SRv6-INT: Runtime Monitoring for Green Service Function Chaining in B5G-MEC

Xuefeng Yan, Zichen Xu, Bofan Chen and Zuqing Zhu[†]
School of Information Science and Technology, University of Science and Technology of China, Hefei, China
[†]Email: {zqzhu}@ieee.org

Abstract—**Runtime re-optimization of service function chains (SFCs) has been considered as a must-have feature to enable cost-effective SFC provisioning for the multi-access edge computing (MEC) in beyond 5G (B5G) networks. Therefore, in this work, we propose SRv6-INT, which time-multiplexes the fields of segment routing over IPv6 (SRv6) and in-band network telemetry (INT) in packets to monitor and adjust SFCs timely and efficiently. We also prototype a closed-loop network control and management (NC&M) system based on SRv6-INT to re-optimize SFC provisioning in runtime for precisely balancing the tradeoff between quality-of-service (QoS) and energy usage of SFCs. Experimental results show that with SRv6-INT, the provisioning of SFCs can be adjusted in runtime to save switch ports and CPU frequency, leading to effective energy-saving without QoS violation.**

Index Terms—**IPv6 Segment Routing (SRv6), In-band network telemetry (INT), Programmable data plane (PDP), Multi-access edge computing (MEC), Service function chaining (SFC).**

I. Introduction

Nowadays, technical innovations are reshaping the Internet consistently to adapt to the ever-increasing traffic and network services [1, 2] and to satisfy more stringent quality-of-service (QoS) demands with cost-effective and energy-efficient network technologies [3–6]. For example, the "Beyond 5G (B5G)" initiative [7] tries to integrate the latest advances on physical-layer techniques [8, 9], software-defined networking (SDN), network function virtualization (NFV) [10–12], and artificial intelligence (AI) [13, 14] to realize seamless global coverage and interconnection. This promotes the idea of multi-access edge computing (MEC), which can bring computing resources closer to end users for better QoS satisfaction. By leveraging SDN and NFV, we can simplify the network control and management (NC&M) of B5G-MEC. Specifically, we can decompose a network service into a few virtual network functions (vNFs), deploy the vNFs on commodity servers, and steer traffic through them for service function chaining (SFC).

To realize cost-efficient SFC for B5G-MEC, one needs to orchestrate network and IT resources well. As for the network part, segment routing (SR) [15] has been considered as an attractive technique. Specifically, SR lets the ingress switch of a packet flow set its routing path and operations along the path, denote the path as an ordered list of segments, and encode the segments as a stack of labels in the header of each packet in the flow. Then, the switches along the path process the packets according to the labels. In terms of standardization, SR over IPv6 (SRv6) has been specified in [16], which leverages the *Segment Routing Header* in IPv6 extension header to turn on SRv6 on a flow and encodes a stack of *Segment Lists* to indicate the flow's routing path and how packets in it should be processed at the end of each segment. Hence, the operator can customize the *Segment Lists* of an SRv6 flow to steer it through an SFC easily. This advantage makes SRv6 be considered as a bearer protocol of B5G-MEC, and thus many efforts have been devoted to implementing SRv6 in various programmable switches, including both software and hardware ones [15].

As for the IT part of SFC provisioning, vNFs are instantiated in virtual machines (VMs) or containers, whose deployment, configuration, and migration can be managed by platforms like Kubernetes (K8s) [17] and Openstack [18]. To maintain the QoS and cost-effectiveness of an SFC, we might need to re-optimize its provisioning scheme in runtime [11]. Specifically, dynamic runtime re-optimization will be essential to 1) keep the SFC's end-to-end (E2E) latency, especially for the cases where users can move or/and congestions can happen on switches/servers, and 2) improve its energy-efficiency, since dynamic traffic can cause mismatches between energy usage and workload frequently [19] (*i.e.*, IT resources assigned to the SFC need to be readjusted adaptively [20]). However, dynamic runtime re-optimization can hardly be realized without monitoring the operation of an SFC precisely and timely.

Recently, with the advances on programmable data plane (PDP), new network monitoring techniques such as in-band network telemetry (INT) [21] were developed. Specifically, INT lets packets carry telemetry instructions and be inserted with the fields that contain monitoring results, accomplishing realtime, fine-grained, and flow-oriented network monitoring. Nevertheless, excessively long packets can be generated due to repeated insertions of INT fields, which makes INT incompatible with other network innovations that also need to add fields in packets (*e.g.*, SR), because of the restriction applied by maximum transmission unit (MTU). To address this issue, we proposed and demonstrated SR-INT in [22], which time-multiplexes fields in each packet for INT and SR to not only reduce packet length but also combine SR and INT seamlessly for highly-efficient and adaptive network monitoring.

