Highly-Efficient and Adaptive Network Monitoring: When INT Meets Segment Routing

Qitao Zheng, Shaofei Tang, Bofan Chen, and Zuqing Zhu, *Senior Member, IEEE*

***Abstract*—The rapid development of software-defined networking (SDN) has promoted the idea of programmable data plane (PDP), which opens up unprecedented opportunities for realizing powerful and timely network monitoring. This work explores the advantages of PDP to merge two famous techniques (*i.e.*, the segment routing (SR) and in-band network telemetry (INT)) seamlessly for highly-efficient and adaptive network monitoring. Specifically, by leveraging the protocol-oblivious forwarding (POF), we propose SR-INT, which time-multiplexes the header fields in each packet for INT and SR, and keeps packet length constant end-to-end even though both INT and SR are used. Hence, our proposal can enjoy the benefits of INT and SR, while avoiding the accumulated overheads due to simultaneous usage. We design the packet format of SR-INT, and lay out its packet processing procedure to guarantee that the configuration of SR-INT can be adjusted dynamically to adapt to the requirements of network monitoring. We implement and experimentally demonstrate SR-INT in a POF-based SDN environment. Our results show that SR-INT not only reduces the bandwidth overheads of using SR and INT simultaneously but also simplifies the operations in software-based POF switches.**

***Index Terms*—Segment routing (SR), In-band network telemetry (INT), Software-defined networking (SDN), Protocol-oblivious forwarding (POF), Network monitoring and troubleshooting.**

I. Introduction

NOWADAYS, the Internet is being reshaped consistently by technical innovations, to adapt to the skyrocketing of new network services and the emerging of unique and stringent quality-of-service (QoS) demands [1–3]. For instance, the global deployments of datacenters have motivated operators to upgrade their transport networks from fixed-grid wavelength-division multiplexing (WDM) to flexible-grid elastic optical networking (EON) [4–6]. While to improve the flexibility and efficiency of network control and management (NC&M), software-defined networking (SDN) [7–10], network function virtualization (NFV) [11–14], and virtual network slicing [15–17] have been widely considered in production networks. Although these new network infrastructures and technologies are making the Internet more efficient, programmable and application-aware, they also increase the complexity of NC&M and cause network elements more prone to faults [18, 19]. This has stimulated unprecedented demands for realtime, adaptive and efficient network monitoring techniques [20].

Nevertheless, traditional polling-based network monitoring techniques (*e.g.*, SNMP [21]) can hardly satisfy the aforementioned demands. This is because the NC&M system pulls status data from network elements periodically, which is neither real-time nor flow-oriented. Recently, with momentum gained from the programmable data plane (PDP) [22, 23], network monitoring is becoming more powerful, timely and efficient. Specifically, PDP enables an operator to customize the data plane logic of its switches, *i.e.*, defining new packet fields and programming packet processing pipelines.

According to the protocol independent forwarding (PIF) project of open network foundation (ONF) [24], PDP can be realized with two approaches, which are the programming protocol-independent packet processors (P4) [22] and protocol-oblivious forwarding (POF) [23]. P4 provides the guidance on how to write and compile packet processing programs, and with the P4 language, one can program a PDP switch in two stages, *i.e.*, configuration and runtime [24]. POF takes a different approach to program a PDP switch in runtime by installing protocol-oblivious flow tables and composing packet processing pipelines with them, and to realize this, it defines the underlying primitive instruction set [25].

In-band network telemetry (INT) [26] is one of the most famous and successful PDP-enabled network monitoring techniques. Specifically, based on the INT command that has been precoded in the header of an application packet, each PDP switch on the packet's routing path collects its own status when processing the packet, encodes the status as specific INT fields, and inserts them in the header of the packet. Therefore, INT can monitor the end-to-end performance of a flow by collecting the per-packet/per-hop information of it. This successfully overcomes the delay and consistency issues of polling-based network monitoring, visualizes a network in a real-time and fine-grained way, and eases the provisioning of applications with stringent quality-of-service (QoS) demands (*e.g.*, video streaming [27–29]). Hence, people have designed and implemented various INT systems based on P4 [30–32] and POF [33, 34], to explore the technique's benefits.

