OPEN Compute Project®

HARNESSING PROGRAMMABLE SWITCH SILICON USING NPL – PACKET BROKER USE CASE

FEBRUARY 15TH 2022

AUTHORS: DANNY LOBO, RUSS ERIKSON, CLIFF LIN

NETSCOUT SYSTEMS, INC.

NETSCOUT

OPEN Compute Project

Table of Contents

Abstract

One of the key principles of OCP Networking has been the concept of disaggregation, where organizations can build their value-added layers on top of open platforms and software. This has led to the rapid proliferation of work in the Network Operating System (NOS) space over the last several years, and we see this continuing as companies see how more product lines can be based upon such open platforms.

In this white paper, we highlight Packet Brokers as the target network product category. Packet Brokers, including those from NETSCOUT, are traditionally built on top of closed, custom hardware that use FPGAs and require advanced packet conditioning features. Now, open, disaggregated platforms such as those in the OCP architecture enable us to build Packet Brokers on top of OCP switches with programmable merchant switching silicon. Specifically, NETSCOUT used the open NPL (Network Programming Language) to develop a Packet Broker without the use of FPGAs. We partnered with key vendors to successfully build a viable and working model on top of the NPL emulator. Our next step is to move our emulation on top of a hardware switch that support NPL, such as an existingTrident4-based OCP switch.

This project shows the impact that open, programmable OCP switches can have on the development of specialized devices such as Packet Brokers. They can be developed in a fraction of the time and built at a fraction of the cost as compared to closed, FPGA-based systems. They create a "multiplier effect" within the community by leveraging chips and systems that are already being deployed in the OCP networking community. We encourage other members of the community to see how their own product lines can benefit from leveraging the latest open and programmable OCP switches.

1 Introduction

NETSCOUT uses an Open Compute Project (OCP) switch platform for its Network Packet Broker. Packet brokers have traditionally been built with closed custom hardware. The use of merchant silicon switch chips with FPGAs enables advanced features, such as VLAN and MPLS header stripping, de-encapsulation of tunneling protocols, advanced load balancing, and conditional data masking. However, using custom hardware to build a packet broker has the same drawbacks as any custom hardware implementation. Creating custom switch hardware requires a large hardware design and manufacturing team, for example. Furthermore, the hardware design cycle is usually measured in years. Even when a custom packet broker takes advantage of merchant switching silicon, the delay from silicon introduction to final product completion can be very long. As a result, the end product may not be complete until after the next generation of switching silicon is available, putting custom hardware constantly a step behind the current state of the art.

Meanwhile, the rise of open networking and related efforts has resulted in open white box switches offered by various vendors, and OCP switch platforms allow for rapid adoption of the latest merchant switching silicon. Furthermore, combining OCP switch platforms with Open Network Install Environment (ONIE) compatible packet broker software can result in packet brokers that are both very cost-effective and able to take advantage of the latest developments in switching hardware technologies. However, the feature set of such OCP-based packet brokers is limited to the feature set of the underlying switch hardware. There is no provision for customization of the hardware to include features that may be unique to packet broker applications or were previously available through closed proprietary FPGA implementations.

While still using OCP switch platforms, one approach to deliver advanced packet broker features is to provide a server-based adjunct to the packet broker to offer advanced features via software running on the server. The server can either be a generic server platform, such as an Open Compute Project server, or a proprietary compute platform included in a proprietary chassis. Using a software-based adjunct processor allows a great degree of flexibility in the feature set offered. However, software-based solutions face extreme performance limitations. These performance limitations can be reduced by using more powerful processors -- with a higher number of cores -- but at a much greater cost.

Recently, merchant switching silicon includes mechanisms that allow customization of the switching functions via direct programming of the switching logic. In addition, switch chip vendors providing development environments for network programming languages such as P4 and NPL have opened the door to customization of the switching logic to allow for advanced packet broker features otherwise unavailable in the switch chips. This paper examines using NPL to implement examples of advanced packet broker features.

