Service Function Chaining via SR-MPLS over P4: A rudimentary analysis

Martins Mihaeljans
Institute of Photonics, Electronics and Telecommunications
Riga Technical University
Riga, Latvia
martins.mihaeljans@edu.rtu.lv

Andris Skrastins
Institute of Photonics, Electronics and Telecommunications
Riga Technical University
Riga, Latvia
andris.skrastins@rtu.lv

Jurgis Porins
Institute of Photonics, Electronics and Telecommunications
Riga Technical University
Riga, Latvia
jurgis.porins@rtu.lv

***Abstract*—This study is a rudimentary analysis of segment routing with MPLS (SR-MPLS) use case for service function chaining (SFC) on protocol-independent switch architecture (PISA). Segment routing is a packet forwarding method that leverages both source routing and packet encapsulation. SFC is packet steering through ordered service function (SF) path technique where SFs are firewall, deep packet inspection, network address translation, etc.**

Goal of this study is to investigate the service function chaining domain use case scenario for PISA data plane incorporating SR-MPLS as transport. We developed program code for SFC domain elements in a programmable protocol-independent packet processing (P4) and python languages.

The outcome of this study is functionating SFC domain emulation in Mininet environment. Emulation results shows that SR-MPLS makes minimal overhead for all emulated SF paths in generalized proportion of approximately 1/1000 from overall transmission rate. Results also reveal a minimalistic transmission rate dependency on path length in generalized proportion of approximately 1/40 of transmission rate drop between shorter and longer SF paths. It's evident that SR-MPLS is fit to be used for SFC on PISA switches.

Keywords—Intent-based networking, Programmable protocol -independent packet processing, Software defined networking, Service function chaining, Segment Routing with MPLS

I. INTRODUCTION

The Internet – a perpetual interconnectivity service held under man's providence is a puzzle of many pieces. One of which namely Software Defined Networking (SDN) has approached a significant shift in its underlay hardware structure. Over past years a protocol-independent switch architecture (PISA) has made its way to both academical research and commercial products. PISA enables network equipment re-programmability. Turning SDN functionality development from proprietary vendor provision to an open-source device administrator driven task [1].

Service function chaining (SFC) is data traffic steering concept with objective on differentiating packet processing without modification of underlay network topology [2]. SFC gained traction due recent advancement of segment routing.

Segment routing with MPLS (SR-MPLS) is a routing method that uses source routing in combination with MPLS for encapsulation enabling reactive SFC initialization [3].

In this study we explore mentioned technologies in a Mininet emulation environment. We develop program code for SFC domain elements for PISA switches with use of programmable protocol-independent packet processing (P4) language and python program code for topology host nodes acting as service functions (SFs). Results show SR-MPLS as an adequate choice for SFC encapsulation as it generates minimal overhead. SFC length to transmission ratio shows insignificant dependency.

II. RELATED WORKS

Authors of [4] points out the difference between OpenFlow and P4 switches. OpenFlow was introduced to program control plane. Thus, switch match-action pipeline stayed protocol dependent. P4 however was introduced to program data plane and make match-action pipeline protocol-independent. Which also allows in our experiment to utilize custom SF encapsulation processing.

Study [5] utilizes P4 registers to store address resolution protocol (ARP) information for enablement of autonomous forwarding. An ability for self-organization in P4 pipelines can reduce information exchange with network controller. Our study also reveals a need for control channels between SFC enabled P4 switches.

Authors of [6] propose congestion aware multi-path label switching algorithm for detection of elephant flows in data center networks. Like our study, P4 ability of customized packet processing is used for model development in Mininet.

Research [7] states that stateful SFs are common in real-world deployments but rarely does an existing P4 literature examines them. We do cover the necessity for use of stateful SFs in section VII.

Authors [8] of study introduce a flexible system for NF offloading to P4 switches themselves. Which in SFC terminology would be equal to making an SF forwarding element do the service functions work. For example, P4 switch could work as a network address translator.

Like in our study, authors of [9] also find segment routing generated overhead for SFC excessive and propose a heuristic algorithm for SF encapsulation compression for SR-IPv6 routing. Our proposal of single segment identifier (SID) use is applicable either if proactive classification is available or path taken by the packet has been reactively backtracked as in their study.

III. PROGRAMMABLE PACKET PROCESSING

A. SDN packet processing capability provision shift

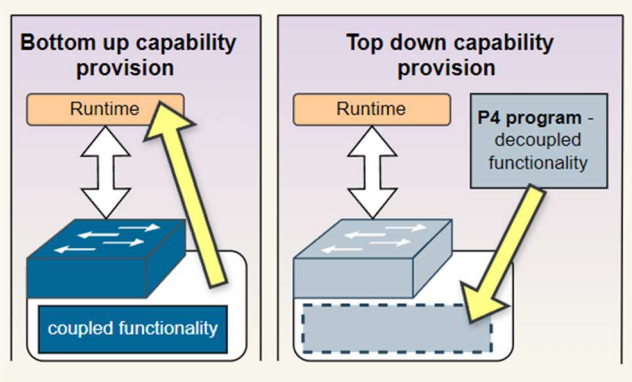

Fig. 1. Capability provision

979-8-3503-5469-0/24/$31.00 ©2024 IEEE

In legacy as well as in older white switches (e.g. Open Virtual Switches) their functionality is coupled to inbuild logics and is dependent on vendor upgrade cycles. This type of capability provision is bottom up as information about available configuration of the switch is gathered from the switch itself. It is shown in Fig. 1.

Programmable packet processing allows to transform switch's functionality according to network administrator's instructions by compiling P4 programs. [10] This type of capability provision is called top down as functionality is enforced from upper control layer. It is shown in Fig. 1.

1) PISA architecture building blocks

PISA architectures building blocks and its derivation into applicable switch model is show in Figure 2 where:

- Parser – Incoming packets evaluation through multiple state filter according to its content for further processing. States are mechanisms that determine whether to continue packet processing or not and with which action to proceed.
- Match and action pipeline – Parsed header content match to predefined custom to users' requirements logic for appropriate modification and processing. Memory holds users' predefined logic in table values while arithmetic logic unit (ALU) allows for packet header content modification actions.
- Deparser – An exit point where transmittable packet is constructed from processed packet headers and original or modified payload before its egress from the appropriate switch interface.

