DOCA Flow Tune Tool

Table of contents

Introduction

- Monitor Mode Overview
- Analyze Mode Overview
- Visualize Mode Overview

Dependencies

Prerequisites

Execution

- Monitor Command
- CLI Examples
- Analyze Command
- CLI Examples
- Visualize Command
- CLI Examples

Configuration

- Config File Default Values
- Custom Config File
- Overriding Config Values from CLI
- Common Configuration Values
- Output Directory
- Connection to DOCA Flow Tune Server

Monitor Mode

- Hardware Counters

Software Key Performance Indicators

Configuration

CSV Format

Configuration File

Analyze Mode

Pipeline Export

Visualize Mode

Viewing the Pipeline

Reading the Visualization

Input Graph

Pipeline Graph

Output Graph

Troubleshooting

Telemetry fwctl driver is not loaded

Error

Solution

Mermaid visualization in Visual Studio Code

Limited feature set – could not detect a running DOCA Flow program

Error

Solution

Appendix – Configuration File Example

DOCA Flow Tune is a powerful, one-stop-shop solution, providing visibility and analysis capabilities for DOCA Flow programs.

Introduction

> **Info**
>
> DOCA Flow Tune is supported at alpha level.

DOCA Flow Tune is a one-stop-shop solution which allows developers to visualize their traffic steering pipelines, have a live monitor of software Key Performance Indicators (KPIs) as well as hardware counters, and gain valuable performance insights about their DOCA-Flow-based program.

DOCA Flow Tune is especially useful for the following scenarios:

- Aiding developers during the development of their traffic steering pipeline by providing visualization of the pipeline, and later also performance insights about the designed pipeline
- Aiding developers in pre-production environment by providing live monitoring of the performance indicators of the program on both software and hardware levels, and helping detect possible bottlenecks/critical paths so they are addressed before deployment to production environments
- Aiding administrators monitor the program in production by providing live monitoring as well as high-rate hardware counters to be used when analyzing a possible deployment/setup issue

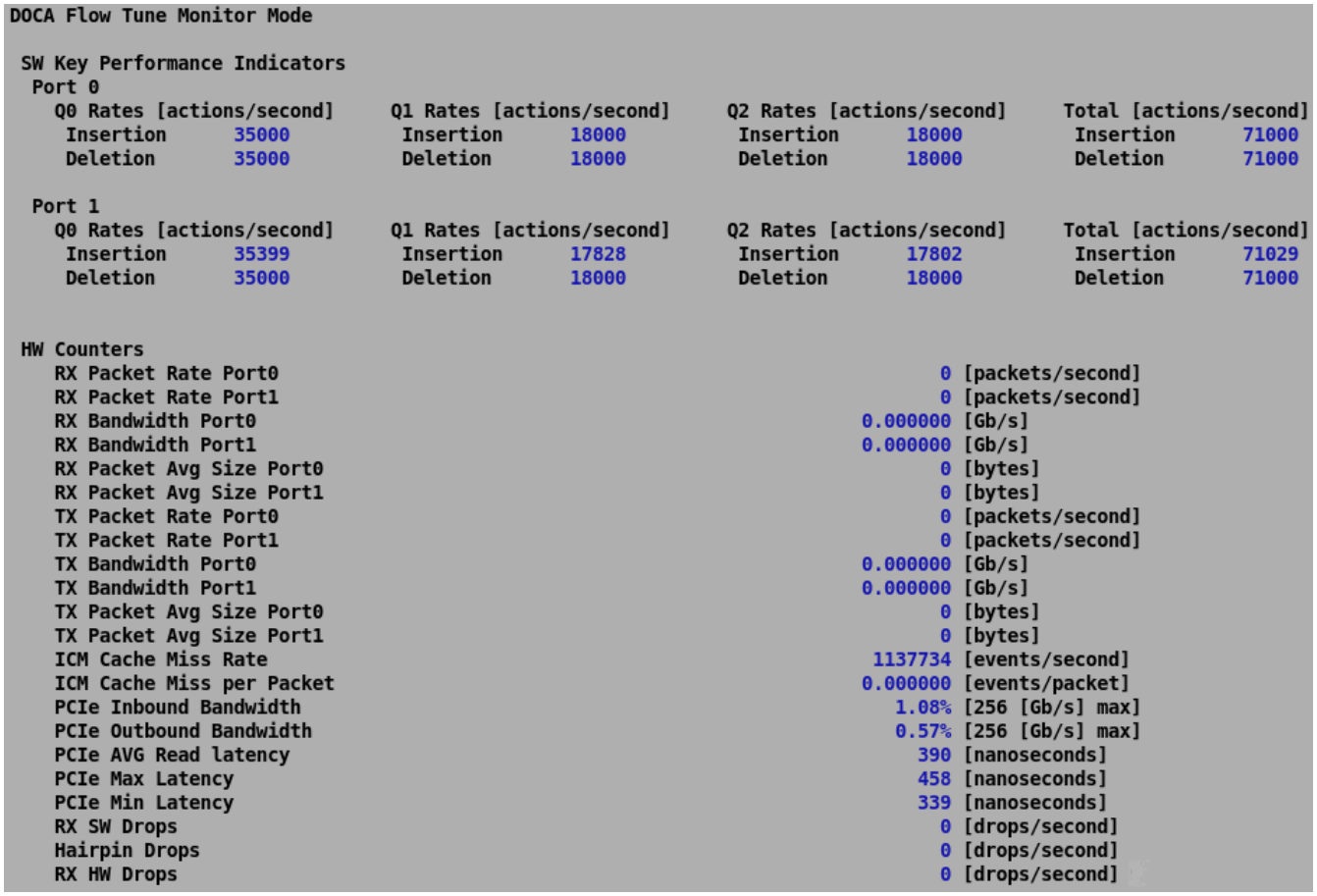
The tool operates in three distinct modes, Monitor, Analyze, and Visualize, which are presented in the following subsections.