However, INT also has its drawbacks, *e.g.*, 1) it can degrade the throughput of packet processing in a PDP switch (especially the software-based one) because of the need of invoking *AddField* actions frequently to insert INT fields, and 2) it can generate excessively long packets due to the repeated insertions of INT fields. These drawbacks restrict the usages of INT, especially for the second one. This is because other network innovations may also add new header fields in packets. For instance, the well-known segment routing (SR) [35] lets the source of a packet flow choose its routing path, represents the path with an ordered list of segments, and encodes them as a stack of labels in the header of each packet that belongs to the flow. Then, each switch along the path only

Q. Zheng, S. Tang, B. Chen, and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China (email: zqzhu@ieee.org).

Manuscript received on November 16, 2020.

needs to forward the packets according to the encoded labels. Therefore, SR simplifies the message exchange between the control and data planes, and realizes end-to-end policy without creating any per-flow state in the network [35]. This makes it promising to apply SR to traffic engineering, failure protection, *etc*. Nevertheless, as INT and SR both need to encode a stack of header fields in a packet, they might not be compatible with each other due to the upper limit on packet length (*i.e.*, the maximum transmission unit (MTU)).

To the best of our knowledge, the aforementioned dilemma has not been addressed in the literature yet. Hence, in this work, we study how to mitigate the overheads of INT and SR, and combine them seamlessly to realize highly-efficient and adaptive network monitoring. We first design the label field of SR to make its length equal to that of a bundle of INT fields, and then propose the procedure to replace an SR label field with a bundle of INT fields at the last switch of each path segment. To this end, our proposed system, namely, SR-INT, time-multiplexes the header fields in each packet for INT and SR, and keeps the packet's length constant end-to-end even though both INT and SR are used. This enables our proposal to explore the benefits of INT and SR, while avoiding the accumulated overheads due to simultaneous usage. Moreover, to ensure the adaptivity of SR-INT, we also incorporate implementations in the control plane such that the configuration of path segments can be adjusted dynamically to adapt to the requirements of network monitoring.

We implement SR-INT in a POF-based SDN environment. Specifically, we expand the Open vSwitch (OVS) platform [36] to realize the data plane functionalities of SR-INT, while its control plane is programmed based on another open-source platform, *i.e.*, ONOS [37]. To achieve closed-loop network automation, we also realize a home-made INT collection and data analytics module that can capture, parse and analyze the INT data carried by packets with a high throughput, and then provide timely and accurate suggestions to the control plane for network readjustments. The SR-INT system is verified and evaluated with the experiments in a real network testbed. Experimental results demonstrate that SR-INT reduces not only the overheads of using SR and INT simultaneously but also the operation complexity in OVS, and it achieves highly-efficient and adaptive network monitoring for fault diagnosis and can recover the network from soft-failures quickly.

The rest of the paper is organized as follows. Section II introduces the background of POF and conducts a brief literature survey. We present the design of SR-INT in Section III, and the details of system implementation are discussed in Section IV. The experimental demonstrations are shown in Section V. Finally, Section VI summarizes the paper.

II. Background and Related Work

In this section, we first briefly describe the operation principle of POF to provide a context for our design of SR-INT, and then conduct a literature survey on related researches.

A. Review on Protocol-Oblivious Forwarding (POF)

POF presumes a typical SDN architecture that is similar to the one considered for OpenFlow, *i.e.*, a centralized control plane manages the behaviors of the switches in the data plane by installing flow tables in them [23, 38]. Meanwhile, different from the protocol-dependent scheme used in OpenFlow, POF refers to packet fields in a more generic manner, which makes its field matching and packet processing protocol-independent. More specifically, POF describes a packet field as a tuple <*offset, length*>, where *offset* tells the bit offset of the field to denote its location in a packet, and *length* represents the length of the field in bits [38]. Then, each entry in a flow table refers to its match field(s) in the format of <*offset, length*>, and specifies the corresponding match action(s) with the instructions defined in the POF instruction set (POF-FIS) [25]. Here, the instructions/actions in POF-FIS also operate based on <*offset, length*>, and thus a POF-enabled PDP switch can manipulate any segment of bits in a packet freely.

