Technology Guide

FD.io VPP - Generic Flow Offloading Overview

Authors

Ting Xu

Qi Zhang

Ping Yu

1 Introduction

With rising Ethernet speeds of 100 Gb, 200 Gb, and even 400 Gb combined with the evolving requirements in communication and cloud, offloading flow-based packet filtering from the CPU to the network controller becomes a must-have feature for network adapters.

To offer the flexibility to accelerate different workloads in a single hardware, modern network adapters provide a programmable packet processing pipeline increasing the complexity of driver development. Sometimes, the software teams must deliver hundreds of offloading features on variant hardware configurations in a single driver. It becomes challenging for the team to provide reliable software at par with the customers' expectations.

Providing a generic hardware offloading API is the current state-of-the-art framework for network drivers. However, this function requires the driver to maintain mapping between named protocols and hardware representations. Once the hardware is reconfigured due to workload changes, you may need to modify the driver to update the mapping, which requires considerable software engineering effort.

This document introduces a protocol agnostic flow offloading method named "Generic Flow" for the open source FD.io VPP framework. The Generic Flow-enabled driver learns the network adapter's hardware at runtime and translates flow offloading requests to the specific hardware configure commands. This learning mechanism keeps the driver unchanged even though the hardware has been reconfigured.

Generic Flow is now available on the Intel® Ethernet 800 Series Network Adapters. This documentation introduces how to use Generic Flow with Intel® Ethernet 800 Series Network Adapters to recognize the offload benefits. It is intended for network driver developers, especially those working on network adapter hardware flow offloading.

This document is part of the Network Transformation Experience Kits.

Table of Contents

Figures

Tables

Document Revision History

Revision	Date	Description
001	April 2023	Initial release.

1.1 Terminology

Table 1. Terminology

Abbreviation	Description
API	Application Programming Interface
CAM	Content Addressable Memory
COMMS	Telecommunication
CPU	Central Processing Unit
DDP	Dynamic Device Personalization
FD.io	The Fast Data Project
GRE	Generic Routing Encapsulation
GTP	General Packet Radio System (GPRS) Tunneling Protocol
GTPoGRE	GTP over GRE
JSON	JavaScript Object Notation
PPPoE	Point to point over Ethernet
VAPI	VPP API
VLAN	Virtual Local Area Network
VPP	Vector Packet Processing

1.2 Reference Documentation

Table 2. Reference Documents

Reference	Source
DPDK RTE_FLOW API	https://doc.dpdk.org/guides/prog_guide/rte_flow.html
Intel® Ethernet Controller E810 Dynamic Device Personalization (DDP) Technology Guide	https://www.intel.com/content/www/us/en/content-details/617015/intel-ethernet-controller-e810-dynamic-device-personalization-ddp-technology-guide.html
VPP Wiki	https://wiki.fd.io/view/VPP

2 Overview

This document is intended to show how Generic Flow can significantly reduce the effort to support new protocols and update the mapping between named protocols and hardware offloading API.

The document introduces the key idea of Generic Flow – A driver with learning mechanism and the workflow of Generic Flow in VPP. It describes the method to create Generic Flow commands and use them to add or delete flow rules to the network adapter hardware with the VPP interface.

2.1 Challenges Addressed

To meet the rapidly evolving requirements of the communication and cloud market, a programmable packet processing pipeline has been introduced in modern network adapters. However, high flexibility leads to more challenges:

- Different protocols are needed according to new use cases, hence the driver would need to be modified each time to support hardware offloading. It is a considerable effort for software engineers to maintain the driver with repetitive work.
- VPP flow interface and applications based on VPP flow API would have to be updated regularly to adapt to the change of drivers, which blocks the development and causes additional cost to the users.

With Generic Flow, the driver can parse every packet type as long as the hardware supports it. The driver can automatically learn the newly added protocols in hardware. Those users of VPP do not need to wait for the driver to support new protocols or the specific packet type they need. The only thing that the user should do is to create a flow rule based on the Generic Flow specification.