Table 1 - Advanced Packet Broker Feature Options

	Performance	Cost	Flexibility	Feature Set
Software Compute Based	Lowest	Highest	Highest	Full
FPGA Based	Medium to High	High	Medium	Full
Merchant Switch Silicon	Highest	Lowest	Low	Minimal
NPL Programmable Switch Silicon	High	Low	High	Nearly Full

NPL is an open high-level language for developing feature-rich solutions for programmable network platforms. NPL is used to program the behavior of packets as they are processed by the switch data plane, describing the behavior of the packets as they traverse a switch pipeline. NPL uses a combination of programmable pipeline tables and intelligent function primitives to describe this behavior. NPLang.org has provided a development emulation environment that allows for prototyping and testing NPL programs without a switch. Using this emulator, we can

illustrate NPL implementations of advanced packet broker features, including advanced header stripping, de-encapsulation, dynamic load balancing, and conditional data masking. We provide examples of actual code, using the emulator, so other members of the community can build on our examples and prove them out without requiring any investment in switching hardware.

2 Header Stripping

NPL can be used to remove headers from VLAN, MPLS, MPLS+VLAN, VN-Tag, VXLAN, ERSPAN, GTP-U, GRE, 6in4, 4in6, or Cisco FabricPath packets before they are forwarded to network tools or appliances. Removing these headers is useful when working with tools that either cannot recognize these headers or must engage in excessive additional processing to adjust for them.

2.1 VLAN

A traditional switch may have a limitation on handling traffic that contains more than 2 VLAN tags in the packet. Most packet brokers can only support 802.1Q and Q-in-Q VLAN headers. Similarly, a packet broker based on a traditional switch chip may only be capable of supporting a well-known Tag Protocol Identifier (TPID) like 0x8100 or 0x9100 – but not TPID (0x88A8) or a custom TPID. The non-NPL-based switch may not recognize or account for the additional tags to be able to analyze the traffic encapsulated inside VLANs. With NPL, a programmable switch would support multiple VLAN tags and configurable TPID, identifying the VLAN tags to be stripped or to load balance on the user session's layer 3 and layer 4 headers.

```
logical_register tpid_values {
    fields {
        bit[16] tpid0 = 0x8100;
        bit[16] tpid1 = 0x9100;
        bit[16] tpid2 = 0x88a8;

    }
}

parser_node otag {
    extract_fields(ingress_pkt.group1.otag);
    switch(latest.ethertype) {
        tpid_values.tpid1 : next_node itag;
        tpid_values.tpid2 : next_node itag;
        0x0800            : next_node ipv4;
        0x0000 mask 0xFC00: next_node snap_or_llc;
        0x0400 mask 0xFE00: next_node snap_or_llc;
        default           : next_node ingress;
    }
}
```

Figure 1 - TPID Example

In NPL, configurable TPID can be defined as a logical register, as shown. It can be assigned any custom TPID and as many values as desired. As an NPL-based switch is fully configurable, the number of VLAN tags in a stack is only limited by the number of resources committed to VLAN tag processing. The number of stacked VLAN tags supported is theoretically unlimited, but practically it would be the same as a typical PFGA based packet broker, up to about eight.

2.2 MPLS

In the case of MPLS, a tool may have been designed for handling traffic that contains only one MPLS label. In bridged or cross-provider networking where multiple MPLS labels can be present, the tool may not recognize or account for the additional labels to analyze the traffic encapsulated

inside MPLS. Most non-NPL-based switches can support a limited number of MPLS labels. An advanced feature of a packet broker is to support multiple MPLS label stripping to filter and load balance on the user session's layer 3 and layer 4 headers.

2.3 EtherType Replacement for MPLS Traffic

EtherType Replacement for MPLS traffic becomes very interesting when MPLS IPv4-IPv6 coexistence. Unlike VLAN traffic, the layer 3 protocol type is not carried in the packet for an MPLS packet. When MPLS stripping is enabled, most non-NPL switches have difficulty determining which EtherType to insert, including 0x0800 for IPv4 or 0x86dd for IPv6. Most non-NPL switches fail to support both IPv4 and IPv6 running over MPLS traffic coming to the same physical port. With NPL, it is very simple to examine the IP version and the byte following the MPLS header and determine whether the packet is IPv4/IPv6 and able to insert correct EtherType for outgoing packets.

3 Tunnel Header Stripping

Monitoring, analysis, and security are often developed for targeted specific applications, and the software and/or hardware implementations are not designed to handle certain protocols.

3.1 GRE

Generic Routing Encapsulation (GRE) is a tunneling protocol developed to encapsulate various network-layer protocols inside a virtual connection. Often, when a tool has not been specifically designed for handling GRE, it may not be able to either recognize GRE encapsulation or account for the added tunnel headers. This results in the tool being unable to analyze the traffic encapsulated inside GRE. However, an NPL-capable packet broker can be programmed to strip the GRE tunnel headers to allow the tool to analyze the original packet. Stripping the GRE tunnel headers consists of removing the outer IP header and the GRE header.

