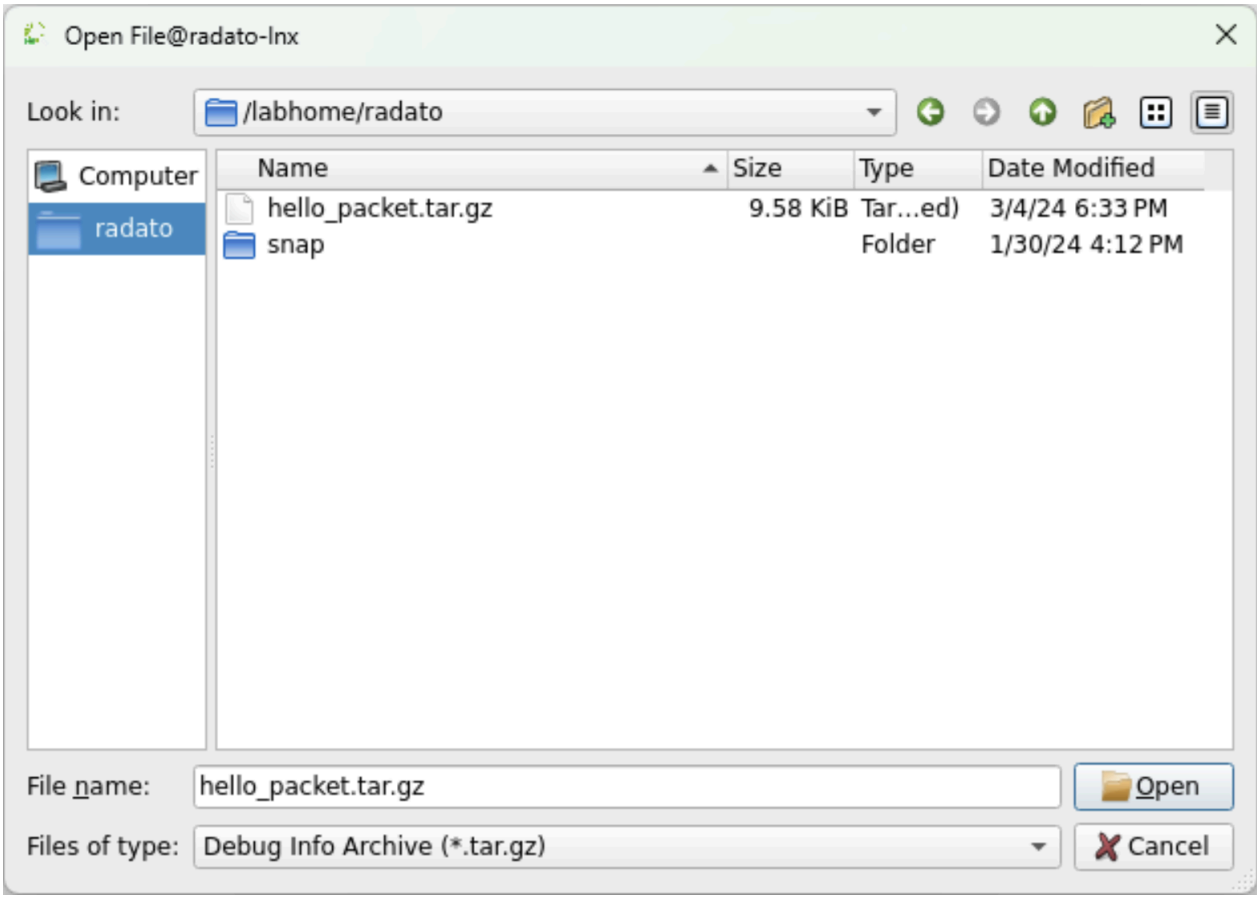
Open Debug Session File

To open a pre-recorded debug session in the DPL Debugger GUI:

1. Select the **File** → **Open Debug Session File**.



2. Choose the appropriate debug session file (`hello_packet.tar.gz` in this example).



Info

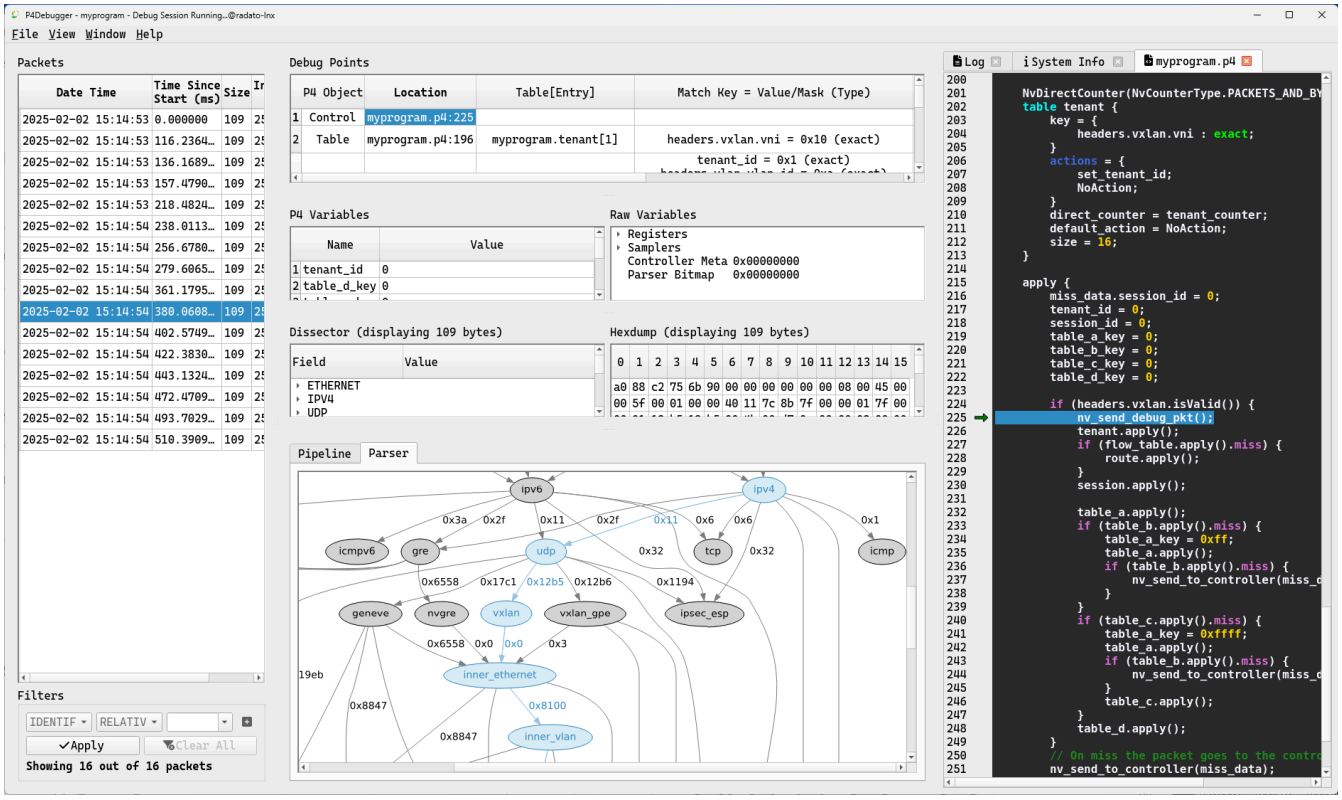
The `tar.gz` archive should be the output of a recorded debug session created using the `dpl_nspect debug` command.

