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

Service Function Chaining (SFC) is a process of passing network traffic among individual typically virtualized network functions in NFV and SDN infrastructures. The typical SFC can be a series of IDS/IPS, firewalls, 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 network function virtualization infrastructure (NFVI) 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].

This raised our attention first during P4.org workshop in May 2017 where 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 PoC, we focused on demonstrating the ability to quickly develop SRv6 acceleration using an 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 can be best explained using a picture.

Figure 1: IPv6 and Segment routing header. Source [6].

In order 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 the Fig 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.

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 a perfect fit for the acceleration of virtual network functions.

Implementation of the SRv6 PoC is quite straightforward. Below is a 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;
}
```

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);
```

<table>
<tr>
<td>

```
header_type ipv6_seg_t {
    fields {
        val : 128;
    }
}

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

</td>
<td>

```
    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;
}
```

</td>
</tr>
</table>

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 Segments Left field and decrement the Segments Left field.

<table>
<tr>
<th>tables.p4</th>
<th>main.p4</th>
</tr>
<tr>
<td>

```
// 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;
    }
}
```

</td>
<td>

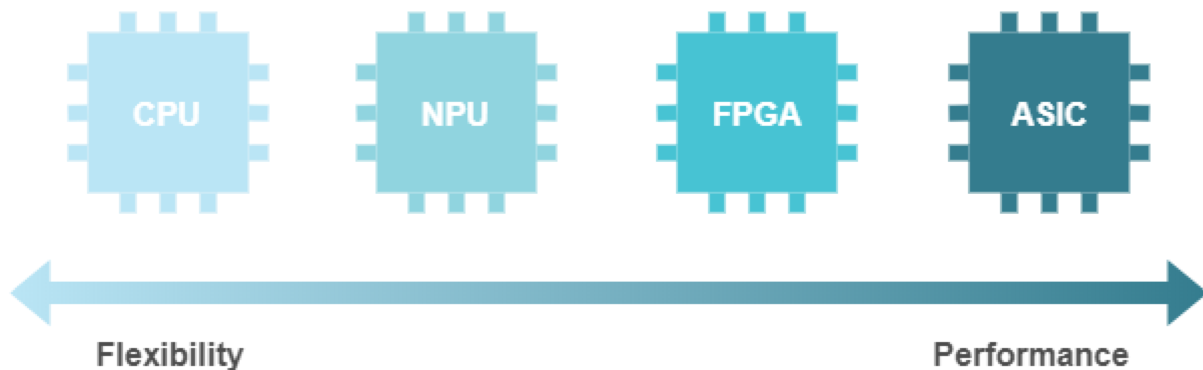
```
#include "headers.p4"
#include "parser.p4"
#include "tables.p4"

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

</td>
</tr>
</table>

FPGA-based smart NIC as a target

P4, being a target independent language, makes it easy for the designer to select target architecture. FPGA is a very suitable option for its efficiency as compared to CPUs and NPUs but was always behind in ease of use.

Using a P4 language as a programming language waives that barrier and brings FPGAs on par with NPUs in the networking domain when it comes to ease of use.



There are two projects:

- P4 - NetFPGA backed by NetFPGA community and using Xilinx SDNet to first convert P4 to internal language called Px and then to RTL. While the P4 to Px to RTL flow produces an IP core, NetFPGA framework provides the platform integration with the latest NetFPGA-SUME offering 4x10G network capacity.