However, the SR-INT designed in [22] was not based on SRv6, which makes it incompatible with practical network systems for SFC provisioning. More importantly, the actual effects of the efficient and adaptive network monitoring achieved by SR-INT on SFC provisioning were not demonstrated in [22]. Hence, in this work, we design SRv6-INT, which time-multiplexes *Segment Lists* for SRv6 and INT fields to monitor the operation of an SFC timely. We also implement a closed-loop NC&M system based on SRv6-INT to realize the dynamic runtime re-optimization of SFC provisioning for precisely balancing the tradeoff between the QoS and energy consumption of SFCs. Specifically, we prototype our closed-loop design with commodity servers, commercial PDP switches (*i.e.*, those based on Tofino chips), and open-source software platforms (*i.e.*, ONOS and K8s), and conduct experiments to evaluate our proposal. Experimental results show that with SRv6-INT, the provisioning schemes of SFCs can be re-optimized in runtime to save switch ports and CPU frequency, leading to effective energy-saving without QoS violation.

The rest of the paper is organized as follows. Section II surveys the related work briefly. In Section III, we present our design of the closed-loop NC&M system based on SRv6-INT. The experimental demonstrations are discussed in Section IV. Finally, Section V summarizes this paper.

II. Related Work

The basic behaviors of SRv6 were standardized in [16], and how to support SRv6 in 5G networks has been discussed by 3GPP in [23]. Mayer *et al.* [24] tried to provision SFCs with SRv6 and extended the implementation of SRv6 in the Linux kernel to realize an open-source SR proxy. However, they did not address the performance monitoring or dynamic runtime re-optimization of SFCs. Recently, INT [21] has attracted intensive research interests because it can realize realtime, fine-grained, and flow-oriented network monitoring. In [25], the authors proposed to combine SR and INT for path-controllable network monitoring, but they just applied SR and INT simultaneously and did not try to reduce the accumulated overheads. To the best of our knowledge, our previous study in [22] was the only one that tried to make SR and INT benefit each other mutually with minimized overheads for efficient and adaptive network monitoring. Nevertheless, in [22], we did not design the system based on the standardized SRv6 or demonstrate the actual effects of SR-INT on SFC provisioning.

Energy-saving has always been important for 5G/B5G/6G, especially for the MEC scenarios whose power budgets are limited. The authors of [19] designed energy-aware SFC provisioning schemes for Internet-of-Things (IoT) applications in edge-cloud environments. However, the work did not address the dynamic runtime re-optimization of SFC provisioning based on network monitoring. The study in [20] tried to adjust the frequency of network processors based on the status of their line-cards (*e.g.*, the queue lengths and port utilization) for energy-saving. Although the proposal was about dynamic runtime re-optimization, it was not on SFC provisioning, and only tackled the re-optimization of switches (*i.e.*, the orchestration of network and IT resources was not considered).

III. System Design

In this section, we first introduce our design of the SRv6-INT protocol, and then describe the closed-loop NC&M system based on SRv6-INT for SFC re-optimization.

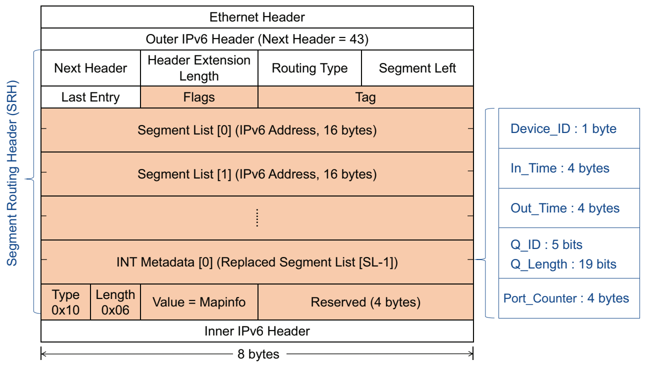


Fig. 1. Our design of the packet format for SRv6-INT.

A. Protocol Design of SRv6-INT

Fig. 1 shows the packet format that we design for SRv6-INT based on the standardized SRv6 [16]. Specifically, we modify the *Segment Routing Header* (SRH) in IPv6 extension header, and the new/modified fields are marked in yellow in Fig. 1. The *Flags* field will be filled with new values to assist the operation of SRv6-INT. Since our SRv6-INT tries to minimize the length of each packet, we replace a *Segment List* with an *INT Metadata* after the packet has finished a segment on its routing path. As shown in Fig. 1, each *Segment List* is in 16 bytes and stores the IPv6 address of the last node of a segment (*i.e.*, an endpoint), *SL* denotes the total number of segments on the packet's routing path, and the INT fields that can be included in each *INT Metadata* are detailed on the right subplot of the figure. Specifically, each *INT Metadata* tells the status of the last hop of an experienced segment, and can contain 5 INT fields at most. The type-length-value (TLV) tuple in the modified SRH defines the *MapInfo* for INT data collection, *i.e.*, each of the last 5 bits of the *Value* field denotes whether a corresponding INT field should be filled with status data. The last 4 bytes are reserved for future use and stuff the SRH to make its length a multiple of 8 bytes [16].

In *INT Metadata*, *Device_ID* tells the unique ID of the last switch of a segment, *In_Time*/Out_Time are for the time when the packet arrives at/leaves the switch, respectively. *Q_ID* and *Q_Length* are considered as one INT field, which denotes the ID and length of the queue that the packet was stored in, respectively. *Port_Counter* stores the instant bandwidth usage of the packet's flow. In the *Value* field, the highest used bit is for *Device_ID*, and the remaining used bits follow the aforementioned order to denote the other INT fields. Hence, to collect all the 5 INT fields, we should set *Value*$= 0x001f$.