Packets List

The left pane of the DPL Debugger displays the **Packets** list, which includes the following fields:

- Date Time – The timestamp when the packet entered the DPL pipeline
- Time Since Start – The elapsed time since the first packet entered the DPL pipeline
- Size – The packet size

- Ingress Port – The port through which the packet entered the pipeline
- Protocol – The network protocol associated with the packet

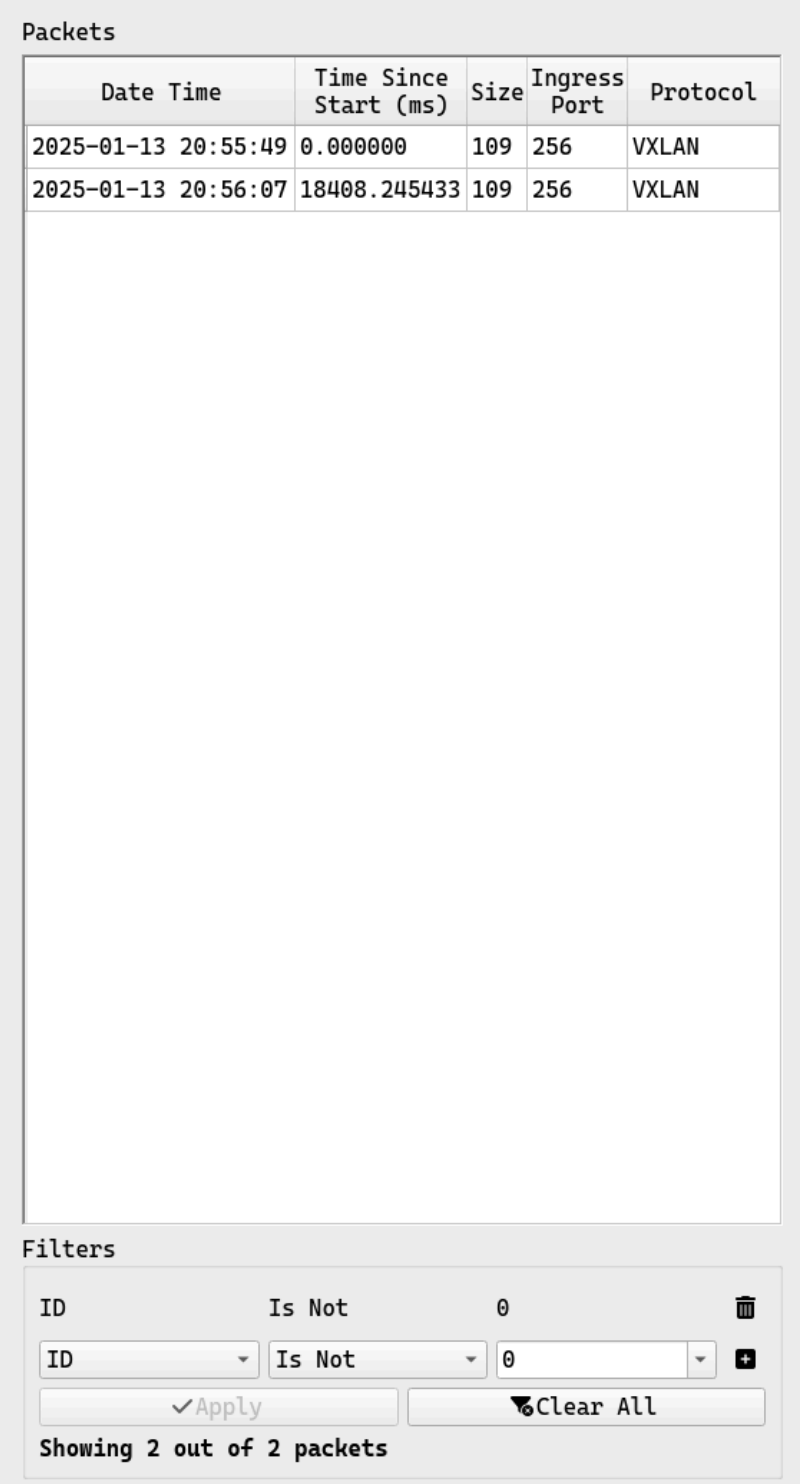


Filter

The **Packets** list can be filtered using the fields located at the bottom left of the GUI.

To apply a filter:

1. Enter a condition (e.g., "packet ID is not 0").
2. Click the **+** button to add the filter.
3. Click the **Apply** button to apply the filter.