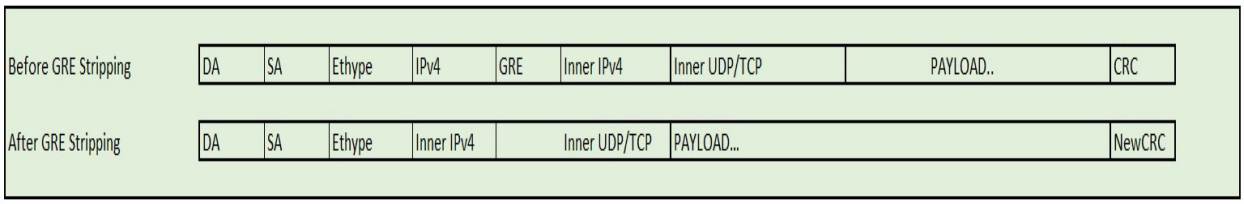


Figure 2 - GRE Stripping

Similarly, Network Virtualization using GRE (NVGRE) leverages GRE encapsulation to tunnel layer 2 packets over layer 3 networks. NVGRE adds an outer L2 header, an outer IP header, and a GRE header. To make the original packet available for analysis, an NPL-capable packet broker strips

all three headers.

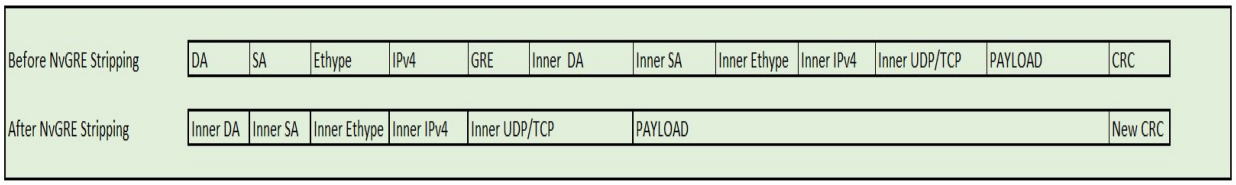


Figure 3 - NVGRE Striping

In NPL, the required L2/L3/GRE/L3/L4 headers are specified within ingress packets. During egress packet processing, the relevant outer headers are removed before forwarding to the egress port.

```
struct group1_t {
    fields {
        l2_t        ol2,
        vlan_t      otag;
        vlan_t      itag;
        }
}

struct group2_t {
    fields {
        ipv4_t      outer_ipv4,
        ipv6_t      outer_ipv6;
        }
}
```

```
struct group3_t {
    fields {
        gre_t       gre,
        l2_t        il2,
        }
}

struct group4_t {
    fields {
        ipv4_t      inner_ipv4,
        ipv6_t      inner_ipv6;
        tcp_t       inner_tcp;
        udp_t       inner_udp;
        }
}
```

```
struct ing_pkt_t {
    fields {
        group1_t group1;
        group2_t group2;
        group3_t group3;

        }
}

struct egr_pkt_t {
    fields {
        group1_t group1;
        group2_t group2;
        group3_t group3;
        group4_t group4;
    }
}
```

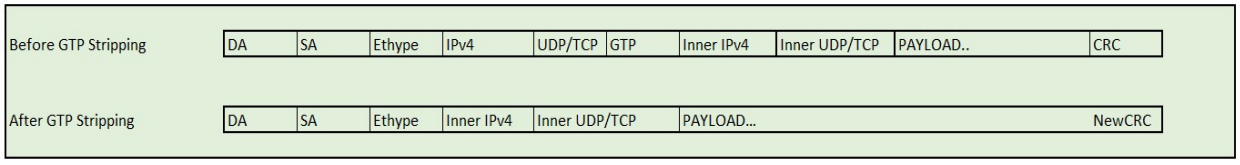
```
//GRE Case
delete_header(egr_pkt.group2);
delete_header(egr_pkt.group3);

//nvGRE Case
delete_header(egr_pkt.group1);
delete_header(egr_pkt.group2);
delete_header(egr_pkt.group3.gre);
```