Portable switch architecture (PSA) [11] introduces traffic manager (TM) building block. TM is responsible for fixed functions like packet buffering, reordering, and queuing, etc. TM can differentiate between hardware and its working principle is hardware vendor defined. V1model is PISA derivation implemented in bmv2 software switch of Mininet emulator that we used in our experiment.

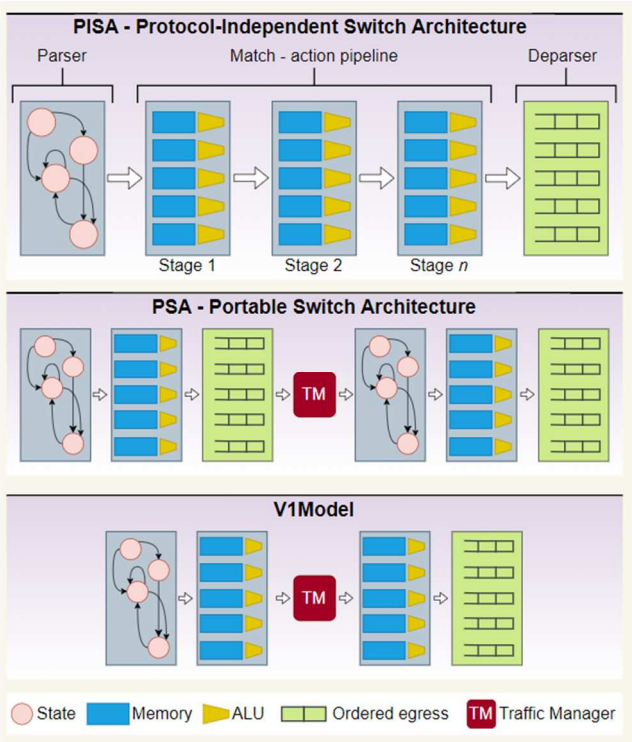

Fig. 2. PISA model derivation

IV. SEGMENT ROUTING WITH MPLS

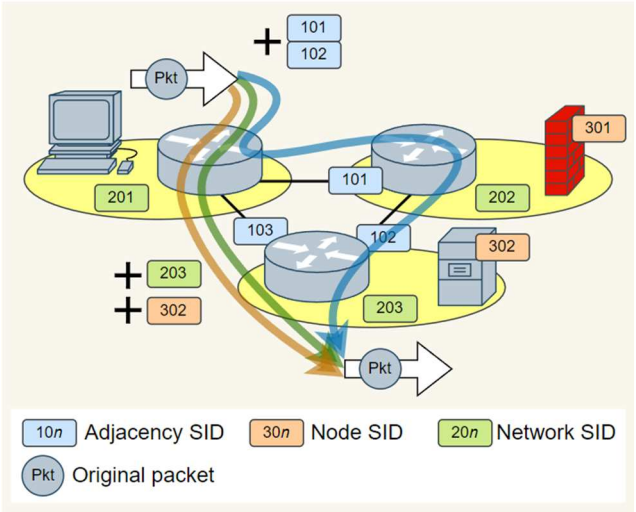

Fig. 3. SR-MPLS example

Segment routing with MPLS (SR-MPLS) utilizes labels as segment identifiers (SIDs). Shown in Fig. 3. is SR-MPLS with three possible routes. When original packet enters SR-MPLS enabled routing domain an encapsulation is pushed on top of packet. This encapsulation can consist of a single label or label stack. One label is equivalent to one SID. There are three types of SIDs:

- Adjacency SID – used if the packet needs to be routed along a very specific path as these SIDs represent directly connected neighbors.
- Node SID – used if the packet needs to be routed to a specific network node.
- Network SID – used if the packet needs to be routed to a specific network.

The encapsulation can be popped upon exiting the SR-MPLS enabled routing domain. A swap is a composite action made from push and pop action combination. Swap is performed inside a SR-MPLS domain for when packet path consists of multiple stacked labels or if label stacking is prohibited and swap is done on each packet hop.

In Fig. 3. a computer can reach server via use of three different SID types. A stack of adjacency SIDs represent blue path where SID 101 would be popped before packet reached the second router, while SID 102 would be popped before packet reached the third router. Although SIDs 203 and 302 represent different values their path is equivalent therefore they can be used interchangeably.

The difference between SR-MPLS and traditional MPLS is that SR-MPLS does not utilize an additional label distribution protocol (LDP). Instead, label information is exchanged among routers via routing protocol updates. Protocols in use are modified versions of Intermediate System-to-Intermediate System (IS-IS), Border Gateway Protocol (BGP) or Open Shortest Path First (OSPF) [3].

V. SERVICE FUNCTION CHAINING

Service function chaining (SFC) is designed to ease network configuration by allowing to steer traffic through ordered service function (SF) path without modification of underlay network topology. Originally it was introduced by Cisco conglomerate alongside a Network Service Header (NSH) protocol as applicable SF encapsulation. However, SFC quickly grew out of use in virtualized network functions and is heavily leveraged in 5G networks as well.

A. SFC placement in SDN conception

Network configuration policy setup with and without SFC thorough SDN structural planes is shown in Figure 4. With use of SFC user at a management plane is not required to specificly indicate which network elements should data traffic cross on its way to destination. It is enough to indicate what network functions needs to be in use. It is conroller's task to map requested SFs with underlay network topology.

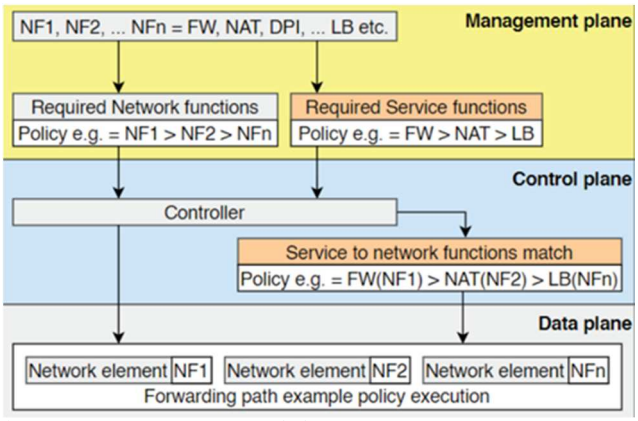

Fig. 4. SFC among SDN structural planes

We have examined SFC domain in our previous research [12] that can be in use if an in-depth explanatory of SFC elements is required.