> **Note**
>
> Collecting, analyzing, and displaying information from the analyzed DOCA-Flow-based program requires the explicit activation of the DOCA Flow Tune server by the target program. For more information

about that, please refer to the DOCA Flow Tune Server programming guide.

Monitor Mode Overview

This mode collects and displays both hardware counters and software KPIs in real time (as extracted from the running DOCA Flow program and the underlying setup), providing a comprehensive view of the system's performance:



This information can also be exported to a CSV file for further analysis.

Info

For more information about this mode, please refer to section "Monitor Mode".

Info

For information about running DOCA Flow Tune in this mode, please refer to section "Monitor Command".

Analyze Mode Overview

The analyze mode supports the ability to dump the internal steering pipeline state to be used by Visualize mode.

Info

For more information about this mode, please refer to section "Analyze Mode".

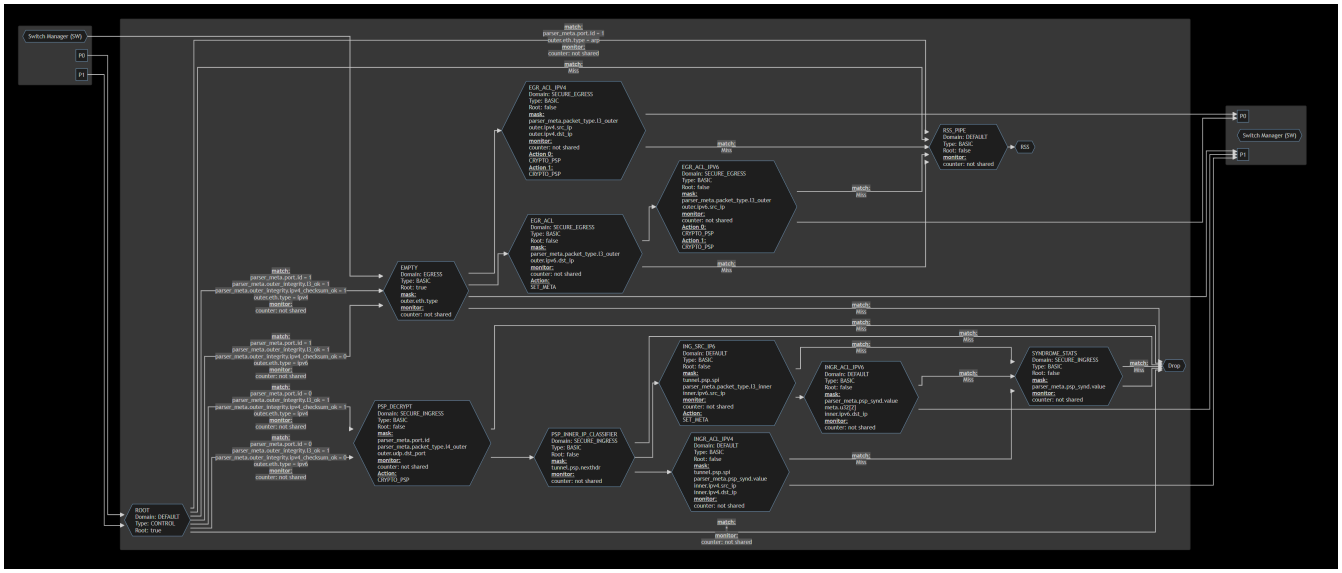
Info

For more information about running DOCA Flow Tune in this mode, please refer to section "Analyze Command".

Visualize Mode Overview

This mode allows users to produce a graphical representation of their steering pipeline (as built using the DOCA Flow API), allowing developers to quickly understand their program's pipeline, and compare it with their intended architecture.

The following is an example from the DOCA PSP Gateway reference application:



This schema shows the program's pipeline state at the time of query. This figure is read left-to-right and represents the possible packet flow in the defined pipeline.

> **Info**
>
> For more information about this mode, please refer to section "Visualize Mode".

> **Info**
>
> For more information on running DOCA Flow Tune in this mode, please refer to section "Visualize Command".