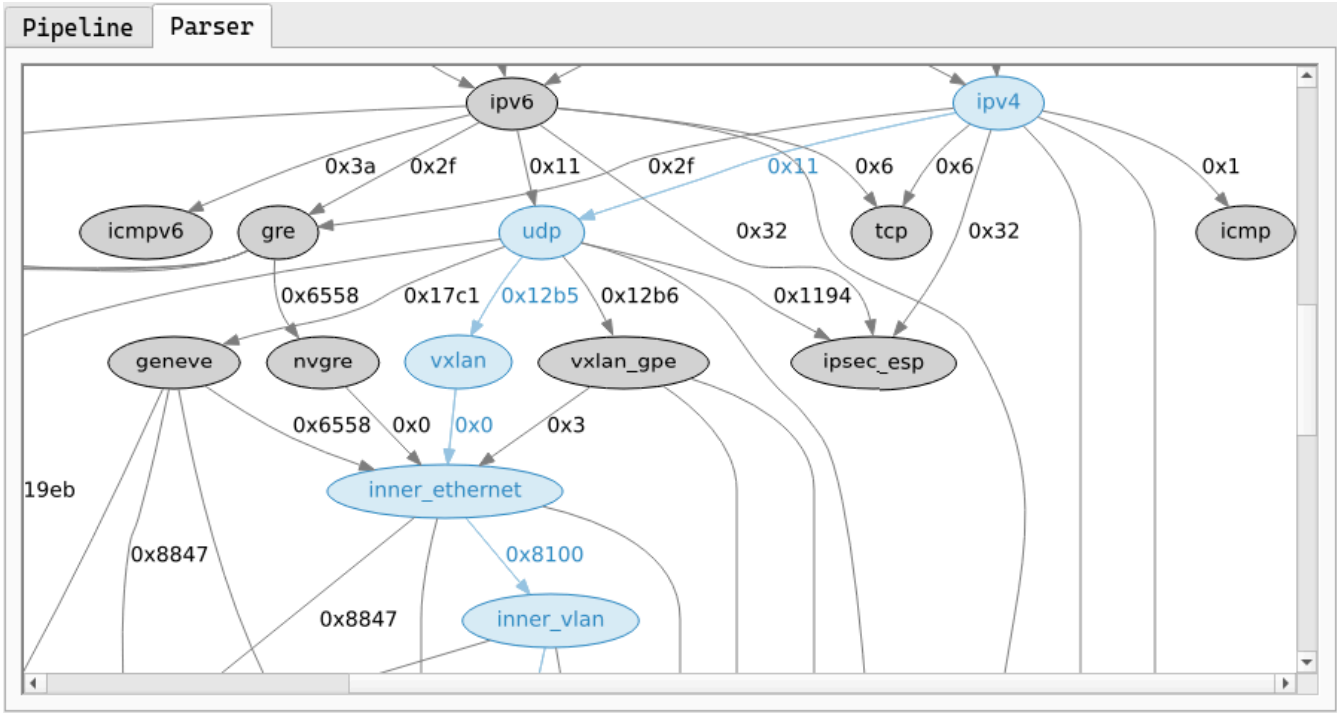
To reset the **Packets** list to its original state:

1. Click the **Clear All** button.
2. Click **Apply**.

Parser Graph

The **Parser** graph, located at the bottom of the GUI, outlines the possible routes a packet can take within the DPL pipeline.

- Blue nodes (`inner_ethernet` in the example) indicate the nodes the packet has traversed.
- The graph consists of two types of nodes:
  - Flex nodes – Defined by the DPL programmer
  - Static (fixed) nodes – Built-in components of the hardware



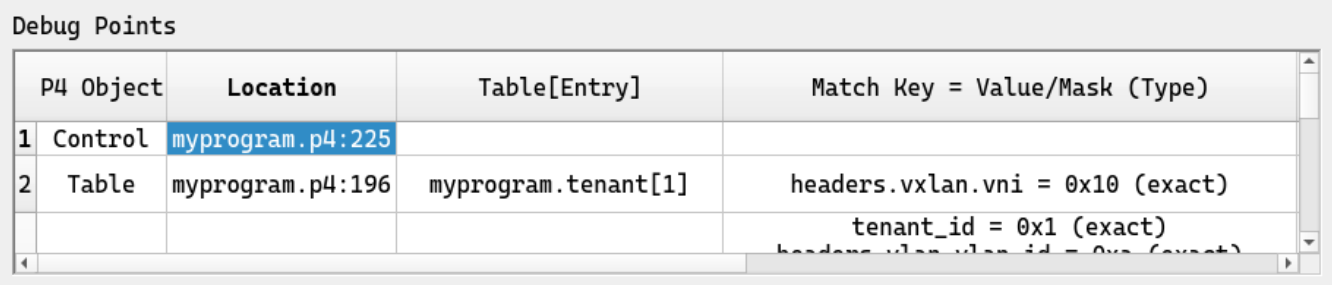
This visualization helps developers understand the packet's path through the parsing stages.

Debug Points

Selecting a packet from the **Packets** List displays its pipeline stages in the Debug Points view at the top of the GUI:

- Each row corresponds to a specific source location in the DPL program code

- The table cells are interactive—clicking a cell highlights the relevant source location, when applicable



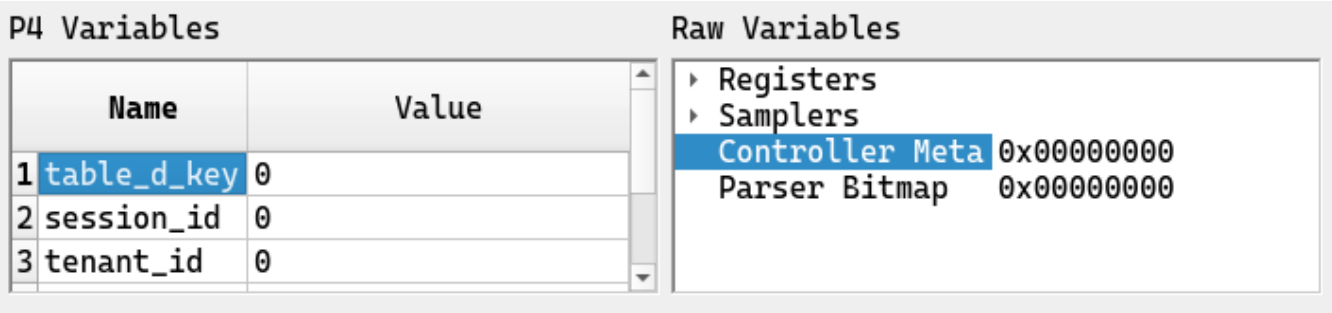
This allows developers to trace a packet's processing path and correlate it with the corresponding code sections.

Variables

Selecting a pipeline stage (either a single cell or an entire row) displays variable information in two tabs:

- P4 Variables – Shows DPL variables defined in the program and their values at the selected pipeline stage.

- Raw Variables – Provides a low-level view of internal variables, such as registers, samplers, and other hardware-related data.