B. SF path types

SF paths differentiate not only with SFs that they contain but also with how flows are handled. Three SF path types are shown in Figure 5. Service function forwarder (SFF) is SF domain element that is used only to forward network traffic along the path and does not modify it. SF Path types are:

1. SFC is applied for one of the data flow directions.
2. SFC application is asymmetrical for both directions.
3. SFC application is symmetrical for both directions.

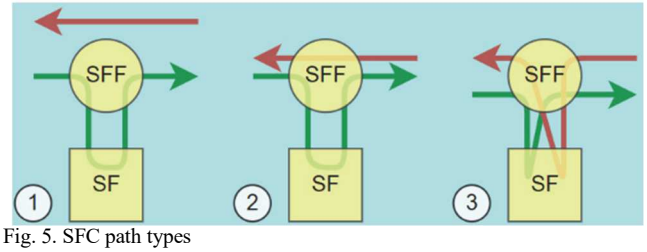

Fig. 5. SFC path types

We have previously studied paths symmetries effect on successful SF application [13]. Symmetrical SFC application might be an obligatory for SF chains with stateful SFs in them like non-transparent proxies, SIP servers, L7 firewalls.

C. Proactive SF path policy

For SFC to work an SF path policy is required. This policy holds the rule base of network traffic classification for appropriate SF encapsulation application. Proactive SF classification process is shown in Fig. 6.

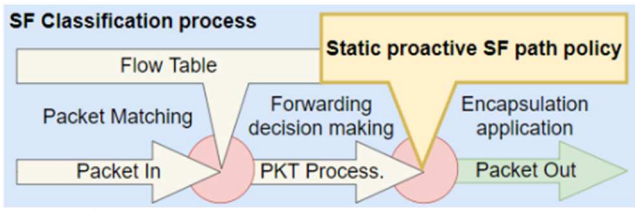

Fig. 6. SF classification process

Static proactive SF path policy is a rule base set in advance of any network traffic arrival by network administrator. Main disadvantage of path policy setup proactively is that only expected network traffic can be classified and enter SFC domain.

In our research we examined an alternative to proactive SF path policy which is reactive policy instantiation.[14] Reactive SF path discovery works on try and catch principle with utilization of SFC control plane channels for network traffic reclassification[15]. With use of reclassification a need for proactively set rule base would be eliminated.

VI. EMULATION SETUP

A. Network topology

To investigate working principle of PISA we developed an SFC domain shown in Fig. 7. Classifier pushes MPLS encapsulation on top of incoming network traffic coming from source. SF forwarders steers network traffic according to outermost SID in MPLS label stack. Service functions are transparent proxies. SF proxy removes bottom of stack label for packet to leave SFC domain and is sent to destination.

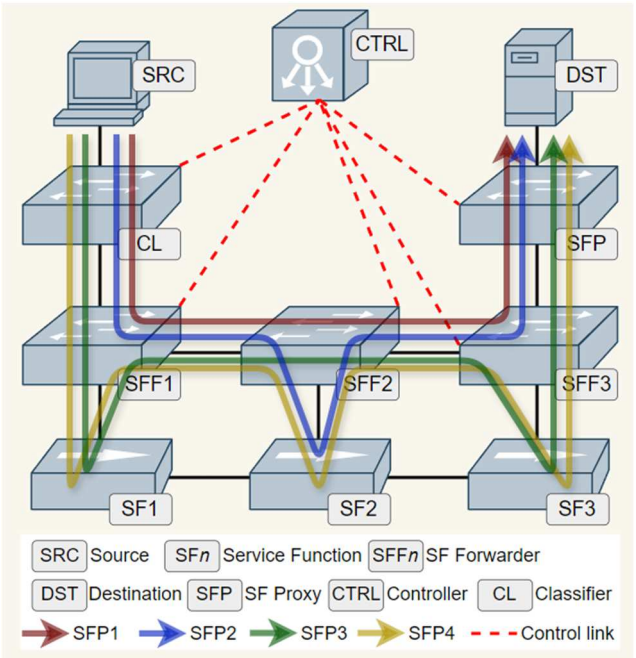

Fig. 7. Emulated SFC domain

There are four SF paths shown in Fig. 7. Each path differs from one another with their length. SFP1 is special. It does not cross any of the SFs, therefore, requires no encapsulation as its path matches one provided by underlay network topology. Controller is the network element in which the proactive SF path policy is stored for redistribution to all other network elements.

Experimental scenario was implemented in Mininet network emulator with use of P4-Utils[16]. P4-Utils provides not only emulation needed software but also environment setup script for dependency installation as well as literature and examples of various P4 enabled use cases.

The underlay network topology of our SFC domain is shown in Fig. 8. All switches are behavioral model version 2 (bmv2) software switches that work according to V1model architecture shown in Fig. 2. Switches use loopback interface as outbound link to the controller. Data path link bandwidth was set to 10Mbits/s.

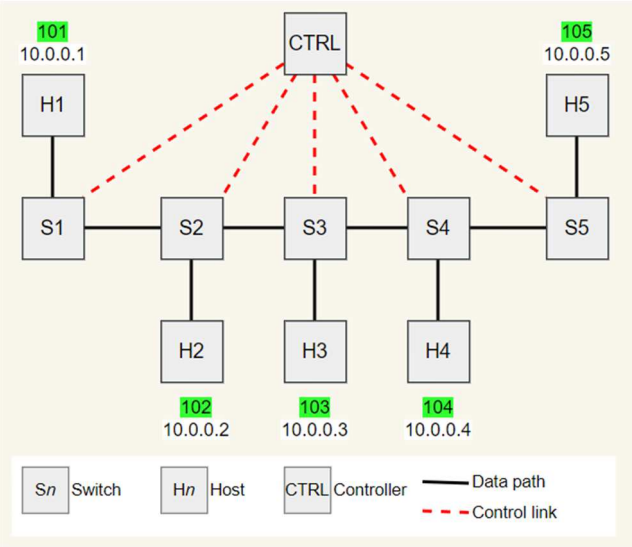

Fig. 8. Underlay network topology

Network element use was as follows:

- H1 – Source host (running Iperf as client).
- H2, H3, H4 – Service Functions (running proxy server python program from Fig. 10).
- S1, S2, S3, S4, S5 – SF forwarding elements (running P4 program from Fig. 9).
- H5 – Destination host (running Iperf as server).

