WHITE PAPER

Segment Routing P4 PoC

Building a PoC of Segment Routing at 100G Using FPGA Smart NIC and P4 Language

NETCOPE
INNOVATIVE ACCELERATION

Authors

Petr Kaštovský
CEO, Netcope Technologies

Chuck Tato
Director,
Communications Business Division
Intel® Corporation

Introduction

Service Function Chaining (SFC) is a process of passing network traffic among individual typically virtualized network functions in network function virtualization (NFV) and software-defined network (SDN) infrastructures. The typical SFC can be a series of IDS/IPS, firewalls, wide area network (WAN) optimizers, and load balancers that the traffic needs to go through on its way from client to server and vice versa [1],[2].

One approach to support SFC in NFV infrastructure is to use segment routing, in particular, the IPv6-based segment routing (SRv6) [3]. This technology is now becoming very attractive and deployed in large networks of the future as demonstrated by SoftBank's recent joint announcement with Cisco [4] as well as other deployments also announced by Cisco [5].

During the P4.org workshop in May 2017, teams from Bell Canada, Cisco Systems, and Barefoot Networks presented the concept of The Extensible Network - Evolution in Protocol and Data Plane Agility and explained the benefits of SRv6 for SFC [6].

In our proof of concept, we focused on demonstrating the ability to quickly develop SRv6 acceleration using an Intel® FPGA-based hardware accelerator and P4 programming language [7]. Similarly to accelerating SRv6, other applications can be accelerated using this approach. Good candidates being processing nodes of Vector Packet Processing (VPP) [8].

The theory of segment routing is explained in Figure 1.

Figure 1. IPv6 and Segment Routing Header. Source [6].

To perform segment routing, SRv6 router goes through the Segment List in Segment Routing Header (SRH) and uses field Segments Left (field SL=1 in Figure 1) as an index of the active segment that is copied over to Destination Address of IPv6 header. We decided to accelerate this data plane operations to demonstrate the productivity and flexibility of P4 language combined with FPGAs.

Table of Contents

P4 Language – Increased Productivity and Abstraction

P4 is a protocol and target independent language that allows for field upgradability of network devices. Furthermore, it significantly improves productivity compared to standard hardware description languages (HDL) that are used to write code for FPGAs. Unlike HDL languages, P4 is domain specific and focused on networking and that makes it an optimal platform for the acceleration of virtual network functions.

Implementation of the SRv6 PoC is quite straightforward. Example 1 shows the P4 code describing packet headers and protocol parser.

headers.p4

```
header_type ethernet_t {
   fields {
      dstAddr    : 48;
      srcAddr    : 48;
      etherType  : 16;
   }
}

header_type ipv6_t {
    fields {
       ver       : 4;
       trafClass : 8;
       flowLab   : 20;
       payLen    : 16;
       nextHead  : 8;
       hopLim    : 8;
       srcAddr   : 128;
       dstAddr   : 128;
    }
 }

//IPv6, extension header and
segments
header_type ipv6_ext_t {
   fields {
      nextHead  : 8;
      pad0      : 16;
      next_seg  : 8;
      pad1      : 32;
   }
   length: next_seg * 16;
}

header_type ipv6_seg_t {
   fields {
      val : 128;
   }
}

// Metadata
header_type seg_meta_t {
   fields {
      segVal : 128;
      nextSeg : 8;
   }
}
```

parser.p4

```
// General constants
#define IPV6_EXT_DEPTH    1

// Protocol numbers
#define PROTOCOL_IPV6
0x86dd
#define PROTOCOL_V6EXT
0x2B

// Instances of headers
// Outer header stack
metadata seg_meta_t    lastSeg;
header ethernet_t      ethernet_0;
header ipv6_t
ipv6;
header ipv6_ext_t      ipv6_ext;
header ipv6_seg_t      ipv6_seg;

// Parse graph
// Start
parser start {
   return parse_ethernet;
}

// Parse graph - Outer layers
// Outer ethernet_0
parser parse_ethernet {
   extract(ethernet_0);
   return select(latest.etherType) {
      PROTOCOL_IPV6          :
parse_ipv6;
      default
: ingress;
   }
}

parser parse_ipv6 {
   extract(ipv6);
   return select(latest.nextHead)
{
      PROTOCOL_V6EXT : parse_
ext;
      default       : ingress;
   }
}

parser parse_ext {
   extract(ipv6_ext);
   return parse_seg;
}

parser parse_seg {
   extract(ipv6_seg);
   set_metadata(lastSeg.
segVal,latest.val);
   return ingress;
}
```