To support SRv6-INT, we follow the principle of SRv6 [16] to define three actions, *i.e.*, *H.Encaps.INT*, *End.T.INT*, and *End.DT.INT*, for modifying packets at the ingress and endpoint switches in an SRv6 domain. *Algorithm* 1 shows the operations of SRv6-INT on an ingress, endpoint and transit switch of a packet. *Lines* 3-7 are for the action of *H.Encaps.INT* on an ingress switch, which encapsulates the SRH and updates outer IPv6 header accordingly. The action of *End.T.INT* for an endpoint switch is described with *Lines* 8-28. Here, we define two mechanisms: 1) the "plain SRv6-INT" if the endpoint switch does not directly connect to one or more vNFs (*Lines* 11-15), and 2) the “SRv6-INT for SFC” otherwise (*Lines* 16-28). Specifically, we design the SRv6-INT for SFC to collect the total time of a packet being processed by switch→vNF(s)→switch and switch only, and overwrite the *In_Time/Out_Time* fields in the corresponding *INT Metadata* to store the results, respectively (*Lines* 23-25). *Lines* 29-32 are for the action of *End.DT.INT* on the last endpoint switch on the packet’s routing path (*i.e.*, the egress switch). Finally, the operation on a transit switch is described in *Lines* 36-37.

Algorithm 1: Operations of SRv6-INT on a Switch

```
1  receive a packet;
2  if SRv6-INT is enabled then
3      if the packet's IPv6 header does not contain an SRH then
4          // ingress switch: H.Encaps.INT
5          encode SRH in IPv6 header and update outer IPv6 header;
6          set SRH.Flags as $0x80$ ;
7          set SRH.Tag as $0x0000$ or $0x4000$ ;
8      else if SRH.Segment Left $> 0$ then
9          $i =$ SRH.Segment Left;
10         // endpoint switch: End.T.INT
11         if SRH.Tag $= 0x0000$ then
12             // plain SRv6-INT
13             replace Segment List[i] with INT Metadata[SL-1-i];
14             decrease SRH.Segment Left by 1;
15             update outer IPv6 header;
16         else if SRH.Tag $= 0x4000$ then
17             // SRv6-INT for SFC: before vNF
18             set SRH.Tag as $0x8000$;
19             replace Segment List[i] with INT Metadata[SL-1-i];
20         else if SRH.Tag $= 0x8000$ then
21             // SRv6-INT for SFC: after vNF
22             set SRH.Tag as $0x4000$;
23             collect packet's ingress and egress time as $t_{in}$ and $t_{out}$;
24             INT Metadata[SL-1-i].In_Time $- = t_{out}$;
25             INT Metadata[SL-1-i].Out_Time $- = t_{in}$;
26             decrease SRH.Segment Left by 1;
27             update outer IPv6 header;
28         end
29     else
30         // endpoint switch: End.DT.INT
31         copy outer IPv6 header with SRH to data analyzer;
32         remove outer IPv6 header;
33     end
34     submit the packet to IPv6 FIB lookup;
35 end
36 /* transit switch: plain IPv6 forwarding          */
37 submit the packet to IPv6 FIB lookup;
```

B. System Architecture

Fig. 2 shows the network architecture of B5G-MEC, which uses SRv6-INT to monitor and re-optimize SFCs in runtime. Specifically, we assume that the SRv6 domain of the B5G-MEC is based on PDP switches (*e.g.*, those with Tofino chips that can be programmed with the P4 language). Hence, SRv6-INT can be supported by programming the PDP switches, and for runtime re-optimization, the switches are managed by the SDN controller based on ONOS through P4 Runtime. According to the definition of 3GPP [23], links within the SRv6 domain serve as the N9 interface, each ingress switch connects to the radio access network (RAN) through the N3 interface, and each egress switch connects the core network to a cloud DC through the N6 interface. To provision SFCs, we place edge-computing platforms (*e.g.*, Linux servers) in the SRv6 domain and attach them to the PDP switches, and vNFs can be instantiated in the cloud DC too. With SRv6-INT, the status of each SFC can be monitored in a realtime and fine-grained manner within the SRv6 domain, and packets carrying INT results will be mirrored to the data analyzer (DA) attached to an egress switch before leaving the SRv6 domain. Then, the DA will process the INT results and provide suggestions to the SDN controller for re-optimizing SFC provisioning dynamically, realizing a closed-loop NC&M system.

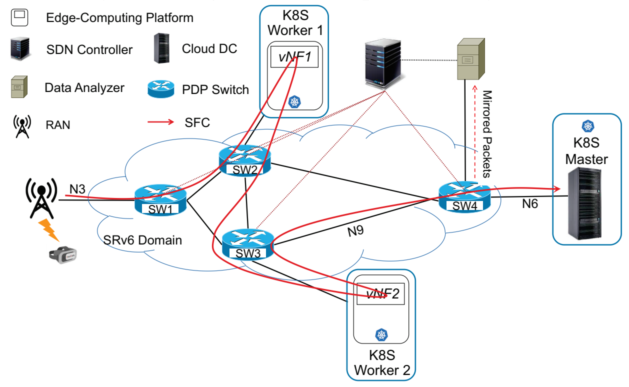


Fig. 2. Network architecture of B5G-MEC with SRv6-INT for SFC.