2.2 Technology Description

2.2.1 Intel® Ethernet 800 Series Network Adapters and Dynamic Device Personalization

Intel® Ethernet 800 Series Network Adapters is a new generation network adapter. It adopts an enhanced programmable packet processing engine, which provides a deeper and more flexible protocol headers processing capability. Such capability is based on Dynamic Device Personalization (DDP). Every Intel® Ethernet 800 Series Network Adapter can load a DDP package dynamically to achieve diverse packet processing hardware offloading. The device driver will read the binary data from the package file and program them into the device's registers, CAMs, or Memory through specific firmware commands.

The OS Default DDP package is installed on all supported Intel® Ethernet 800 Series Network Adapter Kernel and DPDK drivers in most common operating systems. OS default DDP package provides basic flow hardware offloading capability, including MAC, IP, TCP/UDP, VLAN, GRE, and so on.

In addition to the default DDP package, Intel provides COMMS DDP packages designed for specific workloads. For example, besides the basic protocols supported by the OS default package, a COMMS package for 5G service providers also supports GTP and PPPoE. For other cases, COMMS packages can help even more protocols.

The DPDK RTE_FLOW can be used with the DDP package to utilize the hardware offloading of Intel® Ethernet 800 Series Network Adapter, including flow filter and RSS. Based on it, we can make the driver learn the parsing logic in DDP, create a parser emulator to derive training packets, profile, and field vectors, then program them to the hardware directly, without complicated software modification.

2.2.2 Protocol Agnostic Flow Offloading

DPDK provides RTE_FLOW as a generic flow offloading API to meet the programmable pipeline requirement. The RTE_FLOW has a set of pre-defined data structures representing protocol headers with fields that follow industry specs. A driver is supposed to provide a parser to translate the RTE_FLOW format protocol combinations into hardware configuration. However, if we want to support a new protocol, new data structures would need to be added to RTE_FLOW. Since the driver parser covers limited cases, new parser logic would also need to be added to support the new packet types even if the protocols exist.

To solve this problem, Protocol Agnostic Flow Offloading is put forward. It is a DPDK feature to use the RTE_FLOW raw pattern type. It will create a parser named Parser Emulator. Parser Emulator is based on DDP. The DDP package contains the parser logic that decides the packet's type, each protocol's offset, and some flags. Parser Emulator will learn the parser logic, parse on a packet buffer, mask buffer to generate the flow configurations, and download those configurations to hardware directly. Figure 1 shows the difference between traditional flow offloading and Protocol Agnostic Flow Offloading. Figure 2 gives an example to explain how the Parser Emulator works to parse the packet and mask buffer and translate them into hardware configuration.

The DDP package can be changed to support other protocols' offloading, and Parser Emulator will learn the new logic by itself. This way, users can use commands for new protocols directly without any driver modification.

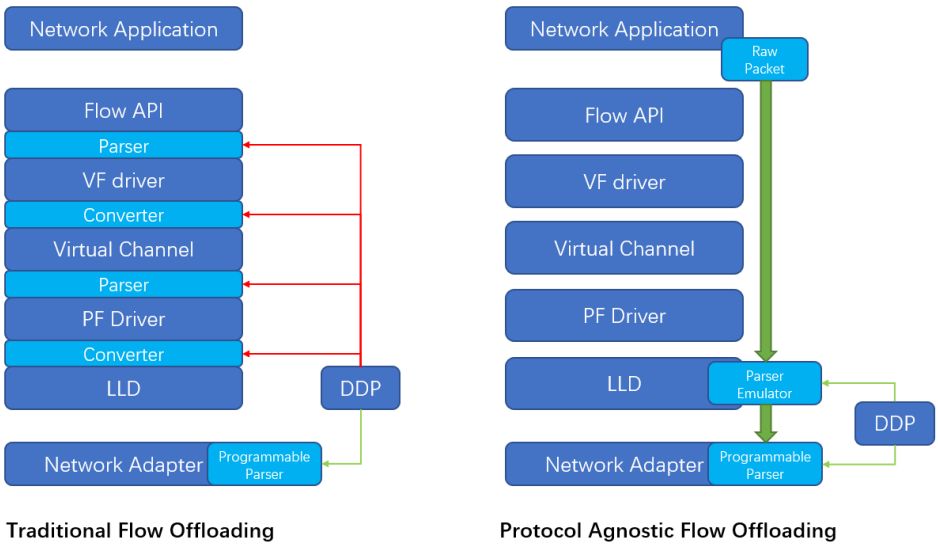

Figure 1. Traditional flow offloading process and Protocol Agnostic Flow Offloading