Example 1. Headers and Parser Description in P4

Once the packets are parsed and appropriate packet header fields are extracted, it is important to define Match + Action tables that will perform the rewrite of destination IPv6 address by a Segment List item indexed by the Segments Left field and decrement of the Segments Left field.

tables.p4

```
// Actions
action rewrite() {
   modify_field(ipv6.
dstAddr,lastSeg.segVal);
   add_to_field(ipv6_ext.
next_seg, -1);
}

// Tables
// Table that does nothing
table tab_rewrite {
   actions {
      rewrite;
   }
}
```

main.p4

```
#include "headers.p4"
#include "parser.p4"
#include "tables.p4"

control ingress {
   if(valid(ipv6_ext)) {
      apply(tab_rewrite);
   }
}
```

Example 2. SRv6 Processing Code in P4

FPGA-Based Smart NIC as a Target

P4, being a target independent language, makes it easy for the designer to select Intel FPGAs as the target architecture. Intel FPGAs are a preferred hardware platform option for its efficiency when compared to CPUs and NPUs .

Using a P4 language as a programming language waives that barriers that are traditionally associated with programming FPGAs and brings the benefits of FPGAs into the networking domain.

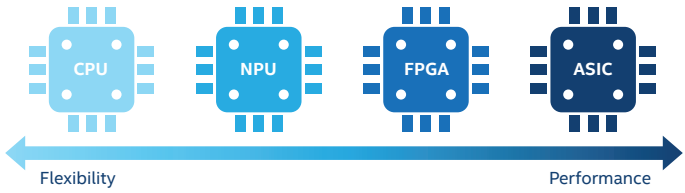

Figure 2. Netcope P4 Improves FPGA Flexibility

Netcope projects:

- Netcope P4 - Netcope P4 is an FPGA-vendor independent project by Netcope Technologies providing integration into different flavors of FPGA-based smart network interface cards (NICs) offering up to 2X 100GbE network capacity to fully deliver on the improved efficiency over NPUs.†

https://www.netcope.com/en/products/netcopep4

Executing the PoC

To perform the PoC we have used the online Netcope P4 Cloud to perform the synthesis of P4 code. The whole compilation process takes several steps but all of them are simplified for the user to only upload P4 code at the beginning and get FPGA bitstream at the end.

Step 1:

After uploading the code, an email confirmation will be sent confirming the task being queued for compilation and synthesis. The time required for compilation and synthesis is around two and a half hours for this PoC. Another email will be sent confirming that the task has finished. Also, it is possible to check the status of the task on the portal shown in Figure 3.

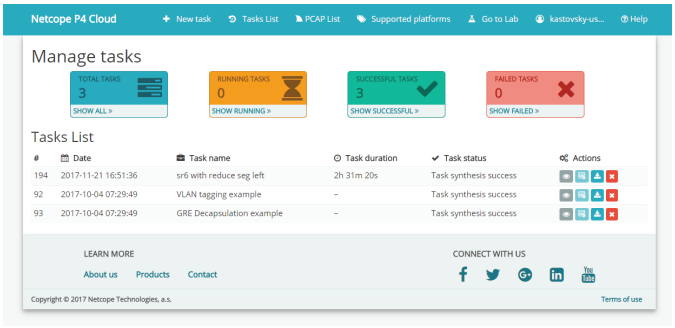

Figure 3. Netcope P4 Cloud GUI

Step 2:

Test the generated FPGA bitstream in the real hardware with the Ethernet cable connected in loopback. All that is needed is to load the bitstream into the FPGA, configure P4 Match&Action tables, start packet capture, and send test traffic from packet capture (PCAP) files. Figure 4 is an example of the specific packet before and after PoC SRv6 pipeline, which was dissected by Wireshark.