For better resource and energy efficiency, we assume that vNFs are instantiated in docker containers and K8s is used to deploy and configure vNFs. Specifically, we deploy a K8s master node in the cloud DC, and use it to manage each edge-computing platform in the SRv6 domain as a worker node for deploying, configuring and removing vNFs.

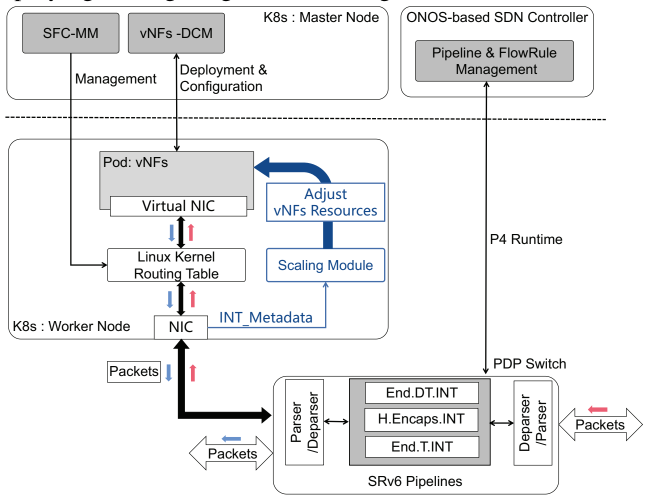


Fig. 3. Functional modules and operation principle of SRv6-INT for SFC.

C. Functional Modules and Operation Principle

Fig. 3 explains the functional modules and their operation principle for applying SRv6-INT on SFCs for dynamic run-time re-optimization. The control plane consists of an SDN controller for centralized NC&M and a K8s master node for IT resource management. The ONOS-based SDN controller manages the PDP switches in the SRv6 domain to steer SRv6-INT traffic through the vNFs on edge-computing platforms and cloud DC to realize SFCs. The K8s master consists of a vNF deployment/configuration module (vNF-DCM) and a SFC management module (SFC-MM), where the latter is for setting up internal connections among the vNFs deployed on one server. Each K8s worker uses a physical line-card (NIC) to connect to a PDP switch directly. After the NIC, packets are forwarded to the vNFs in containers according to the internal routing table in Linux kernel, which is set up by the internal routing module according to the configurations from the SFC-MM in the K8s master. Note that, in K8s, each pod contains a group of containers and is the unit for vNF deployment.

Meanwhile, we design a scaling module in each K8s worker, which will read *Out_Time* and *Port_Counter* fields in the *INT Metadata* about the underlying PDP switch to get the throughput of each input flow and determine how to correctly adjust the IT resources (*e.g.*, CPU cycles and memory space) allocated to the vNFs in each pod for runtime re-optimization. Note that the energy consumption of a server is normally proportional to the frequency of its CPU [26]. Hence, we can use the scaling module to adjust the CPU frequency that is allocated to each vNF according to the vNF's current traffic processing throughput, such that noticeable energy saving can be achieved without sacrificing packet processing in the vNF. On the other hand, from the network perspective, ONOS-based SDN controller can get the information about vNF deployment from K8s and obtain end-to-end SFC monitoring results from a DA in runtime. Therefore, it can leverage SRv6-INT to make the paths of SFCs share links, grooming traffic to fewer switch ports and keeping the remaining ports idle for energy saving.

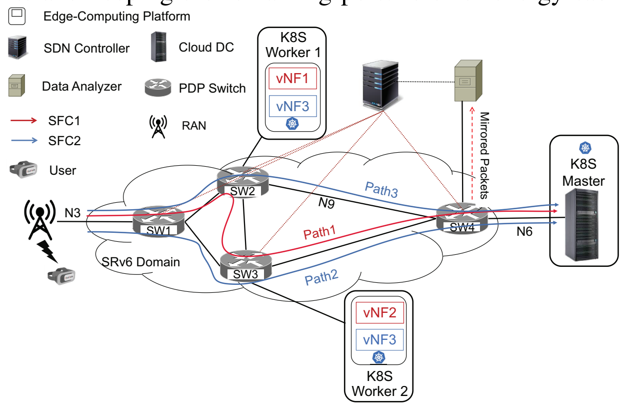


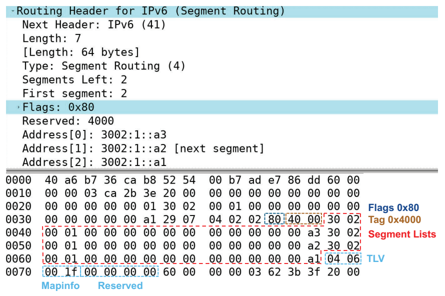
Fig. 4. Experimental setup.