https://github.com/NetFPGA/P4-NetFPGA-public/wiki

http://store.digilentinc.com/netfpga-sume-virtex-7-fpga-development-board/

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

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

https://www.netcope.com/en/products/fpga-boards

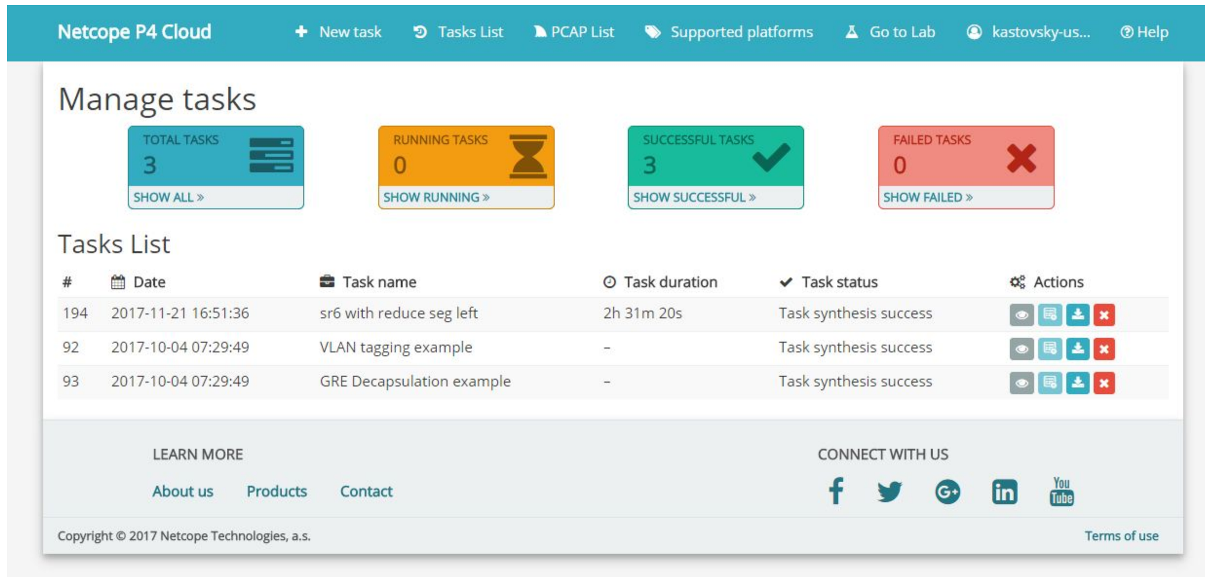
Executing the PoC

To perform the PoC we've used the online Netcope P4 Cloud in order 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.

Please watch the following videos For more information on the Netcope P4 Cloud:

https://www.youtube.com/watch?v=3EjzUL_BCzk&list=PL5FcCAxIMD72STKYYKJGtcStDfMYyP8oa

After uploading the code, we've received email confirmation about the task being queued for compilation and synthesis. In around two and a half hours we've received another email announcing that the task has finished. Opening up the portal to check:

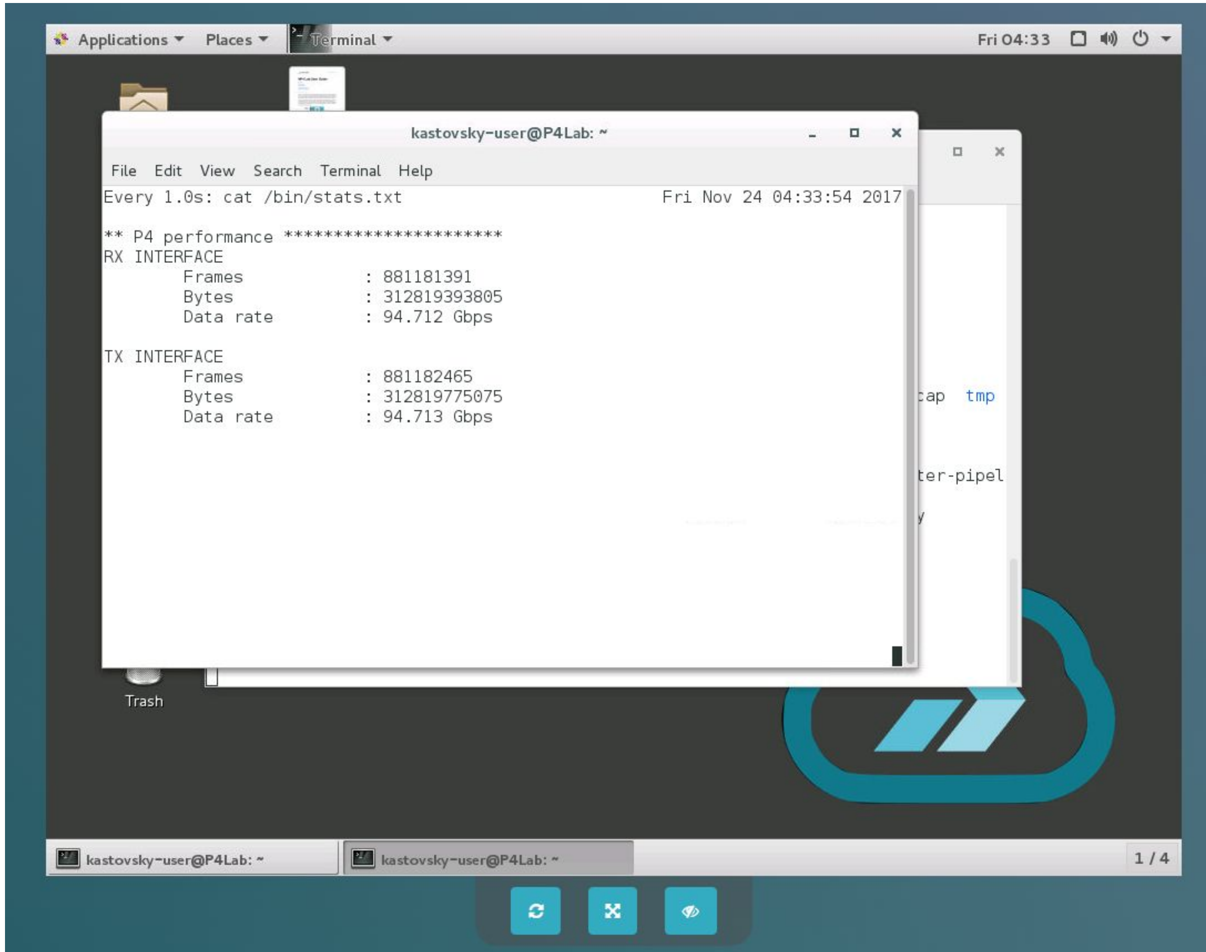


Next step was to go to Netcope P4 lab to verify that the produced bitstream works as expected. We've uploaded test PCAPs to verify that. After logging to Netcope P4 lab, we've loaded the created bitstream into the card, configured match and action tables, started capturing traffic and replayed prepared traffic samples. Here is the specific packet before and after our PoC SRv6 pipeline dissected by Wireshark.

```
> Frame 3: 357 bytes on wire (2856 bits), 357 bytes captured (28
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst:
v Internet Protocol Version 6, Src: ::1, Dst: fe80::9618:82ff:fe
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x0
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x0000
    Payload Length: 303
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 64
    Source: ::1
    Destination: fe80::9618:82ff:fe6f:38eb
    [Destination SA MAC: HewlettP_6f:38:eb (94:18:82:6f:38:eb)]
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  v Routing Header for IPv6 (Source Route)
      Next Header: TCP (6)
      Length: 32
      [Length: 264 bytes]
    > Type: Source Route (0)
      Segments Left: 8
      Reserved: 00000000
      Address[1]: 2001:db8:dead::1
      Address[2]: 2001:db8:dead::2
      Address[3]: 2001:db8:dead::3
      Address[4]: 2001:db8:dead::4
      Address[5]: 2001:db8:dead::5
      Address[6]: 2001:db8:dead::6
      Address[7]: 2001:db8:dead::7
      Address[8]: 2001:db8:dead::8
      Address[9]: 2001:db8:dead::9
      Address[10]: 2001:db8:dead::10
      Address[11]: 2001:db8:dead::11
      Address[12]: 2001:db8:dead::12
      Address[13]: 2001:db8:dead::13
      Address[14]: 2001:db8:dead::14
      Address[15]: 2001:db8:dead::15
      Address[16]: 2001:db8:dead::16
```

```
> Frame 3: 357 bytes on wire (2856 bits), 357 bytes captured (2856 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: Broadcast
v Internet Protocol Version 6, Src: ::1, Dst: 2001:db8:dead::8
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP:
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 303
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 64
    Source: ::1
    Destination: 2001:db8:dead::8
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  v Routing Header for IPv6 (Source Route)
      Next Header: TCP (6)
      Length: 32
      [Length: 264 bytes]
    > Type: Source Route (0)
      Segments Left: 7
      Reserved: 00000000
      Address[1]: 2001:db8:dead::1
      Address[2]: 2001:db8:dead::2
      Address[3]: 2001:db8:dead::3
      Address[4]: 2001:db8:dead::4
      Address[5]: 2001:db8:dead::5
      Address[6]: 2001:db8:dead::6
      Address[7]: 2001:db8:dead::7
      Address[8]: 2001:db8:dead::8
      Address[9]: 2001:db8:dead::9
      Address[10]: 2001:db8:dead::10
      Address[11]: 2001:db8:dead::11
      Address[12]: 2001:db8:dead::12
      Address[13]: 2001:db8:dead::13
      Address[14]: 2001:db8:dead::14
      Address[15]: 2001:db8:dead::15
      Address[16]: 2001:db8:dead::16
> Transmission Control Protocol, Src Port: 20, Dst Port: 80, Seq: 36, Ack
```

Functionality is correct. How about the performance? Using the hardware provided in demo lab that is mostly used for presentation purposes we were able to achieve 94Gbps of throughput without any specific performance optimizations or tunings. Please, see the following figure.



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 matter of 3 hours. At almost 100Gbps. This is a true revolution in FPGA-based smart NIC programming.

Resources

[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