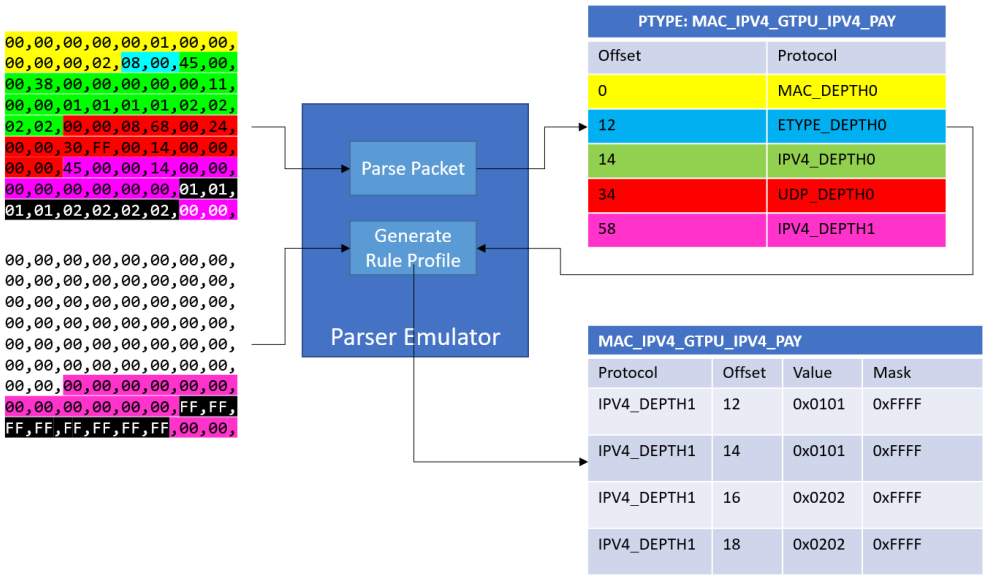

Figure 2. An example of how Parser Emulator works

2.2.3 Generic Flow

Generic Flow is a feature in VPP vnet flow component. It provides a generic interface for VPP to create flow rules. Generic Flow is based on Protocol Agnostic Flow Offloading. Therefore, it should be used together with Intel® Ethernet 800 Series Network Adapters. Traditionally, VPP vnet flow has the same issue: the flow rule creation interface needs to be modified to parse new protocols, even though the hardware has already supported those cases.

Generic Flow uses the current VPP vnet/flow framework, and its CLI and VAPI command. New flow type "VNET_FLOW_TYPE_GENERIC" and data structures are added for Generic Flow input parameters. The input parameters are packet and mask buffers, named "spec" and "mask," instead of the pre-defined protocols' fields in the original commands. They will be passed to DPDK plugins and wrapped as raw packet types for RTE_FLOW. Figure 3 shows the whole process. Generic Flow will pass "spec" and "mask" to hardware and rules not supported by DDP will be rejected.