IV. Experimental Demonstrations

In this section, we conduct experiments to demonstrate the effectiveness of SRv6-INT on realizing green SFCs.

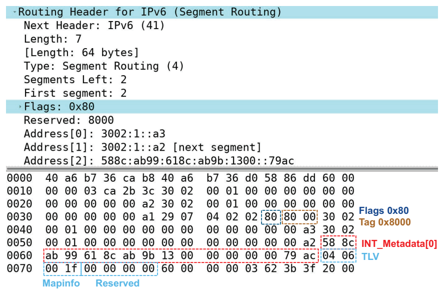
A. Experimental Setup

We build an experimental testbed for SRv6-INT, according to the setup in Fig. 4. Here, we emulate the end users behind the N3 interface with a commercial traffic generator, the edge-computing platforms are servers, the cloud DC is emulated with a high-performance server, and the SDN controller and DA also run on servers. The PDP switches are the commercial 32-port ones based on Tofino chips, which are interconnected with 40-Gbps links in the SRv6 domain. The edge-computing platforms and cloud DC connect 10-Gbps ports to PDP switches. The IPv6 addresses of *vNF1*, *vNF2* and the cloud DC are 3002:1::a1, 3002:1::a2, and 3002:1::a3, respectively.



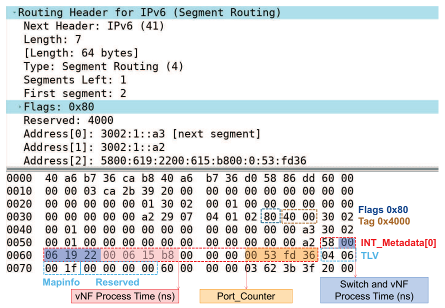
```
Routing Header for IPv6 (Segment Routing)
  Next Header: IPv6 (41)
  Length: 7
  [Length: 64 bytes]
  Type: Segment Routing (4)
  Segments Left: 2
  First segment: 2
  Flags: 0x80
  Reserved: 4000
  Address[0]: 3002:1::a3
  Address[1]: 3002:1::a2 [next segment]
  Address[2]: 3002:1::a1
0000  40 a6 b7 36 ca b8 52 54  00 b7 ad e7 86 dd 60 00
0010  00 00 03 ca 2b 3e 20 00  00 00 00 00 00 00 00 00
0020  00 00 00 00 00 01 30 02  00 01 00 00 00 00 00 00
0030  00 00 00 00 00 a1 29 07  04 02 02 80 40 00 30 02
0040  00 01 00 00 00 00 00 00  00 00 00 00 00 a3 30 02
0050  00 01 00 00 00 00 00 00  00 00 00 00 00 a2 30 02
0060  00 01 00 00 00 00 00 00  00 00 00 00 00 a1 04 06
0070  00 1f 00 00 00 00 60 00  00 00 03 62 3b 3f 20 00
```

(a) Packet at output of *SW1*



```
Routing Header for IPv6 (Segment Routing)
  Next Header: IPv6 (41)
  Length: 7
  [Length: 64 bytes]
  Type: Segment Routing (4)
  Segments Left: 2
  First segment: 2
  Flags: 0x80
  Reserved: 8000
  Address[0]: 3002:1::a3
  Address[1]: 3002:1::a2 [next segment]
  Address[2]: 588c:ab99:618c:ab9b:1300::79ac
0000  40 a6 b7 36 ca b8 40 a6  b7 36 d0 58 86 dd 60 00
0010  00 00 03 ca 2b 3c 30 02  00 01 00 00 00 00 00 00
0020  00 00 00 00 00 a2 30 02  00 01 00 00 00 00 00 00
0030  00 00 00 00 00 a1 29 07  04 02 02 80 80 00 30 02
0040  00 01 00 00 00 00 00 00  00 00 00 00 00 a3 30 02
0050  00 01 00 00 00 00 00 00  00 00 00 00 00 a2 58 8c
0060  ab 99 61 8c ab 9b 13 00  00 00 00 00 79 ac 04 06
0070  00 1f 00 00 00 00 60 00  00 00 03 62 3b 3f 20 00
```

(b) Packet sent to *vNF1* by *SW2*



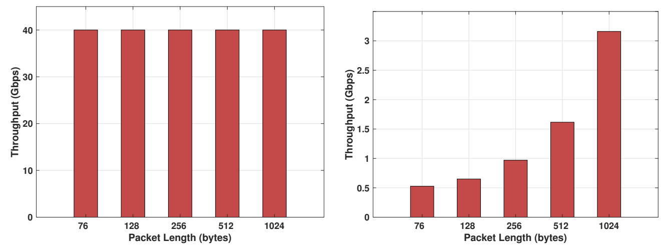
```
Routing Header for IPv6 (Segment Routing)
  Next Header: IPv6 (41)
  Length: 7
  [Length: 64 bytes]
  Type: Segment Routing (4)
  Segments Left: 1
  First segment: 2
  Flags: 0x80
  Reserved: 4000
  Address[0]: 3002:1::a3 [next segment]
  Address[1]: 3002:1::a2
  Address[2]: 5800:619:2200:615:b800:0:53:fd36
0000  40 a6 b7 36 ca b8 40 a6  b7 36 d0 58 86 dd 60 00
0010  00 00 03 ca 2b 39 20 00  00 00 00 00 00 00 00 00
0020  00 00 00 00 00 01 30 02  00 01 00 00 00 00 00 00
0030  00 00 00 00 00 a2 29 07  04 01 02 80 40 00 30 02
0040  00 01 00 00 00 00 00 00  00 00 00 00 00 a3 30 02
0050  00 01 00 00 00 00 00 00  00 00 00 00 00 a2 58 00
0060  06 19 22 00 06 15 b8 00  00 00 00 53 fd 36 04 06
0070  00 1f 00 00 00 00 60 00  00 00 03 62 3b 3f 20 00
```

