NVIDIA DOCA Application Recognition

Application

Table of Contents

Chapter 1. Introduction

Application Recognition (AR) allows identifying applications that are in use on a monitored networking node.

AR enables the security administrator to generate consolidated reports that show usage patterns from the application perspective. AR is also used as a corner stone of many security applications such as L7-based firewalls.

Due to the massive growth in the number of applications that communicate over Layer 7 (HTTP), effective monitoring of network activity requires looking deeper into Layer 7 traffic so individual applications can be identified. Different applications may require different levels of security and service.

This document describes how to build AR using the deep packet inspection (DPI) engine, which leverages NVIDIA® BlueField®-2 DPU capabilities such as regular expression (RXP) acceleration engine, hardware-based connection tracking, and more.

Chapter 2. System Design

The AR application is designed to run as "bump-on-the-wire" on the BlueField-2 instance, it intercepts the traffic coming from the wire, and passes it to the Physical Function (PF) representor connected to the host.
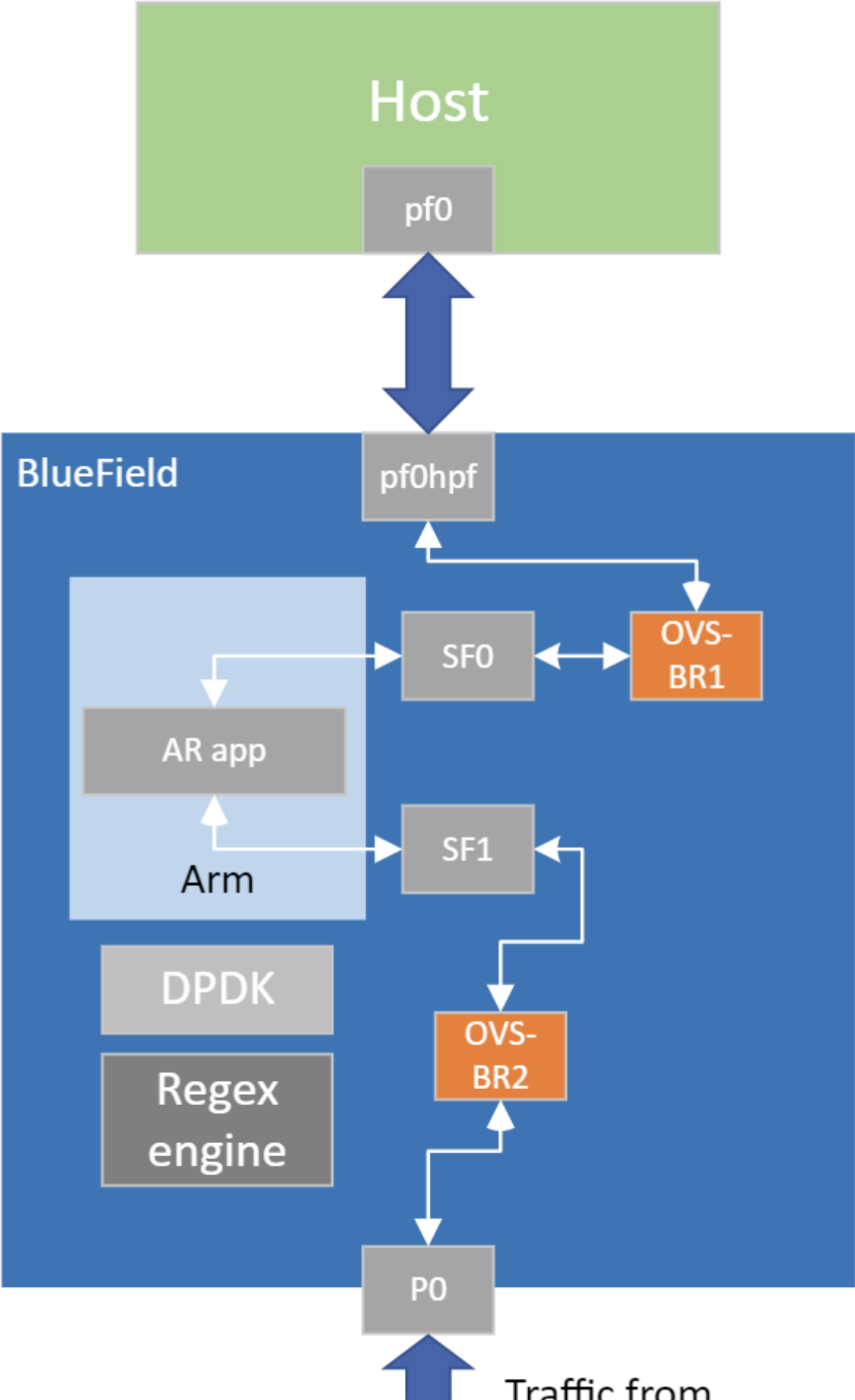
Chapter 3. Application Architecture