B. *Emulation process*

The designing of emulation setup was a challenging task. It required dozens of reruns and misconfiguration fixes. The task included not only topology planning in Mininet, but we also developed a separate P4 program for SF forwarding elements, a python program for service functions and a python program for the controller.

When the network emulation environment was ready, we conducted 40 experiments (10 sequential runs for each SF path). The emulation process of each run was as follows:

1. Running network topology program – it starts the Mininet command line interface (CLI) along with bmv2 software switches with the P4 program and controller python program. It also starts packet capture on all switch interfaces.
2. Setting maximum transfer unit parameter for each host in topology from Mininet CLI. Lower for hosts 1 and 5, and higher for hosts 2, 3, and 4. (SFC underlay network topology shown in Fig. 8.).
3. Running service function program – through Xterm terminals on hosts 2, 3, and 4.
4. Starting Iperf server on host 5 – the TCP port to listen on set according to needed path (5565 port – SF path 1, 5566 port – SF path 2, 5567 port – SF path 3, and 5568 port – SF path 4).
5. Starting Iperf client – we used no additional arguments which makes Iperf to test the available throughput for 10 seconds with gradually increasing TCP window.
6. Repeating Iperf client start for 10 iterations.
7. Collecting network snapshot – making a copy of packet captures (each network interface gets a separate capture for ingress and egress traffic which makes in total of 26 captures in this topology). It's worth mentioning, that realistic real-world cases would have way more network interfaces which potentially could be divided in sub interfaces as virtual local area network (VLAN) technology widely used in data center networks.
8. Repeating previous steps for all SF paths.

We endured a shortcoming of P4-Utils upon development of network emulation setup. The application programming interface (API) does not hold any configuration capability of host node maximum transfer unit (MTU) parameter. We overcame this shortcoming by manually setting MTU values for each host in Mininet CLI after topology initialization and before running any Iperf. This was needed as Scapy packet crafting tool used in SFs did not allow frames bigger than MTU of the host. This problem arose due to Iperf utilizing maximum of allowed packet size and P4 switches added an MPLS label on top of that. Therefore, we needed to limit Iperf MTU and rise SFs MTU.

C. *P4 program for SF forwarding elements*

Flow diagram of SF forwarding elements (classifier, SF forwarders, and SF proxy) logic is shown in Fig. 9. The logic is implemented in program code written in programmable protocol-independent packet processing (P4) language [17].

As shown in Fig. 9. an incoming packet is examined whether it already has an MPLS or not by parsing its ethernet header for ethernet header type. If encapsulation is present (type 0x8847) packet is matched against SID database and forwarded further trough an appropriate egress port. If encapsulation is not present, then TCP header is parsed. If TCP destination port has a match in proactively setup SF classification policy, then an SFC encapsulation is enforced, and packet is forwarded further through an appropriate egress port. However, if no match in SF classification policy is found, then packet is forwarded according to L2 hardware address.

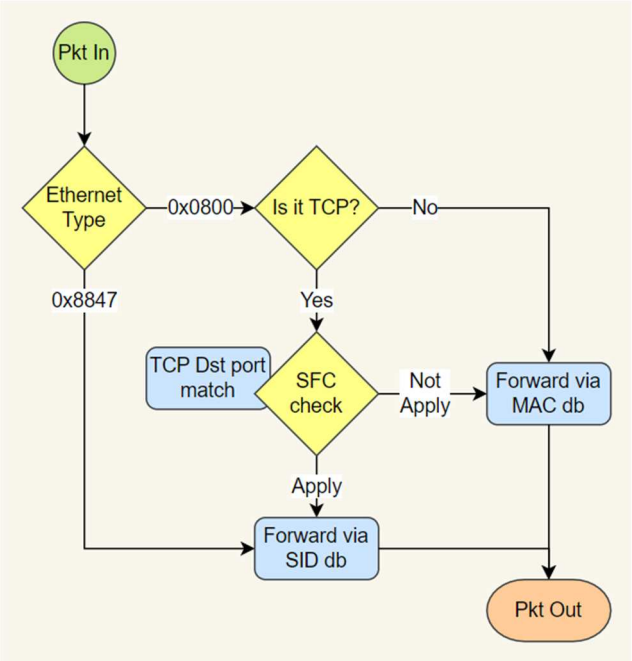

Fig. 9. Logic of SF forwarding elements

D. *Python program for service functions*

Flow diagram of service functions logic is show in Fig. 10. This logic is used in python program run by hosts which act as SFs. Main functionality of this program is provided by Scapy packet crafting tool [18].

On packet arrival it is filtered whether it is MPLS or not. Afterwards destination hardware address is examined to distinguish whether captured packet is incoming or outgoing. If it is incoming only then time to live (TTL) of MPLS and IP headers is modified, and outermost label match is done.

We encountered an issue with Scapy as a noticeable drop of transmission rate appeared for all paths that crossed SFs. After path examination of packet captures, we noticed that lots of packets delivered to first SF in path are being lost, which causes Iperf to limit its rate. Through online query we settled on potential cause being Scapy's sendp() function which opens and closes a socket for each packet.

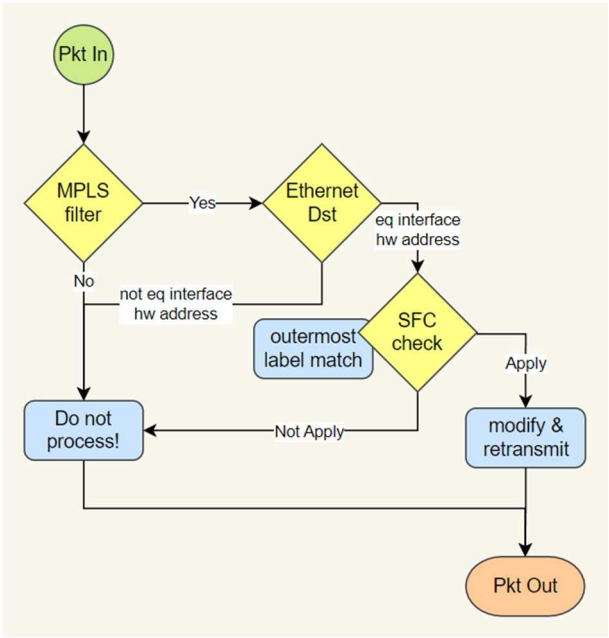

Fig. 10. Logic of service functions

VII. Results and Analysis

A. Emulation results

P4-Utils has an in-built functionality of making a packet capture of each P4 switch interface. In same time P4-Utils also allows active logging for all switches. The combination of both abilities can be referred to as network snapshot creation. Network snapshots relaxes network troubleshooting task as problem solving no longer requires a separate hop at the time approach and enables a cross-examination among all links. This allowed us to understand the traffic rate problem mentioned in the end of previous section.