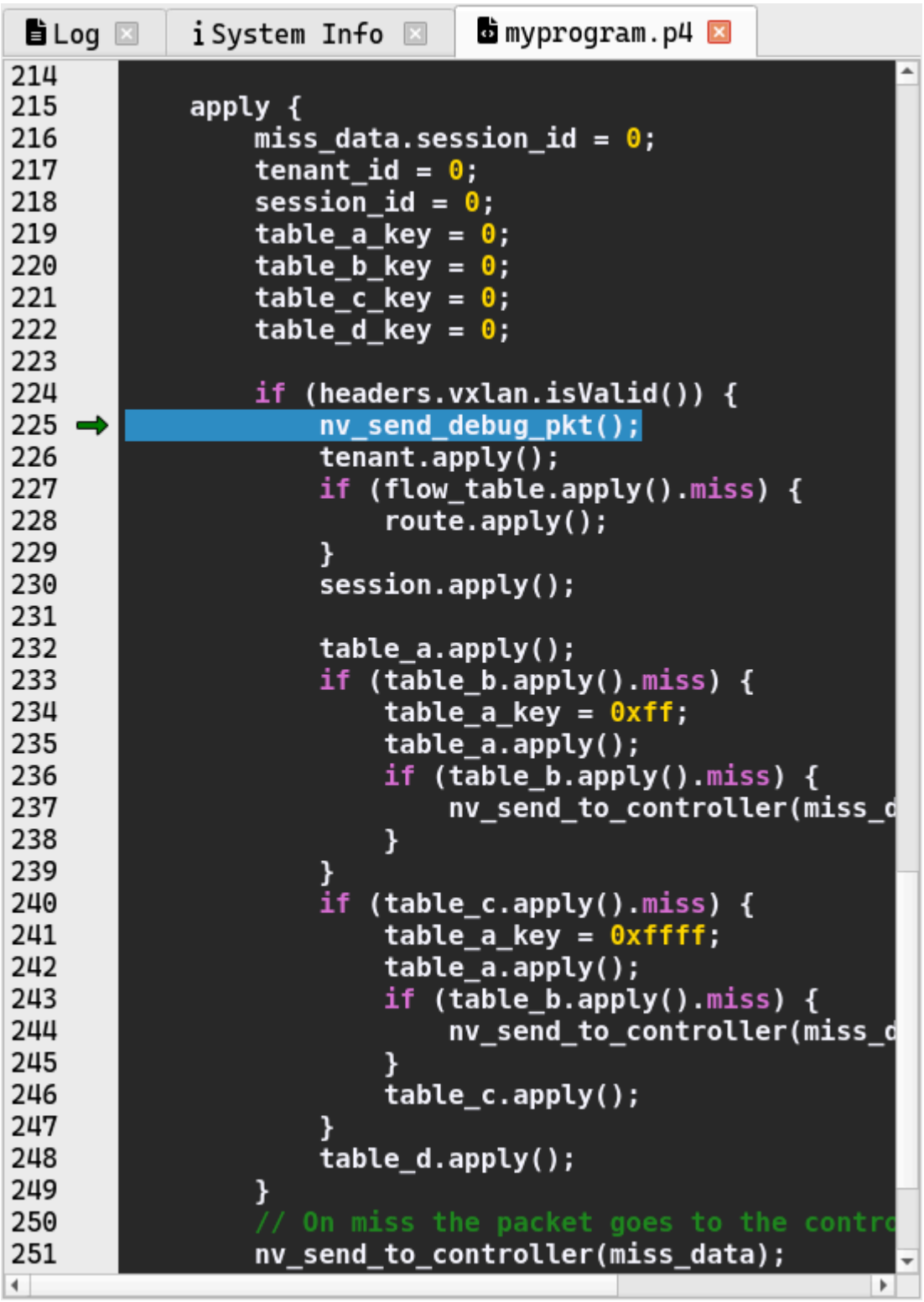
This helps developers inspect both high-level program variables and hardware-level details while debugging.

Source Code

Selecting a cell in a pipeline stage opens the DPL program source code in a new tab on the right-hand side of the DPL Debugger GUI. When applicable, the relevant source location is highlighted.

The debugger locates the DPL source file using:

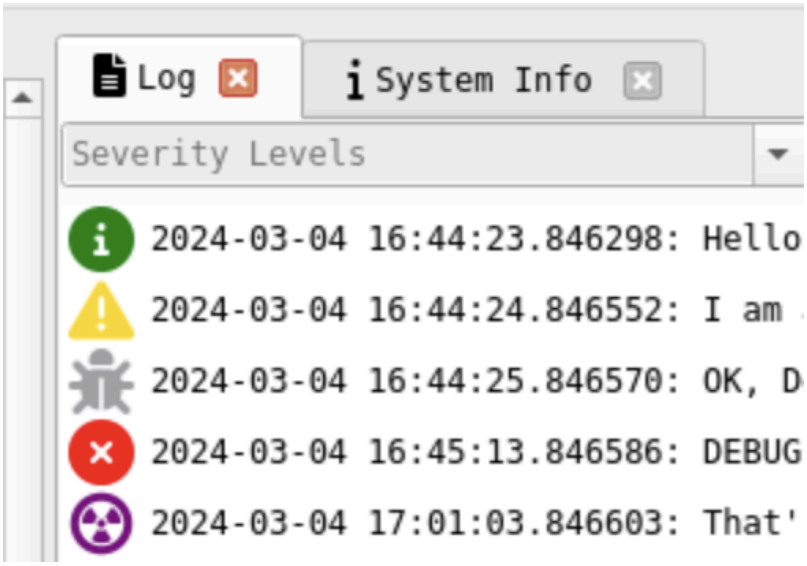
- Search paths defined in the **Settings**
- Manual file selection, if the source file is not found automatically


Log
System Info
myprogram.p4
214
215 apply {
216 miss_data.session_id = 0;
217 tenant_id = 0;
218 session_id = 0;
219 table_a_key = 0;
220 table_b_key = 0;
221 table_c_key = 0;
222 table_d_key = 0;
223
224 if (headers.vxlan.isValid()) {
225 nv_send_debug_pkt();
226 tenant.apply();
227 if (flow_table.apply().miss) {
228 route.apply();
229 }
230 session.apply();
231
232 table_a.apply();
233 if (table_b.apply().miss) {
234 table_a_key = 0xff;
235 table_a.apply();
236 if (table_b.apply().miss) {
237 nv_send_to_controller(miss_d
238 }
239 }
240 if (table_c.apply().miss) {
241 table_a_key = 0xffff;
242 table_a.apply();
243 if (table_b.apply().miss) {
244 nv_send_to_controller(miss_d
245 }
246 table_c.apply();
247 }
248 table_d.apply();
249 }
250 // On miss the packet goes to the contro
251 nv_send_to_controller(miss_data);

Log

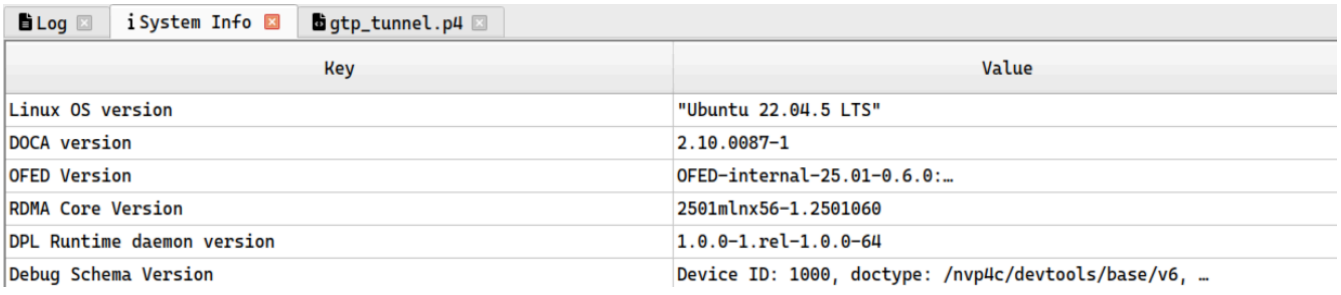
The **Log** tab, located on the right side of the GUI, displays internal logs from the DPL Runtime daemon.