AR runs on top of Data Plan Development Kit (DPDK) based Stateful Flow Tracking (SFT) to identify the flow that each packet belongs to, then uses DPI to process L7 classification.
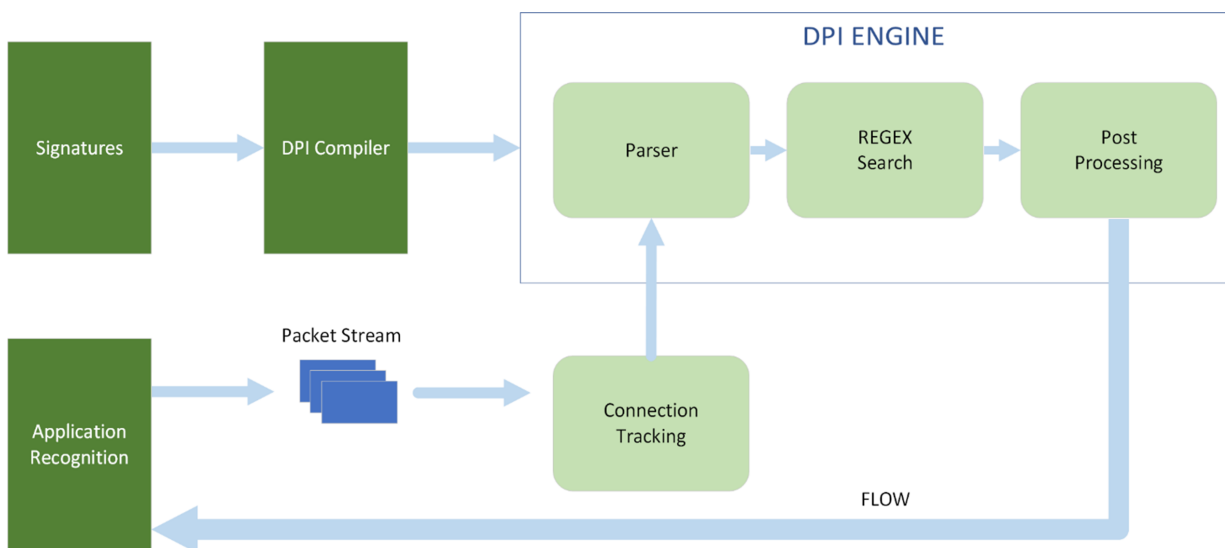
1. Signatures are compiled by DPI compiler and then loaded to DPI engine.
2. Ingress traffic is identified using the stateful table module in the DPDK libs which utilizes the connection tracking hardware offloads. This allows flow classifications to be done in the hardware level and be forwarded to the hairpin queue without being processed by the software, which increases performance dramatically.
3. Traffic is scanned against DPI engine compiled signature DB.
4. Post processing is performed for match decision.
5. Matched flows are identified, and actions can be offloaded to the hardware to increase performance as no further inspection is needed.
6. Flow termination is done by the aging timer set in the SFT to 60 seconds. When a flow is offloaded it cannot be tracked and destroyed.