Dependencies

DOCA Flow Tune depends on the following DOCA SDK libraries:

- DOCA Flow

- DOCA Telemetry

Prerequisites

- DOCA 2.9.0 and higher.
- For optimal experience, it is recommended to comply with the prerequisites of all the listed dependencies, and especially with their recommended firmware versions.

Execution

To execute DOCA Flow Tune tool:

```
Usage: doca_flow_tune [Program Commands] [DOCA Flags] [Program
Flags]

Program Commands:
  analyze                              Run Flow Tune in Analyze mode
  monitor                              Run Flow Tune in Monitor mode
  visualize                            Run Flow Tune in Visualize
mode

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 interface (CLI) using the `-h` (or `--help`) option:

```
doca_flow_tune -h
```

The same applies for each of the tool's commands (and subcommands). For instance:

```
doca_flow_tune monitor -h
```

Monitor Command

The `monitor` command presents software KPIs and hardware counters. Each component offers various options, which can be specified in the configuration file under the `monitor` section, or through the CLI.

```
Usage: doca_flow_tune monitor [Program Commands] [DOCA Flags]
[Program Flags]

Program Commands:
  background                          Collect software key
performance indicators and hardware counters on the background

DOCA Flags:
  -h, --help                          Print a help synopsis
  -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>

Program Flags:
  --enable-csv                       Enable dumping data to CSV
file
  --disable-csv                      Disable dumping data to CSV
file
  --csv-file-name                    CSV file name to create
  --hw-profile                       Register hardware profile
{basic, full}
  --sw-profile                       Register software profile
  -f, --cfg-file                     JSON configuration file
```

Supported sub-commands:

- `background` – This subcommand allows performing CSV dumping without displaying the output on the screen. This is useful for scenarios where one wants to log counters without cluttering the terminal. It also supports high-rate dumping for hardware counters which may be activated using the `--high-rate` flag.

```
Usage: doca_flow_tune monitor background [DOCA Flags]
[Program Flags]

DOCA Flags:
  -h, --help                         Print a help synopsis
  -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>

Program Flags:
```

```
  --high-rate                          Enable dumping hardware
counters data to CSV file in high rate
  --hw-profile                         Register hardware profile
{basic, full}
  --sw-profile                         Register software profile
```

CLI Examples

- To launch the `monitor` command with a given configuration file:

```
doca_flow_tune monitor -f /tmp/flow_tune_cfg.json
```

- To launch the `monitor` command with both a given configuration file and a CLI parameter for specifying the desired hardware counters profile:

```
doca_flow_tune monitor -f /tmp/flow_tune_cfg.json --hw-
profile basic
```

- To launch the `monitor` command with the `background` subcommand and the request to perform a high rate collection and export for the hardware counters:

```
doca_flow_tune monitor -f /tmp/flow_tune_cfg.json background
--high-rate
```

> **Note**
>
> The tool silently creates and updates the `flow_tune.csv` file.

Analyze Command

The `analyze` command runs a specified set of analysis methods over the target DOCA Flow program. The analysis supports the ability to export a JSON description of the steering pipeline, as is used by the `visualize` command, and could later be used for future analysis methods (both online or offline).

```
Usage: doca_flow_tune analyze export [DOCA Flags] [Program Flags]

DOCA Flags:
  -h, --help                        Print a help synopsis
  -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>

Program Flags:
  --file-name                       File name on which the
pipeline information will be saved
  -f, --cfg-file                    JSON configuration file
```

Supported subcommands:

- `export` – This command allows the tool to export a running DOCA Flow program's pipeline into a JSON file. This file is the main input for other features of the tool, such as the graphical visualization.

> **Note**

The `export` subcommand is currently mandatory.

CLI Examples

- To launch the analyze command without a configuration file:

```
doca_flow_tune analyze export
```

  The JSON file is stored into its default path.

- To launch the `analyze` command with a given configuration file that specifies the desired values for all needed configurations:

```
doca_flow_tune analyze export -f /tmp/flow_tune_cfg.json
```

- To launch the `analyze` command with a configuration file while also configuring the output path for the exported JSON file through the CLI:

```
doca_flow_tune analyze export -f /tmp/flow_tune_cfg.json --
file-name my_program_pipeline_desc.json
```

  The exported pipeline is stored as `my_program_pipeline_desc.json` into the chosen/default output directory.

Visualize Command

The `visualize` command visualizes the steering pipeline of a given DOCA Flow program. The command works on a given JSON file as input. This file can either be

generated by the `analyze export` command or queried dynamically from a running program, in which case the command would dump the pipeline from the program and then generate the visualization output file.

Info

The visualization output file is a Mermaid markdown format.

Info

This file can be fed to any of the widely available Mermaid visualization tools, as explained in depth in the corresponding section "Visualize Mode".