Figure 4 – GRE/NVGRE Header Stripping Example

3.2 GTP

The GPRS (General Packet Radio Service) Tunneling Protocol (GTP) is defined by the 3GPP standards group to transport GPRS traffic inside a 3G/4G network. Any tool not explicitly designed to support cellular mobile communications networks may not recognize GTP or account for the added GTP headers. Therefore, these tools may not be able to analyze traffic encapsulated inside GTP. GTP-u packets are used to transport user data inside the core GPRS network. GTP tunnel stripping removes the header and trailer for GTP-u packets inside the GTP tunnel between the



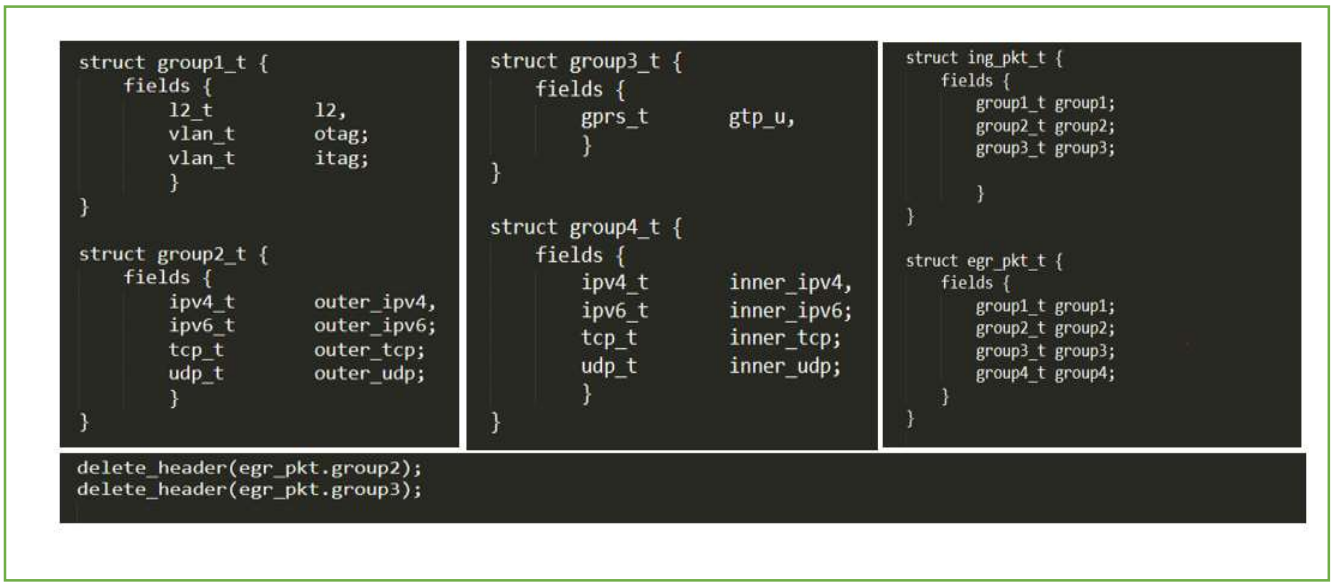


Figure 6 - GTP Header Stripping

SGSN and GGSN interfaces in a 3G network and between the eNodeB (eNb) and the SGW and between the SGW and the PGW in an LTE network.

For NPL to remove the GTP headers, specify the required L2/L3/GRE/L3/L4 headers within received packets. The L2/L3/L4 and GTP headers are removed before forwarding the packets to the egress port.

3.3 Generic Header Stripping

Generic protocol header stripping would remove any arbitrary headers from a packet. The headers are stripped based on the offset and the length of the header. Generic protocol stripping is one of the most powerful features of using NPL. NPL can be configured with header offset and length as pairs, allowing users to create multiple pairs of protocol headers that can be stripped from the same packet before sending to an egress port.

3.4 Load Balancing Limitation

Typically, load balancing is done based upon a 5-tuple derived from layer 3 and layer 4 header fields. Normal layer 3 and layer 4 flow-aware load balancing cannot account for the presence of additional headers between layers 2 and 3. This is particularly true when there are multiple tags or labels or encapsulation headers such as GRE or GTP-U. NPL can construct any type of protocol headers and custom headers based on the feature requirement. Therefore, an NPL-based switch can support any number of VLANs or MPLS tags, any encapsulation header, or any custom header when processing packets. Any combination of tags and headers can be stripped from the packet during processing from a user session's layer 3 and layer 4 and beyond without difficulty.