As shown in Fig. 8. MPLS encapsulation was globally significant and resembled host IP addressing. For example, node SID of 103 instructs switch to forward packet in hosts with IP 10.0.0.3 direction, SID 104 would be packet forwarding to host with IP 10.0.0.4 and so on.

Service function path 1 (red arrow in Fig. 7.) is packet switched by hardware address therefore it gains no SF encapsulation as it does not to cross any SFs. This path was created to test whether SF classification worked correctly.

No.	Time	Eth Src	Eth Dst	Label	TTL	Length
6	23:36:47.644	00:00:0a:00:00:01	00:00:0a:00:00:05		64	74
7	23:36:47.646	00:00:0a:00:00:01	00:00:0a:00:00:03	103,105	63	82
8	23:36:47.648	00:00:0a:00:00:01	00:00:0a:00:00:03	103,105	62	82
9	23:36:47.650	00:00:0a:00:00:01	00:00:0a:00:00:03	105	61	78
10	23:36:47.674	00:00:0a:00:00:01	00:00:0a:00:00:05	105	60	78
11	23:36:47.675	00:00:0a:00:00:01	00:00:0a:00:00:05	105	59	78
12	23:36:47.677	00:00:0a:00:00:01	00:00:0a:00:00:05	105	58	78
13	23:36:47.679	00:00:0a:00:00:01	00:00:0a:00:00:05		57	74
14	23:36:47.679	00:00:0a:00:00:05	00:00:0a:00:00:01		64	74

```
> Frame 7: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
> Ethernet II, Src: 00:00:0a:00:00:01, Dst: 00:00:0a:00:00:03
> MultiProtocol Label Switching Header, Label: 103, Exp: 0, S: 0, TTL: 63
> MultiProtocol Label Switching Header, Label: 105, Exp: 0, S: 1, TTL: 63
> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.5
> Transmission Control Protocol, Src Port: 46782, Dst Port: 5566, Seq: 0, L
```

Fig. 11. SF path 2

Service function path 2 (blue arrow in Fig. 7.) is shown in Fig. 11. From merging packet captures taken in multiple places throughout the topology we were able to track SF encapsulation changes along the taken path. Circled with red is the SIDs in use. Packet nr. 6. and 13. shows that original packet enters and exits SFC domain with no encapsulation. From the TTL we can see that the path is 8 hops long. It's also evident that L2 hardware addressing gets modified but L3 logical does not.

No.	Time	Eth Src	Eth Dst	Label	TTL	Leng
19	23:48:08.314	00:00:0a:00:00:01	00:00:0a:00:00:05		64	74
20	23:48:08.315	00:00:0a:00:00:01	00:00:0a:00:00:02	102,104,105	63	86
21	23:48:08.316	00:00:0a:00:00:01	00:00:0a:00:00:02	104,105	62	82
22	23:48:08.334	00:00:0a:00:00:01	00:00:0a:00:00:04	104,105	61	82
23	23:48:08.335	00:00:0a:00:00:01	00:00:0a:00:00:04	104,105	60	82
24	23:48:08.336	00:00:0a:00:00:01	00:00:0a:00:00:04	104,105	59	82
25	23:48:08.337	00:00:0a:00:00:01	00:00:0a:00:00:04	105	58	78
26	23:48:08.359	00:00:0a:00:00:01	00:00:0a:00:00:05	105	57	78
27	23:48:08.361	00:00:0a:00:00:01	00:00:0a:00:00:05	105	56	78
28	23:48:08.363	00:00:0a:00:00:01	00:00:0a:00:00:05		55	74
29	23:48:08.363	00:00:0a:00:00:05	00:00:0a:00:00:01		64	74

```
> Frame 20: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)
> Ethernet II, Src: 00:00:0a:00:00:01, Dst: 00:00:0a:00:00:02
> MultiProtocol Label Switching Header, Label: 102, Exp: 0, S: 0, TTL: 63
> MultiProtocol Label Switching Header, Label: 104, Exp: 0, S: 0, TTL: 63
> MultiProtocol Label Switching Header, Label: 105, Exp: 0, S: 1, TTL: 63
> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.5
> Transmission Control Protocol, Src Port: 45498, Dst Port: 5567, Seq: 0, L
```

Fig. 12. SF path 3

Service function path 3 (green arrow in Fig. 7.) is shown in Fig. 12. This path is 10 hops long. It's also more evident that the L2 addressing changes occur according to next host node in SF path on its egress. The same logic is used for MPLS label stack, but the pop happens before the SF ingress. For example, at nr. 21. packet enters SF1 with L2 destination address 00:00:0a:00:00:02 that is equal to SF1 interfaces hardware address but label with SID 102 has been popped by the previous SFC element SFF1. The remaining SF encapsulation with the outermost label containing SID 104 tells SF1 and next SFC element along the path that packet needs to be routed to SF3.