```
Usage: doca_flow_tune visualize [DOCA Flags] [Program Flags]

DOCA Flags:
  -h, --help                        Print a help synopsis
  -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>

Program Flags:
  --pipeline-desc                   Input JSON file that
represents the Flow application pipeline
  --file-name                       File name on which the
visualization information will be saved
```

```
  -f, --cfg-file                    JSON configuration file
```

CLI Examples

- Launching the `visualize` command without the configuration file leads to a live query of the pipeline against the running DOCA Flow program:

  ```
  doca_flow_tune visualize
  ```

- Launching the `visualize` command with a given configuration file that specifies the desired values for all needed configurations:

  ```
  doca_flow_tune visualize -f /tmp/flow_tune_cfg.json
  ```

- To launch the `visualize` command with a configuration file while configuring the output path for the Mermaid file through the CLI and providing an offline pipeline file:

  ```
  doca_flow_tune visualize -f /tmp/flow_tune_cfg.json --file-
  name my_program_pipeline_viz.md --pipeline-desc
  my_program_pipeline_desc.json
  ```

  The exported Mermaid file is stored as `my_program_pipeline_viz.md` into the chosen/default output directory. Because the pipeline description file is explicitly provided, this command could be used offline, as it would not need connection with the DOCA Flow program to visualize.

Configuration

DOCA Flow Tune has a configuration file which allows customizing various settings.

> **Info**
>
> The configuration file is divided into sections in order to simplify its usage.

Config File Default Values

If a configuration file is not provided, DOCA Flow Tune uses its default values for fields which are mandatory.

> **Info**
>
> A list of all default values can be seen in the appendix.

In Monitor mode, if a software KPIs or hardware counters query is not needed, removing the `hardware` or `software` fields from the configuration file disables the respective feature.

Custom Config File

Instead of using default configuration values, users can create a file of their own and provide a file path when running DOCA Flow Tune (`-f`/`--cfg-file`).

Once used, DOCA Flow Tune loads all provided values directly from the file, while the rest of the fields (if any) use their respective default values.

Overriding Config Values from CLI

Setting s ome of the fields in the configuration file is supported through CLI using the `--file-name` flag. If used, the provided values from the CLI would override the values of the fields from the configuration file. This allows for easier configuration of common values without the need to create a new custom file or to modify an existing configuration file.

Common Configuration Values

Some sections of the configuration file are shared between multiple runtime modes of DOCA Flow Tune (i.e., Monitor, Analyze, Visualize) and generally have to do with the output/input file paths and interaction with the live DOCA Flow program.