4 Conditional Data Masking

Data masking allows one or more fields within a packet to be hidden to protect sensitive information. It is instrumental in concealing information like IP addresses, credit card information, phone numbers, login names, or passwords. Data masking works by writing user-defined data over existing data in the packet, effectively hiding the data that was initially in the packet. This prevents any monitoring tools from storing or displaying sensitive information. This feature can be combined with filters allowing only packets that match the filter condition, typically something in the L2/L3/L4 headers, making the feature more flexible and powerful. The example below illustrates how L3 Data Masking can be done based on L2 (VLAN VID) as a conditional matching filter using NPL.

To correctly mask the data, the condition for masking the data must be provided along with the location of the data within that packet. Four major user configurations are required to support full data masking features.

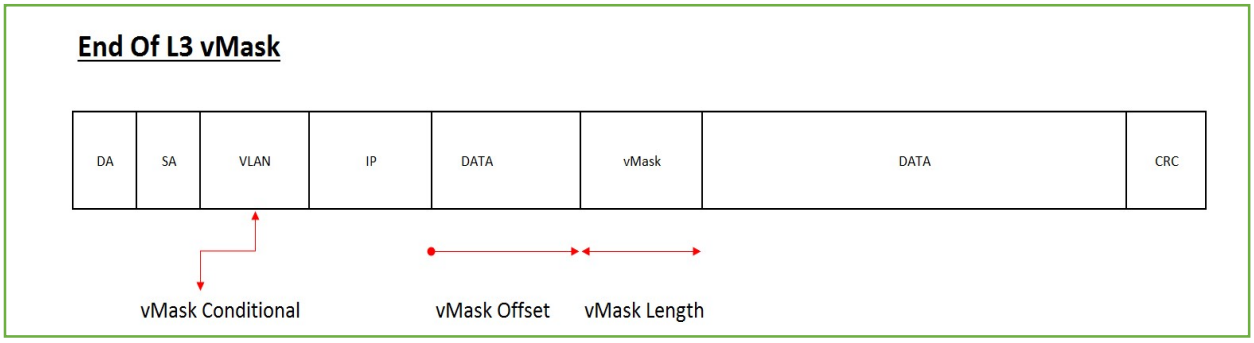


Figure 7 - Example of Data Masking Parameters

Data Mask Conditional Filter Configuration

Any headers field or combination of header fields (L2/L3/L4) can be used as a conditional filter. VLAN-VID is used here as an example.

Data Mask Offset Configuration

The data mask offset indicates where to look for the beginning of the data to be masked. The offset can be made relative to a header field. Here, the end of the L3 header is used.

Data Mask Length Configuration

The data mask length is the number of bytes that the user wants to overwrite.

Data Mask Pattern Configuration

The data pattern is the value to be written into the selected data area.

In NPL, the required L2/L3/L4 headers are specified within received packets. NPL will then examine those headers from the packet, based upon the filter matching condition, to modify the defined masking area during egress packet processing.