Chapter 4. DOCA Libraries

This application leverages the following DOCA libraries:

▶ DOCA DPI library

▶ DOCA Telemetry library

Chapter 5. Configuration Flow

1. Parse application argument.
   ```
   doca_argp_init();
   ```
   a). Initialize arg parser resources and register DOCA general parameters.
   ```
   doca_argp_init();
   ```
   b). Register AR application parameters.
   ```
   register_ar_params();
   ```
   c). Parse the arguments.
   ```
   doca_argp_start
   ```
   i.   Parse DPDK flags and invoke the handler for calling the `rte_eal_init()` function.
   ii. Parse app flags.

2. DPDK initialization.
   ```
   dpdk_init();
   ```
   Calls `rte_eal_init()` to initialize EAL resources with the provided EAL flags.

3. DPDK port initialization and start.
   ```
   dpdk_queues_and_ports_init();
   ```
   a). Initialize SFT.
   b). Initialize DPDK ports, including mempool allocation.

4. AR initialization.
   ```
   ar_init();
   ```
   a). Initialize signature database.
   b). Initialize DPI engine.
   c). Load signatures to DPI.

5. Configure DPI packet processing.
   ```
   dpi_worker_lcores_run();
   ```
   a). Configure DPI enqueue packets.
   b). Send jobs to RegEx engine.
   c). Configure DPI dequeue packets.

6. Send statistics and write database.
   ```
   sig_database_write_to_csv();
   send_netflow_record();
   ```
   a). Send statistics to the collector.
   b). Write CSV file with signature statistics.

7. AR destroy.
```
ar_destroy();
```

 a). Clear thread.

 b). Stop DPI worker.

 c). Stop DOCA DPI.

8. DPDK ports and queues destruction.
```
dpdk_queues_and_ports_fini();
```
9. DPDK finish.
```
dpdk_fini();
```

Calls `rte_eal_destroy()` to destroy initialized EAL resources.

10. DPI destroy
```
doca_dpi_destroy();
```
11. Arg parser destroy.
```
doca_argp_destroy()
```

Chapter 6. Running Application

1. Refer to the following documents:

   ▶ NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField-related software
   ▶ NVIDIA DOCA Troubleshooting Guide for any issue you may encounter with the installation, compilation, or execution of DOCA applications
   ▶ NVIDIA DOCA Applications Overview for additional compilation instructions and development tips regarding the DOCA applications

2. The application recognition binary is located under `/opt/mellanox/doca/applications/application_recognition/bin/doca_application_recognition`. To build all the applications together, run:
   ```
   cd /opt/mellanox/doca/applications/
   meson build
   ninja -C build
   ```

3. To build only the application recognition application:

   a). Edit the following flags in `/opt/mellanox/doca/applications/meson_option.txt`:

   ▶ Set `enable_all_applications` to `false`
   ▶ Set `enable_application_recognition` to `true`

   b). Run the commands in step 2.

   > 📝 **Note:** `doca_application_recognition` will be created under `./build/application_recognition/src/`.

   Application usage:
   ```
   Usage: doca_application_recognition [DPDK Flags] -- [DOCA Flags] [Program Flags]
   DOCA Flags:
     -h, --help                Print a help synopsis
     -v, --version             Print program version information
     -l, --log-level           Set the log level for the program <CRITICAL=20,
    ERROR=30, WARNING=40, INFO=50, DEBUG=60>
   Program Flags:
     -p, --print-match         Prints FID when matched in DPI engine
     -n <source_id>, --netflow  exports data from BlueField to remote DOCA Telemetry
    service, also sets source_id to be written to the Netflow packet.
     -i, --interactive         Adds interactive mode for blocking signatures
     -o, --output-csv <path>   Path to the output of the CSV file
     -c, --cdo <path>          Path to CDO file compiled from a valid PDD
   ```

```
-f, --fragmented              Enables processing fragmented packets
```

> 📝 **Note:** For additional information on available flags for DPDK, use `-h` before the `--` separator:
> ```
> /opt/mellanox/doca/applications/application_recognition/bin/
> doca_application_recognition -h
> ```

> 📝 **Note:** For additional information on the application, use `-h` after the `--` separator:
> ```
> /opt/mellanox/doca/applications/application_recognition/bin/
> doca_application_recognition -- -h
> ```