```
{
...
        "outputs_directory": "/tmp/flow_tune/",
...
        "network": {
                "server_uds": "/tmp/tune_server.sock",
                "uds_directory": "/var/run/doca/flow_tune/"
        },
...
}
```

Output Directory

`outputs_directory` defines the main directory on which all output products are saved. This field does not have a default value. If no value is provided, DOCA Flow Tune files are saved at the following directories:

- CSV file – `/var/log/doca/flow_tune/`
- Analyze export pipeline description file – `/tmp/flow_tune/`
- Pipeline visualization file – `/tmp/flow_tune/`

Connection to DOCA Flow Tune Server

Some features of DOCA Flow Tune work by interacting with a live DOCA Flow based program. This is enabled through a server that is running in the background as part of the DOCA Flow library, and requires all of the following to be applied:

1. DOCA Flow based program should explicitly enable the server.

   > **Info**
   >
   > More information is available in the relevant DOCA Flow Tune Server programming guide.

2. The DOCA-Flow-based program should run using the trace-enabled DOCA Flow library.

   > **Info**
   >
   > More information is available in the "Debug and Trace Features" section of the DOCA Flow programming guide.

DOCA Flow Tune should be configured in a way to allow it to connect to the matching server. This can be done by modifying the following variables under the `network` section of the configuration file:

- `server_uds` – DOCA Tune Server Unix Domain Socket (UDS) path. Default value is `/tmp/tune_server.sock`.
- `uds_directory` – Directory on which all local UDSs are created. Default value is `/var/run/doca/flow_tune/`.

Monitor Mode

Hardware Counters

This table provides the supported hardware counters and their associated profiles.

Counter Name	Description	Unit	Profile Basic	Profile Full	Notes
RX Packet Rate	The number of received packets per second	pkt/sec	✔	✔	
RX Bandwidth	The data transfer rate based on the number of packets received per second	Gb/s	✔	✔	
RX Packet Average Size	The average size of received data packets	Bytes	✔	✔	
TX Packet Rate	The number of packets transmitted per second	pkt/sec	✔	✔	
TX Bandwidth	The data transfer rate based on the number of packets transmitted per second	Gb/s	✔	✔	
TX Packet Average Size	The average size of transmitted data packets	Bytes	✔	✔	
RX SW	The number of dropped packets due to a lack of WQE for the	drop	✔	✔	If drops are observed, this may be because the software was unable to process all received

Counter Name	Description	Unit	Profile Basic	Profile Full	Notes
Drops	associated QPs/RQs (excluding hairpin QPs/RQs) **Info** Supported only on NVIDIA® ConnectX®-7 and above.	s/sec			packets. Consider reducing CPU processing time or increasing the number of utilized cores and queues.
Hairpin Drops	The number of dropped packets due to a lack of WQE for the associated hairpin QPs/RQs **Info** Supported only on NVIDIA® ConnectX®-7 and above.	drops/sec	✓	✓	If drops are observed, the Tx packet processing is probably causing a bottleneck. Consider simplifying the process or adjusting the number or size of hairpin queues, or implementing locking mechanisms.
RX HW Drops	The number of packets discarded due to no available data or descriptor buffers in the RX buffer	drops/sec	✓	✓	If drops are observed, the Rx packet processing is probably causing a bottleneck. Consider simplifying it.
ICM Cache Miss Rate	The rate of data requests that miss in the ICM (interconnect context memory) cache	events/sec	✗	✓	

Counter Name	Description	Unit	Profile Basic	Profile Full	Notes
ICM Cache Miss per Packet	The number of data requests that miss per packet	events/pkt	✗	✓	
PCIe Inbound Bandwidth	The number of bits received from the PCIe toward the device per second.	Gb/s	✗	✓	PCIe counters are supported only on the host side
PCIe Outbound Bandwidth	The number of bits transmitted from the device toward the PCIe per second	Gb/s	✗	✓	
PCIe AVG Read latency	The average PCIe read latency for all read data	nsec	✗	✓	
PCIe Max Latency	T he maximum latency (in nanoseconds) for a single PCIe read from the device	nsec	✗	✓	
PCIe Min Latency	T he minimum latency (in nanoseconds) for a single PCIe read from the device	nsec	✗	✓	

Software Key Performance Indicators

This table provides the supported software KPIs and their associated profiles.

Key Performance Indicator	Description	Units	Profile
Insertion rate	The number of successful table entry insertion actions (per queue) per second.	actions/sec	`entries_ops_rates`
Deletion rate	The number of successful table entry deletion actions (per queue) per second.	actions/sec	`entries_ops_rates`

Configuration

CSV Format

The CSV format stores two types of rows, specific to each counter module:

- ***Hardware Counter Rows (Module ID=0)***

Module ID	HW Counter ID	Counter Value	Timestamp
0	1	8	142623139459
0	2	197503959728	142623139459

  - Module ID – Hardware module identifier
  - HW Counter ID – Unique identifier for the hardware counter
  - Counter Value – Counter value
  - Timestamp – Hardware timestamp

Hardware Counter ID Mapping

HW Counter ID	Description	Units
0	Number of RX packets on port 0	Packets
1	Number of RX packets on port 1	Packets
2	Number of RX bytes on port 0	Bytes

HW Counter ID	Description	Units
3	Number of RX bytes on port 1	Bytes
4	Rate of RX packets on port 0	Packets per second
5	Rate of RX packets on port 1	Packets per second
6	RX bandwidth on port 0	Gb/s
7	RX bandwidth on port 1	Gb/s
8	Average RX packet size on port 0	Bytes
9	Average RX packet size on port 1	Bytes
10	Number of TX packets on port 0	Packets
11	Number of TX packets on port 1	Packets
12	Number of TX bytes on port 0	Bytes
13	Number of TX bytes on port 1	Bytes
14	Rate of TX packets on port 0	Packets per second
15	Rate of TX packets on port 1	Packets per second
16	TX bandwidth on port 0	Gb/s
17	TX bandwidth on port 1	Gb/s
18	Average TX packet size on port 0	Bytes
19	Average TX packet size on port 1	Bytes
20	Number of ICMC misses	Events
21	ICMC misses rate	Events per second
22	ICMC misses per packet	Events per packet
23	The bandwidth of bytes received from PCIe toward the device	Gb/s
24	The bandwidth of bytes transmitted from the device toward PCIe	Gb/s

HW Counter ID	Description	Units
25	The average PCIe read latency	Nanoseconds
26	The total latency for all PCIe read from the device	Nanoseconds
27	The total number of PCIe packets	Events
28	The maximum latency for a single PCIe read from the device	Nanoseconds
29	The minimum latency for a single PCIe read from the device	Nanoseconds
30	RX software drops	Drops per second
31	Hairpin drops	Drops per second
32	RX hardware drops	Drops per second

- ***Software KPI Rows (Module ID=1)***

Module ID	Port ID	SW Counter Type	Counter Value	Timestamp
1	0	Queue 0 Insertion Rate	34511	1727345744137828
1	1	Queue 0 Insertion Rate	37050	1727345755137828

  - Module ID – Software module identifier
  - Port ID – Software port ID
  - SW KPI Type – KPI type
  - KPI Value – KPI value
  - Timestamp – Software timestamp

Configuration File

DOCA Flow Tune's configuration file consists of two main parts of relevance for Monitor mode:

- `csv` dump object
- `monitor` configuration object

The following is an example for both sections:

```
{
...
            "csv": {
                "enable": false,
                "file_name": "flow_tune.csv",
                "max_size_bytes": 1000000,
                "max_files": 1
            },

...