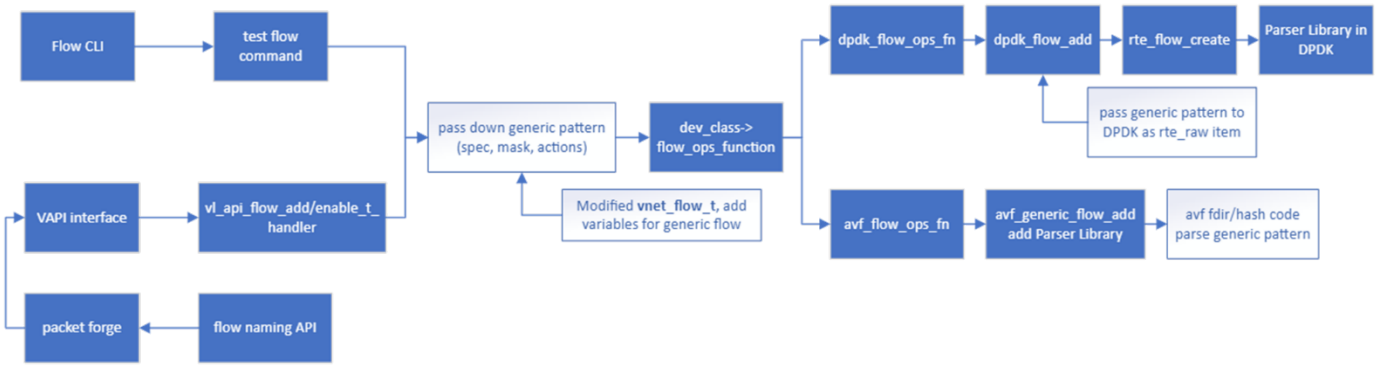

Figure 3. Process of Generic Flow in VPP

2.2.4 Packetforge

Packetforge is a tool to assist Generic Flow. The inputs of Generic Flow are packet and mask buffers, whose format is a binary string. However, it is hard for users to create a binary string format command alone. Packetforge is designed to provide a user-friendly interface that helps users to generate Generic Flow rules with naming format commands or JSON profiles.

Packetforge is a parse graph consisting of nodes and edges. Nodes stand for protocols, including fields, size, and default values. Edges are actions between two protocols; for example, an edge between Ethernet and IPv4 will set the Ethernet type to 0x800. Users can build customized parse graphs according to the specification of the Packetforge. In Packetforge, naming format commands will be parsed to create a JSON profile first. Then, Packetforge will translate the JSON profile into Generic Flow input parameters (packet and mask buffers). Users can also use a JSON profile as a Packetforge input directly. Figure 5 shows an example of the JSON profile.

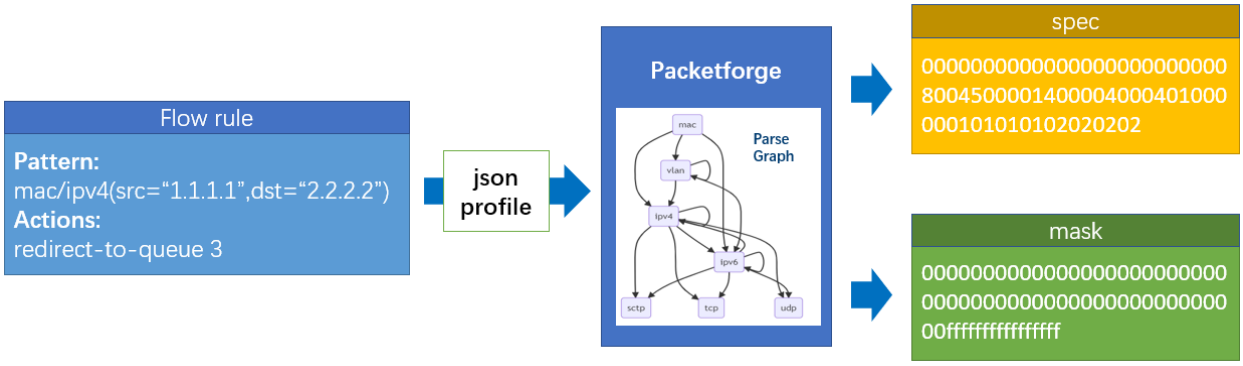

Figure 4. Process of Packetforge

```
{
   "type" : "path",
   "stack" : [
      {
         "header" : "mac"
      },
      {
         "header" : "ipv4",
         "fields" : [
            {
               "name" : "src",
               "value" : "1.1.1.1",
               "mask" : "255.255.255.255"
            },
            {
               "name" : "dst",
               "value" : "2.2.2.2",
               "mask" : "255.255.255.255"
            }
         ]
      }
   ],
   "actions" : "redirect-to-queue 3"
}
```

Figure 5. JSON profile example

3 Deployment

GTP over GRE (GTPoGRE) protocol are widely used in 5G UPF. However, VPP has not supported it yet in its current vnet/flow function. In the traditional VPP vnet/flow, to support GTPoGRE and create flow rules with hardware offloading, software should expose new API for the new protocol. All necessary fields for filter of a GTPoGRE flow must be considered.