4. Running the application on BlueField:

   ▶ Pre-run setup:

   a). The application recognition example is based on DPDK libraries. Therefore, the user is required to provide DPDK flags, and allocate huge pages.
   ```
   sudo echo 2048 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
   ```

   b). Make sure the RegEx engine is active:
   ```
   systemctl status mlx-regex
   ```

   If the status is inactive (`Active: failed`), run:
   ```
   systemctl start mlx-regex
   ```

   c). To use the supplied signature file (`suricata_rules_example`), which is installed to the bin directory, the DPI compiler must be used, as the RegEx engine accepts only `.cdo` files. The CDO files are constructed by compiling a signature file written in the Suricata open-source format.

   Compile the signature file:
   ```
   doca_dpi_compiler -i /opt/mellanox/doca/applications/
   application_recognition/bin/ar_suricata_rules_example -o /tmp/ar.cdo -f
    suricata
   ```

   A `.cdo` is created in the output path flagged as the `-o` input path of the compiler. That file can be used when executing the application using the `-c` flag.

   ▶ CLI example for running the app:
   ```
   /opt/mellanox/doca/applications/application_recognition/bin/
   doca_application_recognition -a 0000:03:00.0,class=regex -a
    auxiliary:mlx5_core.sf.4,sft_en=1 -a auxiliary:mlx5_core.sf.5,sft_en=1 -- -
   c /tmp/ar.cdo -p
   ```

> 📝 **Note:** The SFT supports a maximum of 64 queues. Therefore, the application cannot be run with more than 64 cores. To limit the number of cores, run:
> ```
> /opt/mellanox/doca/applications/application_recognition/bin/
> doca_application_recognition -a 0000:03:00.0,class=regex -a
>  auxiliary:mlx5_core.sf.4,sft_en=1 -a auxiliary:mlx5_core.sf.5,sft_en=1
>  -l 0-64 -- -p
> ```
> This limits the application to 65 cores (core-0 to core-64) with 1 core for the main thread and 64 cores to serve as workers.

> 📝 **Note:** The flags `-a 0000:03:00.0,class=regex -a auxiliary:mlx5_core.sf.4,sft_en=1 -a auxiliary:mlx5_core.sf.5,sft_en=1` are necessary for proper usage of the application. Modifying them results in

> unexpected behavior as only 2 ports are supported. The subfunction number is arbitrary and configurable. The RegEx device, however, is not and must be initiated on port 0.

> 💬 **Note:** Sub-functions must be enabled according to the Scalable Function Setup Guide.

5. Running the application on the host, CLI example:

```
doca_application_recognition -a 0000:04:00.0,class=regex -a 04:00.3 -a 04:00.4 -v
 -- -c suricata_rules_example.cdo -o /tmp/check.csv -p
```

> 💬 **Note:** Refer to section "Running DOCA Application on Host" in NVIDIA DOCA Virtual Functions User Guide.

6. To run `doca_application_recognition` using a JSON file:

```
doca_application_recognition --json [json_file]
```

For example:

```
cd /opt/mellanox/doca/applications/application_recognition/bin
./doca_application_recognition --json ./ar_params.json
```

The application periodically dumps a `.csv` file with the recognition results containing statistics about the recognized apps in the format `SIG_ID`, `APP_NAME`, `MATCHING_FIDS`, and `DROP`.

As per the example above, a file called `ar_stats.csv` will be created.

Additional features can be triggered by using the shell interaction. This allows blocking and unblocking specific signature IDs using the following commands:

▶ `block <sig_id>`

▶ `unblock <sig_id>`

The TAB key allows autocompletion while the `quit` command terminates the application.