> **Note**
>
> Currently, no log messages are being sent.



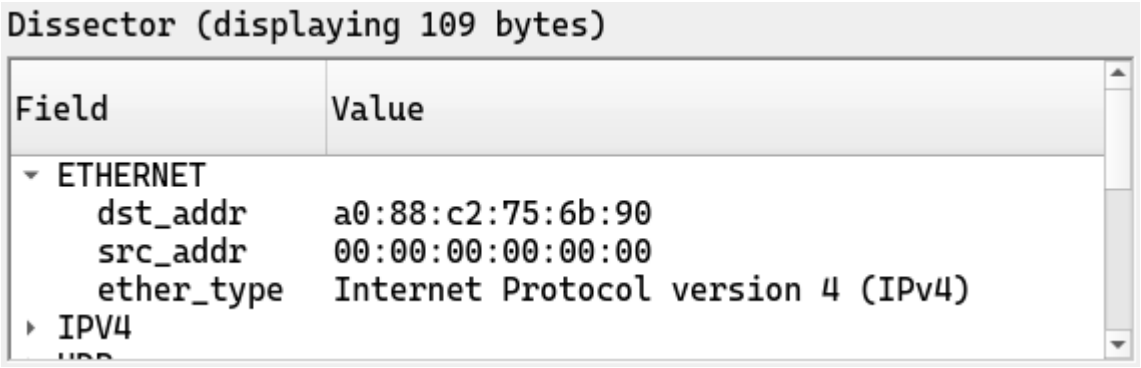
System Info

The **System Info** tab, located on the right side of the GUI, displays details about the system where debugging is performed.



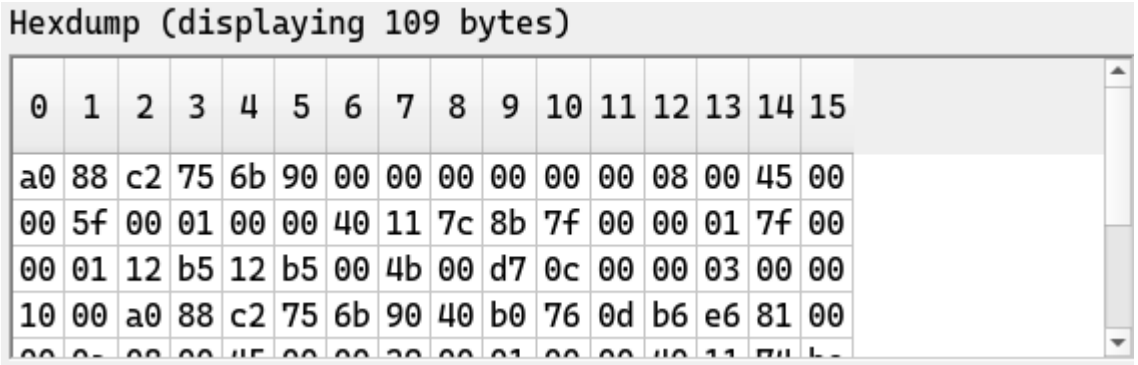
Packet Dissector

The **Dissector** view, located in the center of the GUI, analyzes a packet's structure when a pipeline stage is selected.



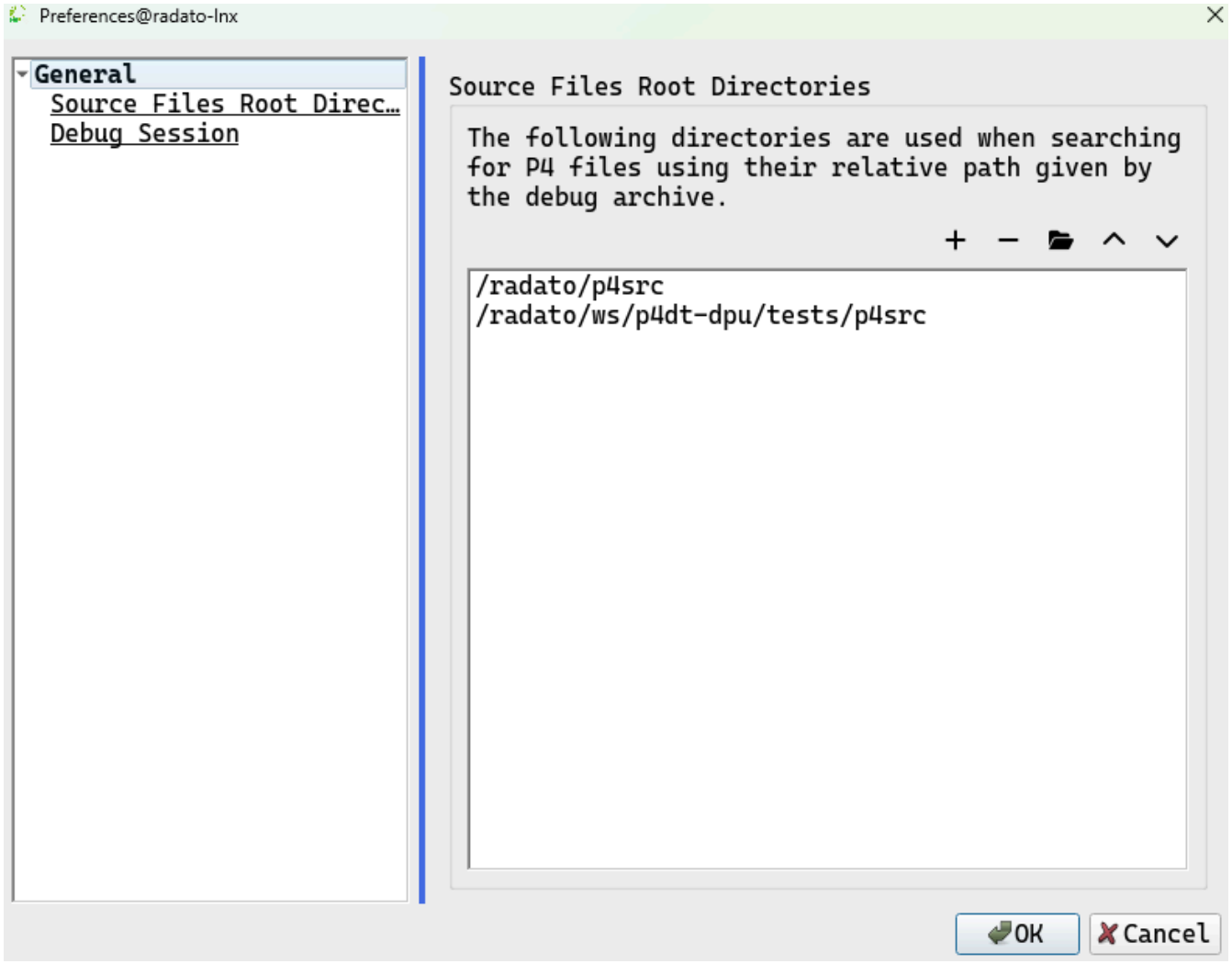
Hex Dump

The **Hex Dump** view, located in the center of the GUI, displays a raw hexadecimal representation of the packet when a pipeline stage is selected.