(c) Packet sent to *SW3* by *SW2*

Fig. 5. Wireshark captures of collected packets.

B. Functional Verification

As shown in Fig. 4, *SFC1* in our experiments uses the path: *User*→*SW1*→*SW2*→*vNF1*→*SW2*→*SW3*→*vNF2*→*SW3* →*SW4*→*Cloud DC*, and thus we naturally divide the path into three segments whose endpoints are *vNF1*, *vNF2* and the cloud DC. According to the principle of SRv6-INT, we encode *Segment List*$[0]$ = 3002:1::a3, *Segment List*$[1]$ = 3002:1::a2, and *Segment List*$[2]$ = 3002:1::a1. Fig. 5 shows the Wireshark captures of packets collected at different locations on the SFC's path. In Fig. 5(a), we can see that as the ingress switch of *SFC1*, *SW1* encodes SRH correctly and sets *SRH.Tag* as $0x4000$ to indicate that the SRv6-INT is for SFC. After the packet is processed by *SW2* for the first time, Fig. 5(b) indicates that *SW2* has replaced *Segment List*$[2]$ with *INT Metadata*$[0]$ and updated *SRH.Tag* as $0x8000$. Here, *INT Metadata*$[0]$ includes the status of *SW2* when processing the packet, and *SRH.Tag*$=0x8000$ means that the packet's next hop is a vNF (*i.e.*, *vNF1*). Finally, after processing the packet for the second time, *SW2* has overwritten the *In_Time* and *Out_Time* fields in *INT Metadata*$[0]$ to include the total processing time of *SW2*→*vNF1*→*SW2* and *vNF1*, respectively. The Whireshark captures in Fig. 5 verify that the functionalities of SRv6-INT have been implemented correctly.



(a) Throughput of PDP switch (b) Throughput of vNF

Fig. 6. Packet processing throughput of SRv6-INT system.

Next, we measure the packet processing throughput of our SRv6-INT system and show the results in Fig. 6. In Fig. 6(a), we can see that the PDP switch's throughput can always reach its line-rate (*i.e.*, 40 Gbps) regardless of the packet sizes[1], which confirms that our procedure of SRv6-INT will not degrade the packet processing performance of PDP switches. Fig. 6(b) indicates that the throughput of a vNF is much lower, especially for small-sized packets. This is actually expected, since the vNF processes packets in a software system, *i.e.*, its performance can be limited by a number of factors, such as Linux kernel data path and containers resource occupation. The results in Fig. 7 confirm the analysis above, as when the throughput of the flow (with a packet length of $1,024$ bytes) to the vNF exceeds 3 Gbps, the processing time of the vNF increases sharply from below 0.3 ms to more than 4.5 ms.

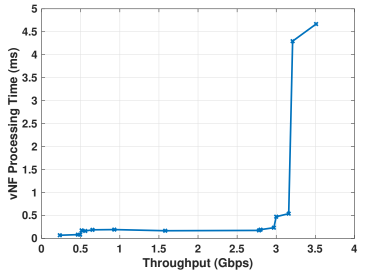
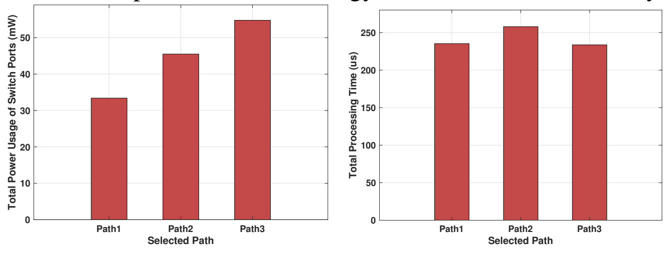


Fig. 7. Results on vNF processing time (packet length is $1,024$ bytes).