            "monitor": {
                "screen_mode": "dark",                  // modes: {light, dark}
                "hardware": {
                    "pci_addresses": [
                        "b1:00.0",
                        "b1:00.1"
                    ],
                    "profile": "full"     // profiles: {basic, full}
                },
                "software": [
                    {
                        "flow_port_id": 0,
                        "profiles": [
                            "entries_ops_rates"     // profiles:
{entries_ops_rates}
                        ]
                    },
                    {
                        "flow_port_id": 1,
```

```
                                        "profiles": [
                                            "entries_ops_rates"
                                        ]
                                    }
                                ]
                            }
    ...
    }
```

CSV Configuration Section

CSV dumping allows exporting the hardware and software counters collected by the tool into a CSV file for further analysis or record keeping. This is particularly useful for logging performance metrics over time.

How to Enable CSV Dumping

To enable CSV dumping, modify the configuration in the JSON file as follows:

```
{
    "csv": {
        "enable": true,
        "file_name": "flow_tune.csv",
        "max_size_bytes": 1000000,
        "max_files": 1
    }
}
```

The supported fields are:

- `enable` – Set to `true` to enable CSV dumping or `false` to disable it. Default value is `false`.
- `file_name` – The name of the CSV file where the data will be saved.

- `max_size_bytes` – The maximum size (in bytes) of the CSV file. Once this limit is reached, a new file is created based on the `max_files` setting.
- `max_files` – The maximum number of CSV files to keep. When this limit is reached, the oldest files are deleted.

CSV dumping can also be enabled or disabled from the CLI using the `--enable-csv` or `--disable-csv` flags, respectively. For example:

```
doca_flow_tune monitor -f /tmp/flow_tune_cfg.json --enable-csv
```

Additionally, the CSV filename can be updated by using the `--csv-file-name` flag, for example:

```
doca_flow_tune monitor -f /tmp/flow_tune_cfg.json --csv-file-name
"counters_dump.csv"
```

Monitor Configuration Section

Screen Mode

The Monitor module supports two screen modes: `dark` and `light`.

Hardware

The `hardware` section includes the `pci_addresses` and `profile` fields:

- The `pci_addresses` field expects an array of PCIe addresses for NIC ports. The tool uses these addresses to retrieve the corresponding NIC device and the desired port IDs.

> **Note**
>
> PCIe addresses must belong to the same device.

> **Info**
>
> The tool supports up to two ports per device.

- The `profile` field expects to receive either a `basic` or `full` profile.
  - `basic` profile – includes packet- and port-related counters (i.e., Bandwidth, Packets Per Second, Average Packet Size, Packet Drops)
  - `full` profile – includes all the `basic` counters and adds additional debug counters (e.g., ICMC and PCIe counters)

    > **Info**
    >
    > For more information about the counters please refer to section "Hardware Counters".

The hardware counters profile can be set from the CLI by adding `--hw-profile`. For example:

```
doca_flow_tune monitor -f /tmp/flow_tune_cfg.json --hw-profile
basic
```

Software

The `software` section includes the `flow_port_id` and `profiles` fields:

- `flow_port_id` field – expects a single DOCA Flow port identification number. Flow port ID should be set by the DOCA Flow program, by calling the `doca_flow_port_cfg_set_devargs()` API call with a proper port ID string.
- `profiles` field – expects to receive one or more supported profiles.
    - `entries_ops_rates` profile – includes both insertion and deletion rates KPIs

> **Info**
>
> Currently, this is the only supported profile.

The software KPIs profile can be set from the CLI by adding `--sw-profile`, for example:

```
doca_flow_tune monitor -f /tmp/flow_tune_cfg.json --sw-profile
entries_ops_rates
```

Analyze Mode

Analyze mode gathers (and later analyzes) information in order to assist users to better understand and debug their DOCA-Flow-based program.

Pipeline Export

This tool export an internal state of the DOCA-Flow-based program in a proprietary JSON format. This allows the tool to provide offline information about a given program which

can be later be analyzed. One such example is the ability to visualize the pipeline of the target program without having said program run on real hardware.

While the pipeline export operation is meant to encode all relevant information for future analysis, the format itself is proprietary and is only meant to be consumed by other DOCA tools.

Visualize Mode

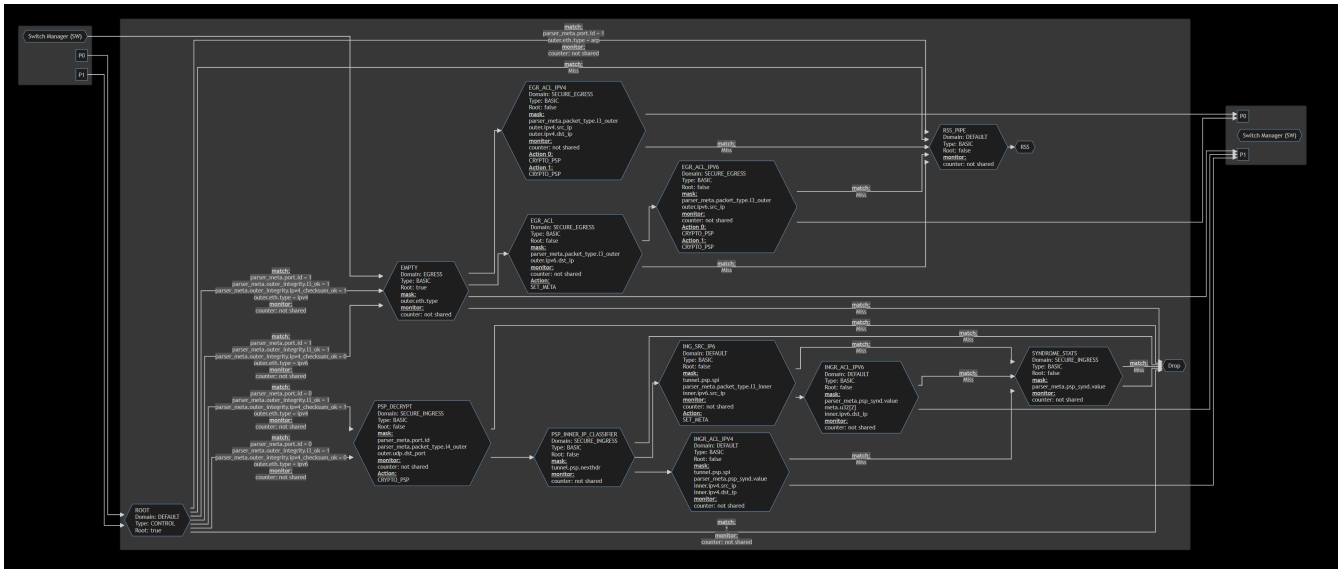
Viewing the Pipeline

After running DOCA Flow Tune tool with the `visualize` command, an output Mermaid file is generated. The content of the file is the raw format for the markdown markup language (known as "Mermaid" format). This is a widely supported industry standard for visualization purposes. An example for an online graphical editor for the Mermaid format is the Mermaid Live website.

> **Note**
>
> Copy and paste the content of the Mermaid file into the online editor to be able to view the visualization pipeline of your program.

Reading the Visualization

As can be shown in the following example, there are three "subgraphs" (gray background) in the visualized graph:



Input Graph

- This is the left-most subgraph
- Nodes in this section represent a possible start for packet flow (i.e., specific port identifier)
- The next hop in the graph is the DOCA Flow root pipe, as shown in the pipeline graph

Pipeline Graph

- This is the middle subgraph
- The subgraph holds all pipes defined by the target program:
  - Nodes represent a **DOCA Flow pipe** – each node shows the relevant pipe's attributes as defined by the program (using DOCA Flow API terminology). For example, attributes include the pipe name, pipe type, match items, etc.
  - The next hop in the graph can either be another pipe or a terminating action. Terminating action may be a drop, RSS, forward to port, or any other action that terminates the packet's flow as defined by the program.
  - **Control pipe links** – the illustration shows links which are egressing from the **control pipe** indicating the entry match item used to draw this link, including additional relevant information
  - **None-control pipe links**:

- For non-control pipes, the match items are present at the node's level as all entries share the same match items
- Different links represent different actions taken (e.g., different forward-pipe attributes)

Output Graph

- This is the right-most subgraph
- This subgraph holds the ports and switch manager nodes
- Links that arrive to this layer represent a packet going to wire (for ports) or to software (for switch manager)

Troubleshooting

Telemetry fwctl driver is not loaded

Error

When running the DOCA Flow Tune in Monitor mode, the following log messages are encountered at startup:

```
[DOCA][WRN][priv_doca_telemetry_fwctl.cpp:121]
[priv_doca_telemetry_fwctl_find_device_by_pci] Failed finding
fwctl device: Opening directory /sys/class/fwctl/ failed. Make sure
you have the fwctl driver loaded
[DOCA][ERR][priv_doca_telemetry_fwctl.cpp:201]
[priv_doca_telemetry_fwctl_open_by_devinfo] devinfo 0x55c572286520:
Failed to open fwctl device: Failed to find matching fwctl device
```

Solution

The DOCA Telemetry SDK uses the `fwctl` driver to query the hardware counters, so it is essential to have it installed and loaded.

Step 1: Verify the Driver Installation

First, check if the driver is installed as follow:

- Debian/Ubuntu:

```
$ sudo apt list --installed | grep fwctl
```

- RHEL:

```
$ sudo yum list installed | grep fwctl
```

If the driver is not installed, install it by running the following commands:

- Debian/Ubuntu:

```
$ sudo apt search fwctl
>> <fwctl-package-name>/....

$ sudo apt install -y <fwctl-package-name>
```

- RHEL:

```
$ sudo yum search fwctl
>> <fwctl-package-name>/....