Options

The **File → Preferences** menu opens the **Preferences** window, where users can configure various debugger and debug session settings.


Preferences@radato-lnx
General
Source Files Root Direc…
Debug Session
Source Files Root Directories
The following directories are used when searching for P4 files using their relative path given by the debug archive.
/radato/p4src
/radato/ws/p4dt-dpu/tests/p4src
OK
Cancel

DPL Admin

Introduction

DPL Admin is a CLI tool used for managing and inspecting the DPL Runtime daemon. It provides the following functionalities:

- Viewing the current configuration of the DPL Runtime daemon
- Reading the DPL Runtime Service logs
- Setting the log level for the DPL Runtime Service
- Retrieving the hardware steering dump, capabilities, and NVConfig of the DPL Runtime Service

Options and Commands

Usage:

```
./dpl_admin.sh -a <host:port> [options] <command> [command-options]
```

Options:

- `-h`, `--help` – Displays help message and exits
- `-v`, `--version` – Displays `dpl_admin`'s version and exits
- `-a`, `--address` – Required; specifies `dpl_admin`'s server address (refer to agent config file) and port of the template ipv4/6: `[address]:[port]`. Examples:
  - `192.168.0.1:9600`
  - `ipv6:[2607:f8b0:400e:c00::ef]:443`

- `ipv6:[::]:1234`

config

Displays the DPL Runtime Service configuration information, including servers, logging, devices and HAL configurations.

- Usage:

```
./dpl_admin.sh -a <host:port> config
```

- Options:
-
  - `-h`, `- help` – Displays command's help message and exit
- Example:

```
./dpl_admin.sh -a 10.1.1.1:9600 config
```

- Output:

```


  CONFIGURATION


     servers
server_address  tcp_port
                                                                dpl_rt
[::]            9559
                                                              dpl_nspect
[::]            9560
```

```
                                                                admin
[::]         9600

       log                                          name:
/var/log/doca/dpl_rtd/agent.log

                                                    level:
INFO

     devices


                     id    counter cache timeout [msec]
controller attr                 interfaces




                    1000               0
port_id:  9876     name  port_id  mtu          mac

   eth7     4     1514  00:00:00:00:00:00

   eth6     3     1514  00:00:00:00:00:00

   eth5     2     1514  00:00:00:00:00:00

   eth4     1     1514  00:00:00:00:00:00

   eth2     0     1514  00:00:00:00:00:00
```

```
       hal
number of queues:   1


queue size:     1024

burst size:      32
```

set-counter-cache-timeout

Sets counter cache timeout for device.

- Usage:

```
./dpl_admin.sh -a <host:port> set-counter-cache-timeout --
device_id/-did <DEVICE_ID> --timeout_value/-nto
<TIMEOUT_VALUE>
```

- Options:
  - `-h`, `- help` – Displays command's help message and exit
  - `--device_id`, `-did` – Required; specifies device ID.
  - `--timeout_value`, `-nto` – Required; specifies new counter cache time-out value
- Example:

```
./dpl_admin.sh -a 10.1.1.1:9600 set-counter-cache-timeout --
device_id 1000 --timeout_value 3
```

- Output:

```
Device ID: 1000, new timeout: 3 msec
```

set-log-level

Sets DPL Runtime Service's log level.

- Usage:

```
dpl_admin.sh -a <host:port> set-log-level --level <LOG_LEVEL>
```

- Options:
  - `-h`, `- help` – Displays command's help message and exit
  - --level – Required; new log level out of `{DISABLE, CRITICAL, ERROR, WARNING, INFO, DEBUG, TRACE}`
- Example:

```
./dpl_admin.sh -a 10.1.1.1:9600 set-log-level --level DEBUG
```

- Output:

```
dpl_rtd log level set to: 'DEBUG'
```

Info

Setting log level is not persistent. DPL Admin restarts with log level according to config file.

get-log-full

Reads DPL Runtime Service's log into a file.

- Usage:

```
./dpl_admin.sh -a <host:port> get-log-full
```

- Options:
  - `-h`, `- help` – Displays command's help message and exit
  - `--outpath` – Optional; specifies output log file path. Defaults to current directory.
- Example:

```
./dpl_admin.sh -a 10.1.1.1:9600 get-log-full --outpath
'/tmp/dpl_agent.log'
```

- Output:

```
Find Log file at: '/tmp/dpl_agent.log'
```

get-log-recent

Displays recent DPL Runtime Service's log lines.

- Usage:

```
./dpl_admin.sh -a <host:port> get-log-recent
```

- Options:
  - `-h`, `- help` – Displays command's help message and exit
  - `--number_of_lines` – Optional; specifies number of recent log lines to read. Defaults to 10.
- Example:

```
./dpl_admin.sh -a 10.1.1.1:9600 get-log-recent --
number_of_lines 6
```

- Output:

```
[12:45:09:480451][73079][DOCA][INF][OnReadDone] New client
connected to device 1000
[12:45:09:481295][73078][DOCA][INF] A program was previously
loaded, clearing it before applying new one...
[12:45:09:481314][73078][DOCA][INF] [DPL Device 1000]
Removing DPL program ...
[12:45:09:498476][73078][DOCA][INF] [DPL Device 1000] DPL
program was removed.
[12:45:09:499432][73078][DOCA][INF] [DPL Device 1000] Loading
DPL program ...
[12:45:09:522419][73078][DOCA][INF] [DPL Device 1000] DPL
program was loaded successfully.
```

hws-dump

Dumps DPL Runtime Service's hardware steering rules into a file.