C. Network Re-optimization with SRv6-INT for Green SFCs

Next, we conduct experiments to validate the capability of SRv6-INT on network re-optimization. This time, we consider that there are two SFCs in the testbed, as shown in Fig. 4. The SDN controller uses SRv6-INT to monitor the total processing latency and bandwidth usage of each SFC, and if possible, it will instruct the SFCs to share switch ports for saving energy. Specifically, when *SFC1* is active, *SFC2* needs to be provisioned and its service requirement is: *User→vNF3→Cloud DC*. Therefore, we can set *SFC2* up with any of the three paths in Fig. 4 (*i.e.*, *Path1*, *Path2* and *Path3*). Fig. 8(a) shows the total power usages of the active switch ports on the three paths, and we can see that *Path1* is the most energy-efficient one, because *SFC1* and *SFC2* share the most switch ports in this case. Meanwhile, the total processing time of *SFC2*, which is measured with SRv6-INT, is plotted in Fig. 8(b). We observe that selecting *Path1* to provision *SFC2* will not cause intolerably-long total processing time (*i.e.*, the total processing time of *Path1* is similar to that of *Path3* and less than that of *Path2*). Therefore, provisioning *SFC2* with *Path1* achieves the best tradeoff between power consumption and total processing latency, and the experimental results in Fig. 8 verify that the runtime monitoring achieved by SRv6-INT can facilitate the network re-optimization for energy-efficient SFCs effectively.



(a) Power usage of paths (b) Total processing time of paths

Fig. 8. Network re-optimization with SRv6-INT.

D. vNF Re-optimization with SRv6-INT for Green SFCs

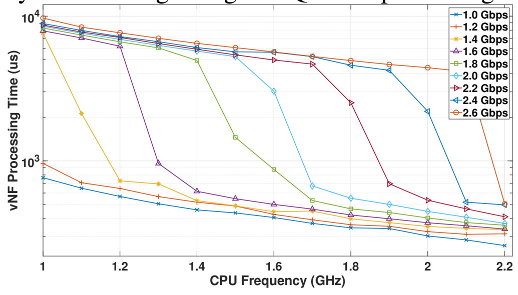
Finally, we would like to verify that with our SRv6-INT, vNF re-optimization can also be achieved for green SFCs. In order to assist the scaling module we designed in Fig. 3, we first measure the processing time of a vNF that is deployed in one K8s worker node, when the vNF has been allocated with different CPU frequencies and is processing traffic at different data-rates (*i.e.*, $1\sim2.4$ Gbps with a packet length of $1,024$ bytes). The results are illustrated in Fig. 9(a), which indicates that except for the cases of 1 and 1.2 Gbps, each data-rate has a cut-off CPU frequency below which the processing time of the vNF will increase dramatically. For instance, when the vNF is processing traffic at 1.6 Gbps (*i.e.*, the purple line in Fig. 9(a)), vNF processing time increases dramatically when the vNF's CPU frequency is below 1.3 GHz. The results in Fig. 9(a) provide the guideline for the scaling module to adjust the CPU frequency allocated to each vNF. Specifically, when the scaling module gets a flow's data-rate by checking the *INT Metadata* encoded in the flow's packets, it should make sure that the CPU frequency allocated to the flow's vNF is higher than the corresponding cut-off CPU frequency.

Then, to evaluate the performance of our proposal on vNF re-optimization, we use the traffic generator to generate a flow with the dynamic traffic profile in Fig. 9(b) and send it to a vNF in our testbed. Then, we consider three scenarios in the experiments: two with fixed CPU frequencies at 1.8 and 2.2 GHz, respectively (*i.e.*, the scaling module is turned off), and one with the scaling module on. Fig. 9(c) shows how the CPU frequency allocated to the vNF changes in the three scenarios. It can be seen that when the scaling module is turned on, our SRv6-INT system does adjust the CPU frequency to adapt to the traffic changes. Specifically, when the traffic's data-rate increases gradually from 1 to 2.6 Gbps, the scaling module increases the CPU frequency from 1.1 to 2.2 GHz adaptively.

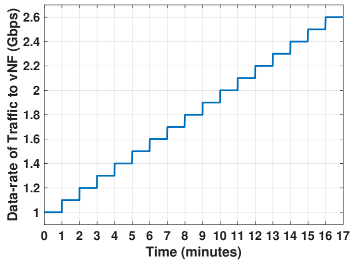
The results about how the vNF processing time changes in the process are plotted in Fig. 9(d). We observe that when the CPU frequency to the vNF is fixed at 2.2 GHz, the vNF processing time will not encounter any sharp increase in the whole experiment. However, the vNF processing time indeed increases gradually with the traffic's data-rate from ~250 to

[1]For fair comparison, the packet length in Fig. 6 excludes the SRH's length.

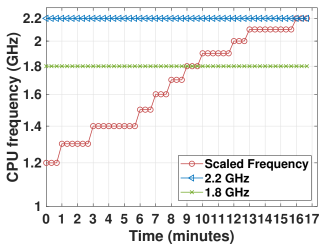
$\sim$500 μs. On the other hand, when the CPU frequency is fixed at 1.8 GHz, the vNF processing time increases dramatically after when the traffic's data-rate reaches 2.2 Gbps. The vNF processing time from the scenario with the scaling module on (*i.e.*, the red curves in Figs. 9(c) and 9(d)) are the most promising, because our SRv6-INT system successfully limits the vNF processing time within $[500, 600]$ μs by adjusting the CPU frequency timely and adaptively. The results in Fig. 9 confirm that our proposed SRv6-INT system realizes runtime vNF re-optimization to improve the energy-efficiency of SFCs effectively without degrading the QoS on processing latency.