In addition to accounting for the software development work, there is also a long development cycle. For VPP vnet/flow with DPDK plugins, software engineers would need to add the support for GTPoGRE in both DPDK and VPP and wait for the next formal release to merge the codes into master repositories. After that, users can create flow rules of GTPoGRE for their business logic. However, the development cycle usually takes several months, which is inconvenient.

With Generic Flow, the whole process is straightforward. As long as the hardware supports GTP and GRE protocols, there is no need to modify the software anymore. Users can immediately create flow rules for GTPoGRE in VPP vnet/flow with Generic Flow commands.

One standard packet of GTPoGRE in the 5G UPF scenario is shown below; it is an Uplink GTPU over GRE packet:

- Ether()/IP()/GRE()/Ether()/IP()/UDP(dport=2152)/GTPU()/GTPExtensionHeader(type=1)/IP(src="192.168.10.10",dst="192.168.1.3")/UDP(sport=23)

Traditionally, since VPP does not support the GRE header, GTP extension header, and its field "PDU type," if users want to create a flow rule for this packet, they should wait for the modification of VPP vnet/flow. They also need to verify that the DPDK driver supports it. However, with Generic Flow, users only need to use the below command:

```
test flow add index 0

spec
00000000000100000000000208004500006a00000000002f00000101010102020202000000000000000
00001000000000000208004500004400000000001100000101010102020202000008680030000024ff002
000000000850110000045000001c0000000000110000c0a80a0ac0a801030017000000080000

mask
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000ffff00000000
000000000000000000000f0000000000000000000000000000ffffffffffffffffffff000000000000
```

Figure 6. An example of Generic Flow command

With Packetforge, users can create an even easier command:

```
python flow_create.py --add
--pattern
"mac()/ipv4()/gre()/mac()/ipv4()/udp(dst=2152)/gtpu()/gtppsc(pdutype=1)/ipv4(src="192.168.10.10",dst="192.168.1.3")/udp(src=23)"
--actions "redirect-to-queue 3"
--interface 1
```

Figure 7. An example of pattern format command

```
python flow_create.py --add
--file "JSON_FOLDER/gtpogre.json"
--interface 1
```

Figure 8. An example of JSON profile command

The second command uses JSON profile as input directly; in this case, refer to the Appendix for the profile example.

4 Summary

In summary, VPP 22.11 has support for Generic Flow and Packetforge. Users can now use these commands directly without any additional operation, and there is no need to wait for software modification. They can create rules immediately for their business logic.

Appendix A JSON Profile Example for GTPoGRE

This is the content of a JSON profile for GTPoGRE Generic Flow rule. See Figure 8 for an example of the JSON profile command.

```
{
  "type" : "path",
  "stack" : [
    {
      "header" : "mac"
    },
    {
      "header" : "ipv4"
    },
    {
      "header" : "gre"
    },
    {
      "header" : "mac"
    },
    {
      "header" : "ipv4"
    },
    {
      "header" : "udp",
      "fields" : [
        {
          "name" : "dst",
          "value" : "2152",
          "mask" : "0xffff"

        }
      ]
    },
    {
      "header" : "gtpu"
    },
    {
      "header" : "gtppsc",
      "fields" : [
        {
          "name" : "pdutype",
          "value" : "1",
          "mask" : "0xf"

        }
      ]
    },
    {
      "header" : "ipv4",
      "fields" : [
        {
          "name" : "src",
          "value" : "192.168.10.10",
          "mask" : "255.255.255.255"
        },
        {
          "name" : "dst",
          "value" : "192.168.1.3",
          "mask" : "255.255.255.255"
```

```
        }
      ]
    },
    {
      "header" : "udp",
      "fields" : [
        {
          "name" : "src",
          "value" : "23",
          "mask" : "0xffff"

        }
      ]
    }
  ],
  "actions" : "redirect-to-queue 3"
}
```

0423/DN/WIT/PDF 773854-001US