- Usage:

```
./dpl_admin.sh -a<host:port> hws-dump --device_id/-did
<DEVICE_ID>
```

- Options:
  - `-h`, `- help` – Displays command's help message and exit
  - `--device_id`, `-did` – Required; specifies device ID
  - `--parser_args` – Optional; specifies arguments to pass to HWS dump tool
  - `--outpath` – Optional; specifies output file path. Defaults to current directory.
- Example:

```
./dpl_admin.sh -a 10.1.1.1:9600 hws-dump --device_id 1000 --
parser_args '\-vvv' --outpath '/tmp/dpl_hws_dump.txt'
```

- Output:

```
Find HWS dump file at: '/tmp/dpl_hws_dump.txt'
```

Info

Intended as a debugging aid for developers.

hca-capabilities

Dumps NIC HCA capabilities into a file.

- Usage:

```
./dpl_admin.sh -a<host:port> hca-capabilities --device_id/-did <DEVICE_ID>
```

- Options:
  - `-h`, `- help` – Displays command's help message and exit
  - `--device_id`, `-did` – Required; specifies device ID
  - `--outpath` – Optional; specifies output file path. Defaults to current directory.
- Example:

```
./dpl_admin.sh -a 10.1.1.1:9600 hca-capabilities --device_id 1000 --outpath '/tmp/dpl_hca_capabilities.json'
```

- Output:

```
Find HCA capabilities file at: '/tmp/dpl_hca_capabilities.json'
```

nv-config

Dumps DPL Runtime Service's NV config into a file.

- Usage:

```
./dpl_admin.sh -a<host:port> nv-config --device_id/-did
<DEVICE_ID>
```

- Options:
  - `-h`, `- help` – Displays command's help message and exit
  - `--device_id`, `-did` – Required; specifies device ID
  - `--outpath` – Optional; specifies output file path. Defaults to current directory.
- Example:

```
./dpl_admin.sh -a 10.1.1.1:9600 nv-config --device_id 1000 --
outpath '/tmp/dpl_nv_config.txt'
```

- Output:

```
Find NV config file at: '/tmp/dpl_nv_config.txt'
```

P4 Runtime Controller

DPL applications are deployed to the NVIDIA® BlueField® networking platform (DPU or SuperNIC) via the P4Runtime API.

Since DPL is derived from the P4-16 language, it is compatible with the P4Runtime specification, enabling standard runtime interaction with the compiled DPL pipeline.

Introduction

The P4 Runtime shell (`p4runtime_sh`) is an open-source CLI tool that provides an interface to the P4Runtime API. It is especially useful for:

- Loading simple DPL programs
- Testing match-action tables
- Debugging pipeline behavior

The shell can be invoked using the launch script provided in the DPL Development container.

> **Info**
>
> For detailed instructions, refer to Loading DPL Applications.

p4runtime_sh Usage

The following are example commands for using the p4runtime_sh P4 Controller after loading a program.

P4 info

Operation	Command
Retrieves the content of `p4info.txt` of the currently loaded DPL program	`p4info`

P4 Table

Operation	Command
Lists all P4 tables	`tables`
Displays information about a specific P4 table	`tables["<P4_TABLE_NAME>"]`

Working with P4 Table Entries

Operation	Command
Default Entry	
Reads P4 table's default entry without counter's value	`te = table_entry["<P4_TABLE_NAME>"]` `te.is_default = True` `te.read(lambda te: print(te))`

Operation	Command
Reads P4 table's default entry with counter's value **Note** Supported only if a direct counter is enabled on the P4 table.	`te = table_entry["<P4_TABLE_NAME>"]` `te.is_default = True` `te.counter_data.byte_count = 0` `te.read(lambda te: print(te))`
Modifies P4 table's default entry action **Note** A default entry cannot be removed or inserted. It can only be modified to perform a different P4 action.	`te = table_entry["<P4_TABLE_NAME>"](action="<P4_ACTION_NAME>")` `te.is_default = True` # Set value for all parameters required by desired action. `te.action["<PARAMETER NAME>"] = "<PARAMETER VALUE>"` `te.modify()`

Regular Entries

Note: In the following examples, some commands require specifying the match key of the desired regular P4 table key.
The syntax for specifying a match key varies according to the defined match method for each key (per the DPL source code where the keys are defined on the P4 table).

Match Method	Syntax for Specifying Match Key Value
exact	`te.match["<MATCH_KEY_NAME>"] = "<MATCH_VALUE>"`

<table>
<tr><th colspan="2">Operation</th><th>Command</th></tr>
<tr><td colspan="3">
<table>
<tr><th>Match Method</th><th>Syntax for Specifying Match Key Value</th></tr>
<tr><td>ternary</td><td>

```
te.match["<MATCH_KEY_NAME>"] = "<MATCH_VALUE>&&&<MASK_VALUE>"
te.priority = <PRIORITY VALUE>
```

Note that mask is provided in the match line, separated by &&& . If mask is not specified , a full match mask will be used.</td></tr>
<tr><td>lpm</td><td>

```
te.match["<MATCH_KEY_NAME>"] = "<MATCH_VALUE>/<PREFIX_LENGTH_VALUE>"
```

Note that LPM prefix_len is provided in the match line, separated by / . If LPM prefix_len is not specified , a prefix_len with full field bitwidth is used.</td></tr>
</table>
For simplicity, the following examples are written using <code>exact</code> match syntax.
</td></tr>
<tr><td colspan="3">Reading Entries</td></tr>
<tr><td colspan="2">Reads a specific regular P4 table entry</td><td>

```
te = table_entry["<P4_TABLE_NAME>"]
try:
    pass
    # Comment out the next line to disable reading counters
te.counter_data.byte_count = 0
except Exception as e:
    # Table does not have a Direct Counter
    pass
# Set value for all keys required by the P4 table.
```
</td></tr>
</table>

Operation	Command
	`te.match["<MATCH_KEY_NAME>"] = "<MATCH_VALUE>"` `te.read(lambda te: print(te))`
Reads all regular entries from a P4 table	`num = 1` `def hndlr(te):` `    global num` `    print(f">> Entry number {num}:")` `    print(te)` `    print("--------------------------------------------------")` `    num += 1` `te = table_entry["<P4_TABLE_NAME>"]` `try:` `    pass` `    # Comment out the next line to disable reading counters` `te.counter_data.byte_count = 0` `except Exception as e:` `    # Table does not have a Direct Counter` `    pass` # Read regular entries `te.is_default = False`

Operation	Command
	`te.read(lambda te: hndlr(te))`
Reads all regular entries from all P4 tables in the P4 program	`for tbl in tables:` `  num = 1` `  def hndlr(te):` `  global num` `  print(f">> Entry number {num}:")` `  print(te)` `  print("----------")` `  num += 1` `print(f"========== {tbl.name} ==========")` `  te = table_entry[tbl.name]` `  try:` `  pass` `  # Comment out the next line to disable reading counters` `te.counter_data.byte_count = 0` `  except Exception as e:` `  # Table does not have a Direct Counter` `  pass`