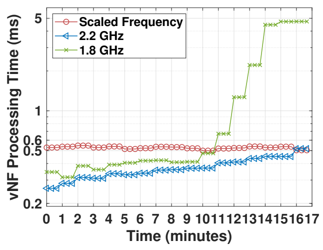
(a) *vNF* processing time *versus* CPU frequency



(b) Dynamic traffic going into vNF



(c) CPU frequency



(d) vNF processing time

Fig. 9. vNF re-optimization with SRv6-INT.

V. Conclusion

In this paper, we designed and experimentally demonstrated a SRv6-INT system, which combined SRv6 and INT effectively for monitoring SFCs timely and implementing closed-loop NC&M to realize dynamic runtime re-optimization of SFC provisioning, such that the tradeoff between the QoS and energy consumption of SFCs can be precisely balanced. We prototyped our design and conducted experiments to verify that with SRv6-INT, the provisioning schemes of SFCs can be re-optimized in runtime to save switch ports and CPU frequency, leading to effective energy-saving without QoS violation.

Acknowledgments

This work was supported by NSFC project 61871357 and Fundamental Fund for Central Universities (WK3500000006).

References

[1] P. Lu *et al.*, "Highly-efficient data migration and backup for Big Data applications in elastic optical inter-datacenter networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[2] Z. Pan *et al.*, "Advanced optical-label routing system supporting multicast, optical TTL, and multimedia applications," *J. Lightw. Technol.*, vol. 23, pp. 3270–3281, Oct. 2005.

[3] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.

[4] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[5] W. Shi, Z. Zhu, M. Zhang, and N. Ansari, "On the effect of bandwidth fragmentation on blocking probability in elastic optical networks," *IEEE Trans. Commun.*, vol. 61, pp. 2970–2978, Jul. 2013.

[6] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 12–20, Mar./Apr. 2017.

[7] K. Samdanis and T. Taleb, "The road beyond 5G: A vision and insight of the key technologies," *IEEE Netw.*, vol. 34, pp. 135–141, Mar./Apr. 2020.

[8] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.

[9] R. Proietti *et al.*, "Experimental demonstration of machine-learning-aided QoT estimation in multi-domain elastic optical networks with alien wavelengths," *J. Opt. Commun. Netw.*, vol. 11, pp. A1–A10, Jan. 2019.

[10] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[11] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[12] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. of INFOCOM 2014*, pp. 1–9, Apr. 2014.

[13] W. Lu *et al.*, "AI-assisted knowledge-defined network orchestration for energy-efficient data center networks," *IEEE Commun. Mag.*, vol. 58, pp. 86–92, Jan. 2020.

[14] X. Chen *et al.*, "Deep-RMSA: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks," in *Proc. of OFC 2018*, pp. 1–3, Mar. 2018.

[15] P. Ventre *et al.*, "Segment Routing: A comprehensive survey of research activities, standardization efforts, and implementation results," *IEEE Commun. Surveys Tuts.*, vol. 23, pp. 182–221, First Quarter 2021.

[16] C. Filsfils *et al.*, "Segment routing over IPv6 (SRv6) network programming," *RFC 8986*, Feb. 2021. [Online]. Available: https://datatracker.ietf.org/doc/rfc8986/.

[17] Kubernetes. [Online]. Available: https://kubernetes.io/.

[18] Openstack. [Online]. Available: https://www.openstack.org/.

[19] N. Thanh *et al.*, "Energy-aware service function chain embedding in edge-cloud environments for IoT applications," *IEEE Internet Things J.*, vol. 8, pp. 13 465–13 486, Sept. 2021.

[20] Q. Yu, T. Znati, and W. Yang, "Energy-efficient, QoS-aware packet scheduling in high-speed networks," *IEEE J. Sel. Areas Commun.*, vol. 33, pp. 2789–2800, Dec. 2015.

[21] C. Kim *et al.*, "In-band network telemetry (INT)," *Tech. Spec.*, Jun. 2016. [Online]. Available: https://p4.org/assets/INT-current-spec.pdf.

[22] Q. Zheng, S. Tang, B. Chen, and Z. Zhu, "Highly-efficient and adaptive network monitoring: When INT meets segment routing," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, pp. 2587–2597, Sept. 2021.

[23] D. Chandramouli and T. Sun, "System architecture for the 5G system," *3GPP Specification #: 23.501 (v17.4.0)*, Mar. 2022.

[24] A. Mayer *et al.*, "An efficient Linux kernel implementation of service function chaining for legacy VNFs based on IPv6 segment routing," in *Proc. of NetSoft 2019*, pp. 1–9, Jun. 2019.

[25] T. Pan *et al.*, "INT-path: Towards optimal path planning for in-band network-wide telemetry," in *Proc. of IEEE INFOCOM 2019*, pp. 487–495, Apr. 2019.

[26] C. Claude *et al.*, "Dynamic frequency scaling for energy consumption reduction in synchronous distributed applications," in *Proc. of IEEE ISPA 2014*, pp. 225–230, Aug. 2014.