.	Time	Eth Src	Eth Dst	Label	TTL	Ler
13	23:57:04.154	00:00:0a:00:00:01	00:00:0a:00:00:05		64	74
14	23:57:04.156	00:00:0a:00:00:01	00:00:0a:00:00:02	102,103,104,105	63	90
15	23:57:04.158	00:00:0a:00:00:01	00:00:0a:00:00:02	103,104,105	62	86
16	23:57:04.187	00:00:0a:00:00:01	00:00:0a:00:00:03	103,104,105	61	86
17	23:57:04.189	00:00:0a:00:00:01	00:00:0a:00:00:03	103,104,105	60	86
18	23:57:04.190	00:00:0a:00:00:01	00:00:0a:00:00:03	104,105	59	82
19	23:57:04.219	00:00:0a:00:00:01	00:00:0a:00:00:04	104,105	58	82
20	23:57:04.220	00:00:0a:00:00:01	00:00:0a:00:00:04	104,105	57	82
21	23:57:04.221	00:00:0a:00:00:01	00:00:0a:00:00:04	105	56	78
22	23:57:04.239	00:00:0a:00:00:01	00:00:0a:00:00:05	105	55	78
23	23:57:04.241	00:00:0a:00:00:01	00:00:0a:00:00:05	105	54	78
24	23:57:04.243	00:00:0a:00:00:01	00:00:0a:00:00:05		53	74
25	23:57:04.244	00:00:0a:00:00:05	00:00:0a:00:00:01		64	74

```
Frame 14: 90 bytes on wire (720 bits), 90 bytes captured (720 bits)
Ethernet II, Src: 00:00:0a:00:00:01, Dst: 00:00:0a:00:00:02
MultiProtocol Label Switching Header, Label: 102, Exp: 0, S: 0, TTL: 63
MultiProtocol Label Switching Header, Label: 103, Exp: 0, S: 0, TTL: 63
MultiProtocol Label Switching Header, Label: 104, Exp: 0, S: 0, TTL: 63
MultiProtocol Label Switching Header, Label: 105, Exp: 0, S: 1, TTL: 63
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.5
Transmission Control Protocol, Src Port: 56356, Dst Port: 5568, Seq: 0, Len:
```

Fig. 13. SF path 4

Service function path 3 (yellow arrow in Fig. 7.) is shown in Fig. 13. This path is 12 hops long. Source routing is most evident in this route as the original packet gains a four SID long MPLS label stack at the encapsulation. The label stack slowly depletes as packet crosses all three SFs. Its worth to note that, the encapsulation stacks length in this experiment is directly related to the network topology that is being emulated.

TABLE I. TCP DOWNSTREAM CONNECTION PARAMETERS

Nr.	*Measurement*	*Path 1*	*Path 2*	*Path 3*	*Path 4*
1.	Avg. pkt rate, [pps]	138.2	37.8	38.2	39.7
2.	Avg. bitrate [Kb/s]	9924	1490	1438	1451

Table I shows data transmission rate. Iperf downstream flow is the one going from client to server as it contains the payload with data. Path 1 shows that Mininet bandwidth parameter set to 10Mbit/s was working accordingly and packet switching had insignificant effect on the transmission.

TABLE II. SERVICE FUNCTION INPUT / OUTPUT PACKET COUNT

Nr.	*SFC Path*	*SF 1*		*SF 2*		*SF 3*	
		Rx	*Tx*	*Rx*	*Tx*	*Rx*	*Tx*
1.	Path 2	-	-	3012	2641	-	-
2.	Path 3	3200	2828	-	-	2828	2826
3.	Path 4	3223	2895	2895	2895	2895	2892

Table II shows received (Rx) and transmitted (Tx) packet count difference for each SF and each path. It is evident that packet rate decrees occur at the first crossed SF in each path. For example, Path 2 crosses only service function 2 and the Rx packet count is much bigger than the Tx packet count.

The difference in I/O packet count likely is a cause of the significant data transmission rate difference between path 1 and rest of the paths shown in Table I. As mentioned earlier rate limiting occurs due to working principle of Scapy tool.

We have studied Intent-based networking (IBN) structure and supposed working principle [19]. In IBN network control and maintenance is achieved with human to artificial intelligence (AI) dialog. It is suggested that AI and machine learning (ML) can utilize network snapshots as network state monitoring mechanism for pre and post configuration change. In the current experiment network snapshot aided in troubleshooting of data transmission rate problem.

TABLE III. SF ENCAPSUALTION OVERHEAD

Nr.	**MPLS Overhead for TCP downstream connection [bits/s]**					
	SFC Path	*CL >*	*SF 1 >*	*SF 2 >*	*SF 3 >*	*Avg.*
1.	Path 2	1551	-	685	-	1118
2.	Path 3	2237	1327	-	663	1409
3.	Path 4	2928	1994	1328	663	1728

Table III shows SF encapsulation overhead in bits per second. At the beginning of each path the overhead is much larger than that at the end. This is the effect of source routing working principle of SR-MPLS. The classifier must attach all necessary SIDs to the original packet for it to be steered along the desired path. Before each SF an SF forwarder (SF) pops outermost label containing the adjacent SFs SID.

B. Conducted analysis

From the gained results shown in previous section we conducted an analysis with use of Octave. Values shown in Table I and Table III of results are mean values calculated from 10 iterations for each path with mean function.

Shown in Fig. 14, Fig. 15., and Fig. 16. measured value points at SFC path lengths 8, 10, and 12 corresponds to path 2, path 3, and path 4 respectively. SF path 1 was not included in the analyses as it did not utilize SR-MPLS routing.

Calculation of encapsulation overhead to path length dependency shown in Fig. 14 was done as follows:

1. Creating a vector of MPLS header overhead in bits per second for each path – data source is analyses of protocol hierarchy with Wireshark for each network snapshot.
2. Calculating of mean values from made vectors with mean function.
3. Calculating standard deviation values from vectors made in point 1. with std function.
4. Creating a vector for x axis of values of 6 to 14 with step 1 with use of linespace function.
5. Creating coefficients of linear regression according to formula (1) from mean points and path lengths respectively with use of polyfit function with curve value 1.
6. Creating a linear regression vector from mean values and x axis vectors with polyval function.
7. Plotting measured values with use of errorbar function and linear regression with use of plot function in single figure.

Similar methodology was repeated for graphs in Figures 15. and 16. Data source for those were network snapshot analyses of endpoint conversations through Wireshark.

Linear regression formula (1) is as follows:

$$a * x + b = y \quad (1)$$

In this study x represents SF path length in hops for all graphs, y represents measured values for each graph respectively, but coefficients a and b are regressions slope determining factor calculated with use of polyfit and polyval functions.