NetFlow collector UI example:
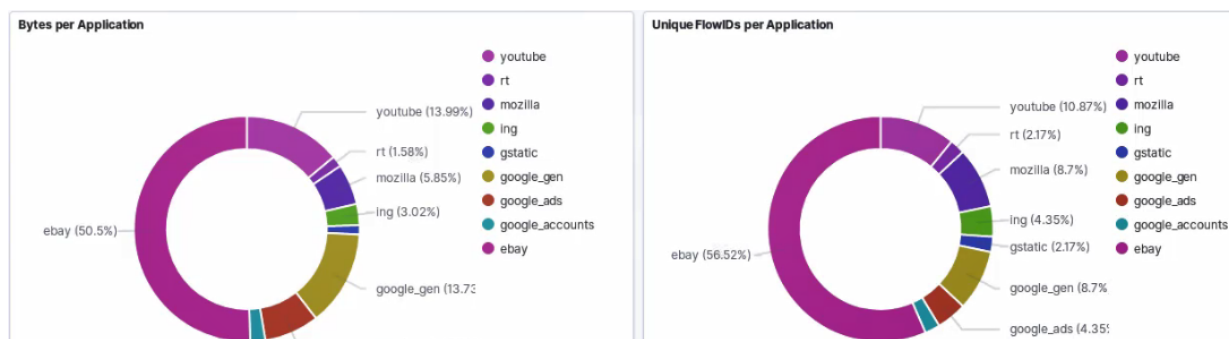
The NetFlow module uses the DOCA Telemetry Netflow library to export NetFlow packets in the NetFlow v9 format. The usage of telemetry is hardcoded to send packets to a collector set on the host connected to the Bluefield device through the rshim interface (specifically 192.168.100.2:2055).

It is recommended to use the DOCA telemetry service (DTS) as an aggregator to export records instead of exporting directly from the client side which requires enabling IPC. Refer to the NVIDIA DOCA Telemetry Service Guide guide for additional information.

Chapter 7. Arg Parser DOCA Flags

Refer to [NVIDIA DOCA Arg Parser User Guide](#) for more information.

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content

DPDK flags	a	devices	Adds a PCIe device into the list of devices to probe	`"devices":`
				`[`
				`  {"device": "regex", "id": "00`
				`  {"device": "sf", "id": "4","s`
				`  true},`
				`  {"device": "sf", "id": "5","s`
				`  true},`
				`]`

	l	core-list	List of cores to run on	`"core-list": "0-4"`
General flags	l	log-level	Sets the log level for the application: ▶ CRITICAL=20 ▶ ERROR=30 ▶ WARNING=40 ▶ INFO=50 ▶ DEBUG=60	`"log-level": 60`
	v	version	Prints program version information	N/A
	h	help	Prints a help synopsis	N/A
Program flags	p	print-match	Prints FID when matched in DPI engine	`"print-match": true`
	n	netflow	Exports data from BlueField to remote DTS. The IP is set to 192.168.100.2 which is the	`"netflow": 0`

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
			host's IP using the RShim interface. Also sets `source_id` to be written to the NetFlow packet.	
	i	interactive	Adds interactive mode for blocking signatures	`"interactive": false`
	o	output-csv	Path to the output of the CSV file	`"output-csv": "/tmp/ar_stats.csv"`
	c	cdo	Path to CDO file compiled from a valid PDD 📝 **Note:** This is mandatory flag.	`"cdo": "/tmp/ar.cdo"`
	f	fragmented	Enables processing fragmented packets	`"fragmented": false`

Chapter 8. Deploying Containerized Application

The application recognition example supports a container-based deployment:

1. Refer to the NVIDIA DOCA Container Deployment Guide for details on how to deploy a DOCA container to the BlueField.
2. Application-specific configuration steps can be found on NGC under the application's container page.