Operation	Command
	# Read regular entries te.is_default = False te.read(lambda te: hndlr(te))
Adding Entries	
Adds a regular P4 table entry	te = table_entry["<P4_TABLE_NAME>"](action="<P4_ACTION_NAME>") # Set value for all keys required by the P4 table. te.match["<MATCH_KEY_NAME>"] = "<MATCH_VALUE>" # Set value for all parameters required by desired action. te.action["<PARAMETER NAME>"] = "<PARAMETER VALUE>" te.insert()
Deleting Entries	
Deletes a specific regular P4 table entry	te = table_entry["<P4_TABLE_NAME>"] # Set value for all keys required by the P4 table. te.match["<MATCH_KEY_NAME>"] = "<MATCH_VALUE>"

Operation	Command
	`te.delete()`
Deletes all regular entries from a P4 table	`te = table_entry["<P4_TABLE_NAME>"]` `te.read(lambda te: te.delete())`

Working with Direct Counters

This allows w orking with Direct Counter directly without getting the whole Table Entry info.

Operation	Command
Lists all defined Direct Counters	`direct_counters`
Default Entry	
Reads counter data of a default entry in a Direct Counter	`DirectCounterEntry("<P4_DIRECT_COUNTER_NAME>")` `ce.table_entry.is_default = True` `ce.read((lambda ce: print(ce)))`
Clears counter data of a default entry in a Direct Counter	`DirectCounterEntry("<P4_DIRECT_COUNTER_NAME>")`

Operation	Command
	``ce.table_entry.is_default = True ce.packet_count = 0 ce.modify()```
Regular Entries	
Reads counter data of a a specific P4 table entry in a Direct Counter	```ce = DirectCounterEntry("<P4_DIRECT_COUNTER_NAME>") # Set value for all keys required by the P4 table. ce.table_entry.match["<MATCH_KEY_NAME>"] = "<MATCH_VALUE>" ce.read((lambda ce: print(ce)))```
Reads all counter data of specific Direct Counter **Info** This will also read the default entry counter data.	```ce = DirectCounterEntry("<P4_DIRECT_COUNTER_NAME>") ce.read((lambda ce: print(ce)))```
Clears counter data of a a specific P4 table entry in a Direct Counter	```ce = DirectCounterEntry("<P4_DIRECT_COUNTER_NAME>") # Set value for all keys required by the P4 table. ce.table_entry.match["<MATCH_KEY_NAME>"] = "```

Operation	Command
	<MATCH_VALUE>" ce.packet_count = 0 ce.modify()
Clears all counter data of specific Direct Counter **Note** This also clears the default entry counter data.	ce = DirectCounterEntry("<P4_DIRECT_COUNTER_NAME>") ce.packet_count = 0 ce.modify()
Clears all direct counters from all table	for dc in direct_counters: ce = DirectCounterEntry(dc.name) ce.packet_count = 0 ce.modify()

Working with Indirect Counters

Operation	Command
Lists all defined Indirect Counters	counters
Shows info about a specific Indirect Counter	counter_entry["<P4_INDIRECT_COUNTER_NAME>"]

Operation	Command
Reads a specific value from a specific indirect counter	`ce = counter_entry["<P4_INDIRECT_COUNTER_NAME>"]` `ce.index = <COUNTER_CELL_INDEX>` `ce.read((lambda ce: print(ce)))`
Reads all values from a specific indirect counter	`ce = counter_entry["<P4_INDIRECT_COUNTER_NAME>"]` `ce.read((lambda ce: print(ce)))`
Clears a specific value from a specific indirect counter	`ce = counter_entry["<P4_INDIRECT_COUNTER_NAME>"]` `ce.index = <COUNTER_CELL_INDEX>` `ce.byte_count = 0` `ce.packet_count = 0` `ce.modify()`
Clears all values from a specific indirect counter	`ce = counter_entry["<P4_INDIRECT_COUNTER_NAME>"]` `ce.byte_count = 0` `ce.packet_count = 0` `ce.modify()`

P4 Actions

Operation	Command
Lists all defined P4 actions	`actions`
Shows info about a specific P4 actions **Info** This includes its parameters' names and sizes.	`actions["<P4_ACTION_NAME>"]`

Packet IO

To use packet IO, it must be enabled in the P4 program source code.

Operation	Command
Packet In – For receiving packets sent from the DPL Runtime daemon to the P4 Controller (according to defined rules in the DPL program)	
Captures packets for 10 second, then displays them	`packet_in.sniff(lambda m: print(m), timeout=10)`
Capturs packets for 10 second, then displays them parsed as well as their metadata info	This example command uses the `impacket` package for parsing the packets, so make sure that it is installed on your system prior to running the `p4runtime_sh` P4 Controller: `pip install impacket` Then, from the `p4runtime_sh` P4 Controller, run:

Operation	Command
	`` from impacket.ImpactDecoder import * for msg in packet_in.sniff(timeout=10): print("----------") print(msg) print("++") print("Raw packet (hex):\n") print(msg.packet.payload.hex()) print("++") print("Parsed packet:\n") print(EthDecoder().decode(msg.packet.payload)) print("++") print("Metadata:\n") for md in msg.packet.metadata: val = int.from_bytes(md.value, "big") print("metadata ID (", md.metadata_id, "): ", hex(val)) ```
Packet Out – For sending a packet from the `p4runtime_sh` P4 Controller to the DPL Runtime daemon (which will process it according to defined rules in the DPL program)	
Syntax of sending a packet	``` my_pkt = b'<PACKET_BYTE_STRING_MESSAGE_GOES_HERE>' ```

Operation	Command
	`packet_out(payload=my_pkt, <METADATA_NAME>="METADATA_VALUE_GOES_HERE").send()` Repeat the `<METADATA_NAME>="METADATA_VALUE_GOES_HERE"` parameter for each defined metadata in the P4 program.
Example of building and sending a packet	Using `scapy` tool, build the desired packet: `>>> p = Ether(src='00:11:11:11:11:11', dst='00:22:22:22:22:22') / IP(src="1.1.1.1", dst="2.2.2.2")` `>>> print(p.build())` `b'\x00"""""\x00\x11\x11\x11\x11\x11\x08\x00E\x00\x00\x14\x00\x01\x00\x00@\x00t\xe4\x01\x01\x01\x01\x02\x02\x02\x02'` `>>>` Then send it using the `p4runtime_sh` P4 Controller: `my_pkt = b'\x00"""""\x00\x11\x11\x11\x11\x11\x08\x00E\x00\x00\x14\x00\x01\x00\x00@\x00t\xe4\x01\x01\x01\x01\x02\x02\x02\x02'` `packet_out(payload=my_pkt, controller_metadata="0x123").send()`