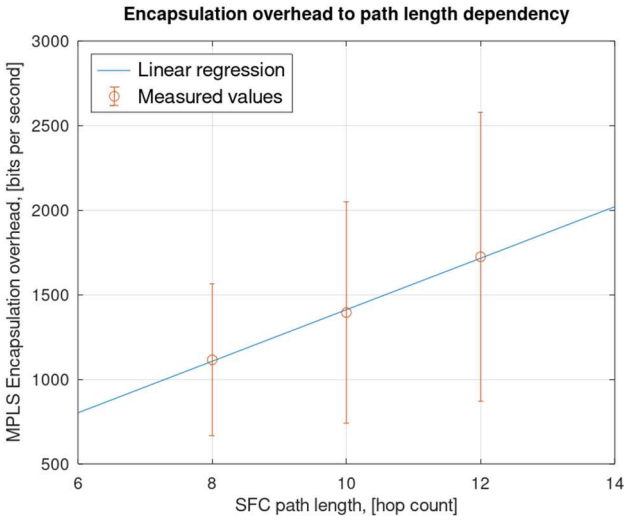

Fig. 14. Encapsulation overhead dependency

SF encapsulation (MPLS labels representing node SIDs) overhead to SF path length (hop count packet takes in SFC domain) dependency is shown in Fig. 14.

Calculated linear regression shows that encapsulation overhead rises with the path length. Path with 8 hops has overhead of approximately 1000 bits per second (bps), but a path with 12 hops has overhead of approximately 1700 bps. This corresponds to 175 bps increase per hop in emulated topology.

An increase is brought in by routing mechanism in use - SR-MPLS. Segment routing uses combination of source routing and MPLS encapsulation. Encapsulation amount required depends on paths complexity. All path's description must be added to the original packet at its ingress in SFC domain for source routing to work.

To eliminate overheads dependency of paths complexity in any topology we propose a use of a single segment ID for each SF path. This could potentially lead to use of hierarchy structured SIDs.

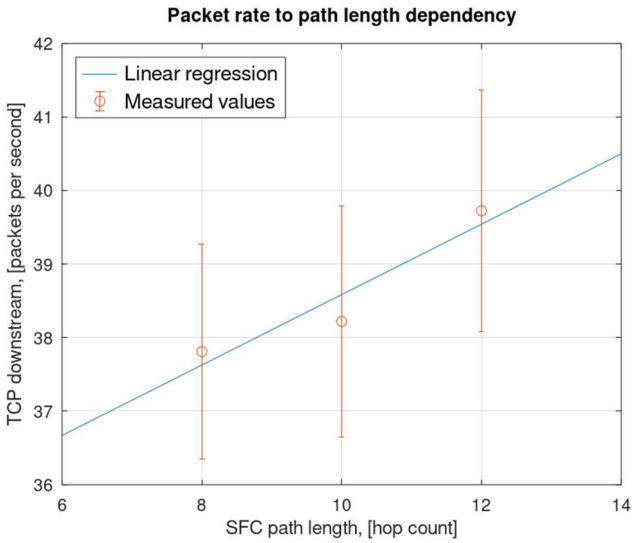

Fig. 15. Packet rate dependency

Packet rate of TCP downstream in packets per second (pps) to SF path length dependency is shown in Fig. 15.

Calculated linear regression shows a packet rate increase with longer SF paths. Packet rate is approximately 38 pps for path of 8 hops and approximately 40 pps for path of 12 hops. This corresponds to 0.5 pps increase per hop in emulated topology.

The packet per second increases was due to 1. SF path type shown in Fig. 5. in use. As the SFs were a transparent proxy servers, they were stateless and held no need for the return (upstream) TCP flow to cross them.

The affected element of flow asymmetry was TCP window. The source and destination nodes unsuccessfully tried for multiple times to arrange a window upscale. So, having more configuration dialog with no payload to carry allude to a slight packet rate increase with no added value.

To avoid the unwanted dialog, we propose use of more granular network traffic classification. For example, enforcing 3. SF path type (shown in Fig. 5.) to be in use for bidirectional communication, and 2. SF path type for use of one-way communication.

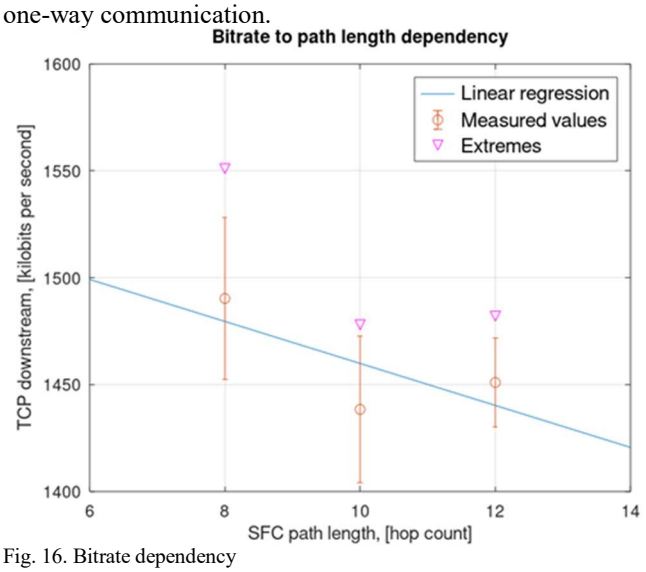

Fig. 16. Bitrate dependency

Bitrate of TCP downstream in kilobits per second (kbps) to SF paths length dependency is shown in Fig. 16. Measured values show mean points from 10 measurements with a standard deviation. Extremes show maximum of the achieved bitrate among same 10 measurements.

Although the calculated linear regression suggests a steady slope downwards, having both extremes and mean value points not following the pattern makes a contradiction.

Bitrate below 1450 kbps for 10 hop long path is the lowest among measured values. Value in this graph can be explained by cross-examining graphs in Fig. 14. and Fig. 15. A common culprit of 10 hop path falling out of linear regression is observable.

The observed occurrence is explainable by having two parameters change among emulated paths where one is paths length change, while the other is count of crossed SFs. Although, both are directly proportional (8 hop path visits 1 SF, 10 hop path visits 2 SFs, and 12 hop path visits 3 SFs), having a multiple parameter change does complexes measurement relation linearity.

Therefore, the uneven bitrate drop is an outcome of both multiple parameters change among paths and TCP window fluctuations discussed under packet rate analyses.