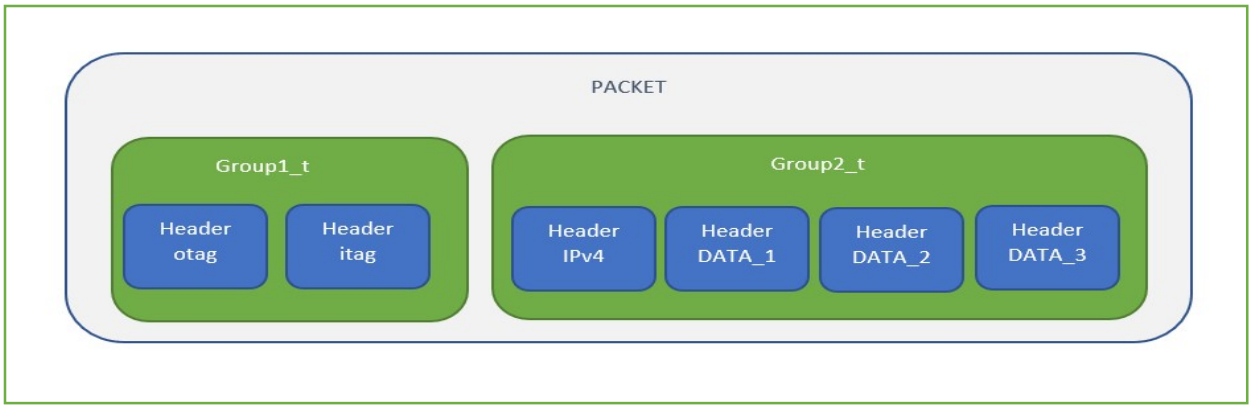


Figure 8 - Header Fields within the Packet

NPL defines the format of the headers using struct definitions.

```
struct group1_t {
    fields {
        vlan_t       otag;
        vlan_t       itag;
        }
}

struct group2_t {
    fields {
        ipv4_t         ipv4;
        data1_t        data1;
        data2_t        data2;
        data3_t        data3;
        }
}
```

```
struct ing_pkt_t {
    fields {
        group1_t group1;
        group2_t group2;
        }
}

struct egr_pkt_t {
    fields {
        group1_t group1;
        group2_t group2;
    }
}
```

Figure 9 - NPL Definition of Header Fields

NPL defines a table which based upon a lookup of the header, results in a set of outputs called the object bus. The object bus is broken into multiple fields which define the parameters for

performing the data masking: filter, pattern, offset, and length.

```
logical_table vlan_table {

    table_type : index;
    minsize    : 4096;
    maxsize    : 4096;

    keys {
        bit[12] vid;
    }

    fields {
        bit[12] vmask_vid;
        bit[64] vmask_pattern;
        bit[8]  vmask_offset;
        bit[8]  vmask_length;
        bit[1]  vmask_ena;
    }

    key_construct() {
        if (_LOOKUP0 == 1) {
            vid = obj_bus.vid;
        }
    }
 fields_assign() {
        if (_LOOKUP0 == 1) {
            obj_bus.port_vid = vmask_vid;
            obj_bus.vmask_pattern = vmask_pattern;
            obj_bus.vmask_offset = vmask_offset;
            obj_bus.vmask_length = vmask_length;
            obj_bus.vmask_match = vmask_ena;
        }
    }
```

```
function do_packet_modif() {
if (obj_bus.vmask_match) {
    if ((obj_bus.vmask_offset == 0x0) && (obj_bus.vmask_length == 0x8)) {
    replace_header_field(egress_pkt.group2.data1.data1, egr_obj_bus.vmask_pattern);
    } else if ((obj_bus.vmask_offset == 0x0) && (obj_bus.vmask_length == 0x16)) {
      replace_header_field(egress_pkt.group2.data1.data1, egr_obj_bus.vmask_pattern);
      replace_header_field(egress_pkt.group2.data2.data2, egr_obj_bus.vmask_pattern);
    } else if ((obj_bus.vmask_offset == 0x8) && (obj_bus.vmask_length == 0x8)) {
      replace_header_field(egress_pkt.group2.data2.data2, egr_obj_bus.vmask_pattern);
    } else if ((obj_bus.vmask_offset == 0x8) && (obj_bus.vmask_length == 0x16)) {
      replace_header_field(egress_pkt.group2.data2.data2, egr_obj_bus.vmask_pattern);
      replace_header_field(egress_pkt.group2.data3.data3, egr_obj_bus.vmask_pattern);
    }
}
```

```
program vmask {


     //-----Inress---------//
    parse_begin(start);
    port_table.lookup(0);
    vlan_table.lookup(0);

    //-----Egress---------//
    assign_ing_to_egr_bus();
     do_packet_modif();

   }
```

Figure 10 - Example Data Masking NPL Code

Conditional if/else code constructs are used to determine the actions based on the table lookup results.