$ apt/yum install -y <fwctl-package-name>
```

Step 2: Check if the Driver is Loaded

After installing the driver, verify that it is loaded by executing:

```
$ sudo lsmod | grep fwctl
```

You should see output similar to:

```
> mlx5_fwctl 20480 0
> fwctl 16384 1 mlx5_fwctl
> mlx5_core 2134016 2 mlx5_fwctl,mlx5_ib
> mlx_compat 69632 14
rdma_cm,ib_ipoib,mlxdevm,mlxfw,mlx5_fwctl,iw_cm,ib_umad,fwctl,ib_c
```

If the driver is not loaded, load it by running:

```
$ sudo modprobe mlx5_fwctl
```

Mermaid visualization in Visual Studio Code

Visual Studio Code provides extensions to view Mermaid markdown format, these extensions can be used to view the Mermaid output from DOCA Flow Tune tool.

However, for these extension to work, the Mermaid file should be modified with Mermaid opening and closure lines as follows:

```
```mermaid
```
```

```
<original_mermaid_file_content>
```
```

# Limited feature set – could not detect a running DOCA Flow program

## Error

When running DOCA Flow Tune, the following log message is encountered at startup, followed by some features failing to work/load:

```
[DOCA][WRN][flow_tune.cpp:195][get_flow_app_data] Could not detect a running DOCA Flow program, some features will be impacted
```

Solution

Some features of DOCA Flow Tune work by interacting with a live DOCA-Flow-based program. This is enabled through a server running in the background as part of the DOCA Flow library, and requires all of the following to be applied:

- DOCA-Flow-based program should explicitly enable the server. More information is available in the DOCA Flow Tune Server programming guide.
- DOCA-Flow-based program should run using the "trace enabled" DOCA Flow library. More information is available in the "Debug and Trace Features" section of the DOCA Flow programming guide.

Appendix – Configuration File Example

```
{
```
```

```
    "outputs_directory" :  "/tmp/flow_tune/",
    "network" :  {
        "server_uds" :  "/tmp/tune_server.sock",
        "uds_directory" :  "/var/run/doca/flow_tune/"
    },
    "csv" :  {
        "enable" :  false,
        "file_name" :  "flow_tune.csv",
        "max_size_bytes" :  1000000000,
        "max_files" :  1
    },
    "analyze" :  {
        "file_name" :  "flow_tune_pipeline_desc.json"
    },
    "visualize" :  {
        "pipeline_desc_file" :  "/tmp/flow_tune_pipeline_desc.json",     // Non-mandatory
field
        "file_name" :  "flow_tune_pipeline_vis.md"
    },
    "monitor" :  {
        "screen_mode" :  "light",
        "hardware" :  {
            "pci_addresses" :  [
                "08:00.0",
                "08:00.1"
            ],
            "profile" :  "full"
        },
        "software" :  [
            {
                "flow_port_id" :  0,
                "profiles" :  [
                    "entries_ops_rates"
                ]
            },
            {
```

```
                                        "flow_port_id" :  1 ,
                                        "profiles" :  [
                                                "entries_ops_rates"
                                        ]
                                }
                        ]
                }
}
```

Where:

- `outputs_directory` – Main directory on which all output products are saved. This field does not have a default value. If no value is provided, DOCA Flow Tune files are saved at the following directories:
  - CSV file – `/var/log/doca/flow_tune/`
  - Analyze export pipeline description file – `/tmp/flow_tune/`
  - Pipeline visualization file – `/tmp/flow_tune/`
- `network`
  - `server_uds` – DOCA Tune Server Unix Domain Socket (UDS) path. Default value is `/tmp/tune_server.sock`.
  - `uds_directory` – Directory on which all local UDS is created. Default value is `/var/run/doca/flow_tune/`.
- `csv`
  - `enable` – true if information should be saved into a CSV file. Default value is `false`.
  - `file_name` – CSV filename. Default value is `flow_tune.csv`.
  - `max_size_bytes` – CSV file maximum size in bytes. When the limit is reached, a new file is created. Default value is `1Gb`.

    - `max_files` – Maximum CSV files to create. Default value is `1`.

- `analyze`

    - `file_name` – Flow program pipeline description filename. File is created under `outputs_directory` path. Default value is `flow_tune_pipeline_desc.json`.

- `visualize`

    - `pipeline_desc_file` – Flow program pipeline description input file path. This file is the product of the `analyze export` command.

    - `file_name` – Flow program pipeline visualization filename. File is created under the `outputs_directory` path. Default value is `flow_tune_pipeline_vis.md`.

- `monitor`

    - `screen_mode` – Monitor command theme to be used. Default value is `light` .

    - `hardware`

        - `pci_addresses` – List of PCIe addresses which DOCA Flow Tune should inspect.

        - `profile` – Hardware profile to be used for each PCIe address given. Default value is `full`.

    - `software`

        - `flow_port_id` – Flow program port identification number which DOCA Flow Tune should inspect.

        - `profiles` – List of software profiles to be used for the specific port identification number given. Default value is `[entries_ops_rates]`.