For a more precise performance evaluations, we suggest conduction of fine-grained comparisons. For example, path difference on a single parameter change.

C. *Future work*

The development did not go without cutting corners:

- The network traffic classification does not reach all SFC elements. Only the classifier (CL in Fig. 7. and s1 in Fig. 8.) can utilize it. This leaves no option for network traffic reclassification.
- The forwarding tables are filled statistically and designed only for emulated topology (Fig. 8). This strains the reusability of our research.
- Even though we succeeded in writing the P4 program code in a way that it is independent from any specific network path descriptors (IP addressing, MAC addressing, segment IDs e. c.), still for egress control we used an action with no tables. That might not be a correct implementation and does requires a revision.
- SF python program also uses try catch as a condition statement for label stack parsing which also should be revisited in future iterations.
- The communication with the controller was through localhost. A more realistic network models would require inbound communication.

An exploration of fundamental characteristics is only a groundwork for a more sophisticated analysis of following:

- Reactive SF path classification implementation – the ability to reclassify network traffic if the SF it's directed at is not fit to fulfil the desired service.
- Path SID forwarding – the ability to map whole path packet needs to be steered along with utilization of a single SID. This could potentially also allow structurization of SID hierarchy.
- SFC control channel implementation – enabling informational message exchange among all SFC elements and the control plane.
- Comprehensive evaluations – a fin-grained division between SF paths should be implemented. SFs individual shortcomings should be excluded from overall SF path analysis.

CONCLUSION

We used Mininet network emulator with P4-Utils and Scapy to create an emulation environment. This required development of program code in P4 and python languages. We made developed code available through GitHub [20].

Analysis shows that SR-MPLS does create a minimal overhead of 2.5 kbps for a flow of 1451 kbps that constitutes to a ratio of 1/580.

Analysis also revealed that a longer SF path does not contribute to a significant packet drop. As path with 8 hops runs at a rate of 1500 kbps and one with 12 hops runs at a rate of 1450 kbps making a drop of 12.5 kbps per hop. Thus, bitrate drop ratio is 1/120.

To limit encapsulation dependency on path length we propose use of a single SID describing the whole path in contrary of using label stack made from node, network, and adjacency SIDs.

We also propose to direct bidirectional communications both flows via same path to avoid inconsistency in path parameter configuration.

REFERENCES

[1] E. F. Kfoury, J. Crichigno and E. Bou-Harb, "An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends," in IEEE Access, vol. 9, pp. 87094-87155, 2021

[2] J. Halpern and C. Pignataro, “Service Function Chaining (SFC) Architecture”, RFC 7665, DOI 10.17487/RFC7665, October 2015, Available: https://www.rfc-editor.org/info/rfc7665

[3] P. L. Ventre et al., "Segment Routing: A Comprehensive Survey of Research Activities, Standardization Efforts, and Implementation Results," in IEEE Communications Surveys & Tutorials, vol. 23, no. 1, pp. 182-221, Firstquarter 2021

[4] Z. Liu, P. Cui, Y. Hu, Y. Dong, K. Tang and L. Xue, "A programmable data plane that supports definable computing," 2021 International Conference on Advanced Computing and Endogenous Security, Nanjing, China, 2022

[5] J. Alvarez-Horcajo, I. Martínez-Yelmo, D. Lopez-Pajares, J. A. Carral and M. Savi, "A Hybrid SDN Switch Based on Standard P4 Code," in IEEE Communications Letters, vol. 25, no. 5, pp. 1482-1485, May 2021

[6] Y. -K. Chang, H. -Y. Wang and Y. -H. Lin, "A Congestion Aware Multi-Path Label Switching in Data Centers Using Programmable Switches," 2021 IEEE International Conference on Networking, Architecture and Storage (NAS), Riverside, CA, USA, 2021

[7] X. Zhang, L. Cui, F. P. Tso and W. Jia, "Compiling Service Function Chains via Fine-Grained Composition in the Programmable Data Plane," in IEEE Transactions on Services Computing, vol. 16, no. 4, pp. 2490-2502, 1 July-Aug. 2023

[8] J. Ma, S. Xie and J. Zhao, "Flexible Offloading of Service Function Chains to Programmable Switches," in IEEE Transactions on Services Computing, vol. 16, no. 2, pp. 1198-1211, 1 March-April 2023

[9] Y. Wang, X. Zhang, L. Fan, S. Yu and R. Lin, "Segment Routing Optimization for VNF Chaining," ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 2019

[10] Noa Zilberman at University of Cambridge, P4 Tutorial Welcome, [Online] Available: https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf, last viewd June 2024

[11] P4.org Architecture Working Group, Portable switch architecture, [Online] Available: https://p4.org/p4-spec/docs/PSA-v1.1.0.html, last viewed June 2024

[12] M. Mihaeljans and A. Skrastins, "Network Topology-aware Service Function Chaining in Software Defined Network," 2020 28th Telecommunications Forum (TELFOR), Belgrade, Serbia, 2020

[13] M. Mihaeljans and A. Skrastins, "Evaluation of reactive service function path discovery in symmetrical environment," Telfor Journal, vol. 14, no. 1, pp. 2-7, 2022

[14] M. Mihaeljans and A. Skrastins, "Reactive Service Function Path Discovery Approach in Software Defined Network," 2021 29th Telecommunications Forum (TELFOR), Belgrade, Serbia, 2021

[15] M. Boucadair, “Service Function Chaining (SFC) Control Plane Components & Requirements,” draft-ietf-sfc-control-plane-08 (Informational), October 2016.

[16] Networked Systems Group at ETH Zürich, P4-Utils, [online] Available: https://nsg-ethz.github.io/p4-utils/, last viewed June 2024

[17] P4 Language Consortium, P4 Language Specification, [Online] Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html, last viewed June 2024

[18] P. Biondi, Scapy, [online] Available: https://scapy.readthedocs.io/, last viewed June 2024

[19] A. Clemm, L. Ciavaglia, and L. Z. Granville, "Intent-Based Networking - Concepts and Definitions," IRTF RFC 9315, 2022

[20] M. Mihaeljans and A. Skrastins, Program code used in this study, [Online], Available: https://github.com/MartinsMihaeljans/SFC-on-P4/tree/main, last viewed June 2024