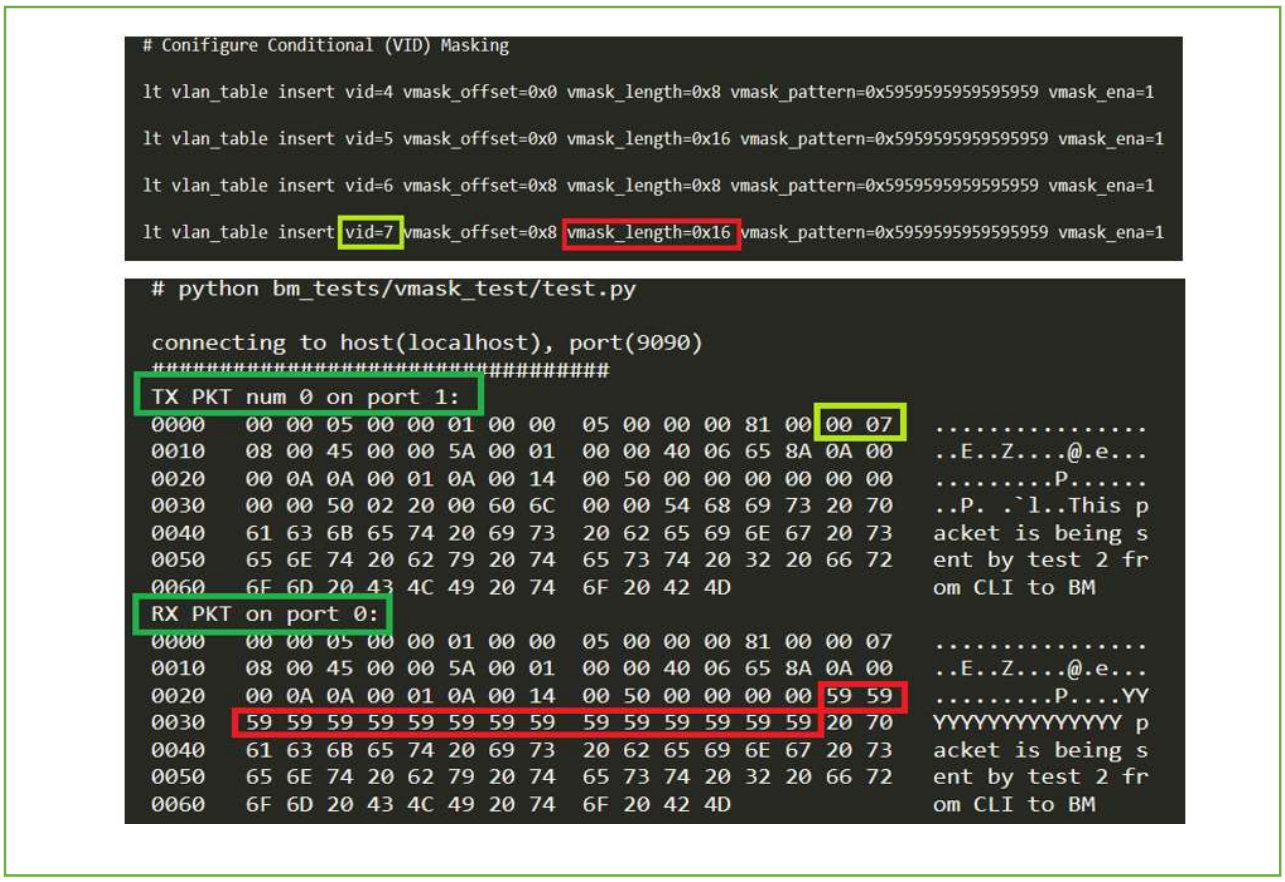


Figure11 - Example Data Masking Results

The NPL code processes an example packet. When the VID = 7 in the ingress packet, the egress packet has the defined data mask filed overwritten with 0x59, as seen in the above example.

5 Summary

This paper has provided several examples of advanced packet broker features implemented in NPL. The NPL code has been tested on the NPL emulator easily downloaded from NPL.org. In the future we intend to demonstrate these same advanced features, written in NPL, running on an OCP white box switch based on merchant switch silicon supporting NPL. The ability to provide custom features though NPL programming on OCP switches, loading custom software via ONIE, moves packet brokers closer to the ultimate goal of providing features, previously only available through custom FPGA implementations, on off the shelf white box switches.

6 References

https://nplang.org/npl/blog

https://opennetworking.org/p4/

Network Programming Language Specification v1.3

https://nplang.org/npl/specifications/

VLANs: IEEE 802.1Q-2014

https://www.ieee802.org/1/pages/802.1Q-2014.html

Multiprotocol Label Switching Architecture (MPLS) RFC3031
https://datatracker.ietf.org/doc/html/rfc3031

Generic Routing Encapsulation (GRE) RFC2784
https://datatracker.ietf.org/doc/html/rfc2784

3GPP Evolved Packet System; General Packet Radio Service Tunneling Protocol (GTP) TS 129 274-V15.4.0
https://www.etsi.org/deliver/etsi_ts/129200_129299/129274/15.04.00_60/ts_129274v150400p.pdf

7 License

OCP encourages participants to share their proposals, specifications, and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, “Materials”) are made available under the Creative Commons Attribution-ShareAlike 4.0 International License found here: https://creativecommons.org/licenses/by-sa/4.0/, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP’s request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at http://www.opencompute.org/participate/legal-documents/. If you have any questions please contact OCP.