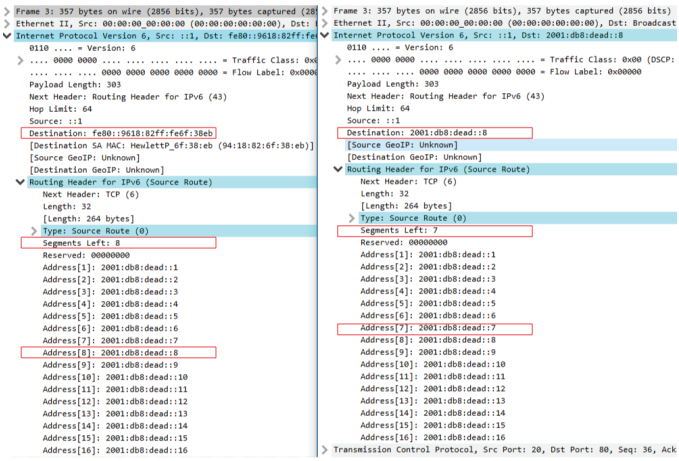

Figure 4. Packet Content Before and After SRv6 Processing

The code is functionally correct. Using the Intel Arria® 10 FPGA on the pre-production version of the Intel FPGA Programmable Acceleration Card N3000 with 8x10GE interfaces, we were able to achieve from 78 to 80 Gbps of port to port throughput without any specific performance optimizations or tunings. See Figure 5.

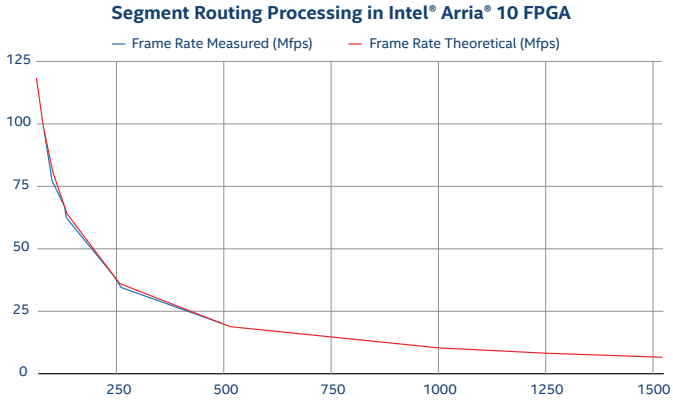

Figure 5. Throughput Achieved with the Real Hardware

It was a very interesting exercise where most of the time to get the PoC working was figuring out the problem on the algorithmic side. In other words, what data transformation should be done. Once this was clear, writing the code was a matter of day and getting the bitstream including test in hardware was a matter of 3 hours. This is a true revolution for Intel FPGA-based NIC programming.

References

[1] Ahmed AbdelSalam, Francois Clad, Clarence Filsfils, Stefano Salsano, Giuseppe Siracusano, Luca Veltri, "Implementation of Virtual Network Function Chaining through Segment Routing in a Linux-based NFV Infrastructure", Extended version of the conference paper [1] - v04 - April 2017. [Online].
Available: https://arxiv.org/pdf/1702.05157.pdf

[2] Open Networking Foundation, "L4-L7 Service Function Chaining Solution Architecture", Version 1.0, 14 June 2015, ONF TS-027. [Online].
Available: https://www.opennetworking.org/wp-content/uploads/2014/10/L4-L7_Service_Function_Chaining_Solution_Architecture.pdf

[3] D. Lebrun, S. Previdi, C. Filsfils, and O. Bonaventure, "Design and Implementation of IPv6 Segment Routing," Tech. Rep., 2016. [Online].
Available: http://dial.uclouvain.be/pr/boreal/object/boreal:174810

[4] Sara Cicero, Carter Cromwell, Emily Hunt, "SoftBank Teams with Cisco to Optimize Network Operations in its Next-Generation Mobile IP Core Network". newsroom.cisco.com website, 2017. [Online].
Available: https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1871147

[5] Jonathan Davidson, "Simplifying Networks through Segment Routing", blogs.cisco.com website, 2017. [Online].
Available: https://blogs.cisco.com/news/simplifying-networks-through-segment-routing

[6] Daniel Bernier, Milad Sharif, Clarence Filsfils, "The Extensible Network - Evolution in Protocol and Data Plane Agility", P4 Workshop 2017. [Online].
Available: http://www.segment-routing.net/images/20170517-bell-barefoot-cisco-P4%20Workshop%202017%20v2.pdf

[7] P4.org website, 2017. [Online].
Available: https://p4.org/

[8] FD.io Developer Wiki, "VPP, What is VPP?", last modified May 2017. [Online].
Available: https://wiki.fd.io/view/VPP/What_is_VPP%3F

† Tests measure performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.