Chapter 9. Managing gRPC-Enabled Application from Host

Refer to [NVIDIA DOCA gRPC Infrastructure User Guide](#) for instructions on running the gRPC application server on the BlueField.
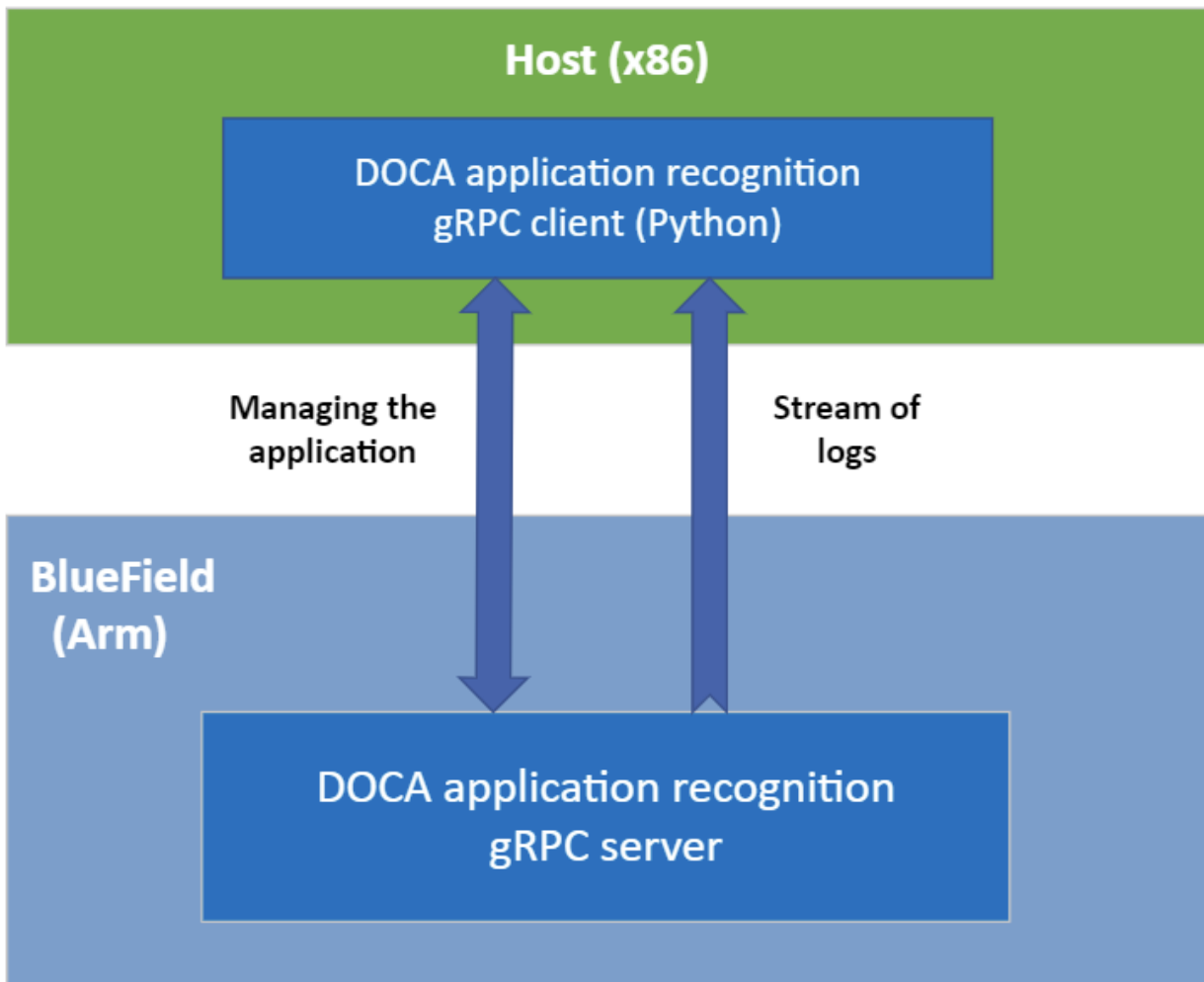
To run the Python client of the gRPC-enabled application:

```
./doca_application_recognition_gRPC_client.py -d/--debug <server address[:server
 port]>
```

For example:

```
/opt/mellanox/doca/applications/application_recognition/bin/grpc/client/
doca_application_recognition_gRPC_client.py 192.168.104.2
```

Chapter 10. References

▶ `/opt/mellanox/doca/applications/application_recognition/src/application_recognition.c`

▶ `/opt/mellanox/doca/applications/application_recognition/src/grpc/application_recognition.proto`

▶ `/opt/mellanox/doca/applications/application_recognition/bin/ar_suricata_rules_example`