This makes the specification of POF more flexible and compact than that of OpenFlow. For example, with OpenFlow v1.5 [39], we need to first determine the protocol in use and then select the right action (*e.g., PushMPLS* and *PushVLAN*) to add a field in one packet. This is much more simplified in POF, because POF-FIS only defines a generic *AddField* action, and by using the action with a tuple <*offset, length*>, we can insert any type of fields at any location in a packet. In a POF-enabled PDP switch, the procedures of packet processing are defined as pipelines, each of which consists of one or multiple stages of flow tables [9]. To steer packets through each pipeline, POF-FIS defines the *GotoTable* action. Specifically, after being processed by one flow table, a packet can be forwarded to the next one defined in its packet processing pipeline with *GotoTable*. Meanwhile, POF allows each PDP switch to allocate a metadata memory to buffer the information about the switch itself or/and packets [40], for packet processing. POF-FIS defines the instructions to read/write data in the metadata memory, which are also based on <*offset, length*>.

B. Related Work

The technical specifications of INT were released in [26] to explain how to leverage this PDP-enable technique for real-time and flow-oriented network monitoring. Based on P4, the studies in [31, 41] demonstrated INT in software-based Mininet environments. Kim *et al.* [31] tried to debug a network by using INT to collect HTTP latency instantaneously. The authors of [41] applied data analytics on the telemetry data collected by INT to realize knowledge-defined networking. Meanwhile, a few hardware-based INT implementations with P4 can be seen in [30, 42]. With high-performance field programmable gate array (FPGA) [30] or application-specific integrated circuit (ASIC) [42], INT has been demonstrated to collect per-packet information at 100 Gbps line-rate. However, all the P4-based INT implementations mentioned above assumed that INT fields would be inserted on per-packet basis, and did not try to reduce the overheads of INT.

In [32], a P4-based selective INT approach was proposed to sample packets for inserting INT fields, and thus the overheads of INT could be reduced significantly. A similar idea was considered in [20], and moreover, the authors also expanded the applications of INT to visualize multilayer packet-over-optical networks in realtime. Note that, as P4-based PDP

switches are programmed in two stages (*i.e.*, configuration and runtime) [24], they can hardly readjust their packet processing pipelines in runtime. This makes it difficult for P4-based INT approaches to change the locations to collect telemetry data and the types of telemetry data to collect in runtime. On the other hand, by leveraging the flexibility of POF, INT can be realized with better runtime programmability. For instance, we designed and implemented a runtime-programmable selective INT scheme (Sel-INT) based on POF in [34]. Although the studies in [20, 32, 34] proposed various approaches to reduce the overheads of INT, they did not address how to mitigate the overheads when INT is used simultaneously with other techniques, which can also insert new fields in packets.

Generally speaking, SR utilizes the idea of source routing that let a source specify how its packets will be routed through a network [35]. The rising of SDN and PDP has promoted the developments and deployments of SR [43]. The SR implementations based on OpenFlow have been discussed in [44, 45], while SR has also been realized with POF [9, 46]. However, the aforementioned studies only considered SR, but did not try to merge its usage with that of INT. In [47, 48], the authors proposed to combine SR and INT to realize path-controllable network monitoring. Nevertheless, they simply included the header fields for both SR and INT in packets, and did not pay any attention to the accumulated overheads. Note that, SR and INT can benefit each other mutually and make network monitoring more effective. This is because SR can manage the configuration of INT data collection, while INT can assist SR to optimize its setting according to network status. This actually motivates us to study how to combine them seamlessly with the minimized overheads, for realizing highly-efficient and adaptive network monitoring.

We started the project on software-based POF switches in [23] by leveraging the software architecture in [38], and later on used the data plane development kit (DPDK) [49] to accelerate the packet processing and achieved a data-rate of 10 Gbps for 512-byte packets [50]. Next, in [34], we considered the OVS platform [36], added the support of POF in OVS to obtain a software-based POF switch (*i.e.*, OVS-POF), and implemented Sel-INT in OVS-POF. The experimental results in [34] indicated that OVS-POF can reach 10 Gbps line-rate when the packet size was set as 256 bytes. Hence, considering the performance of OVS-POF and the ecosystem of OVS for software switch development, we, in this work, further expand the functionalities of OVS-POF and realize SR-INT on it.

III. Design and Operation Procedure of SR-INT

In this section, we introduce the design of the packet format for SR-INT and its operation procedure.

A. Packet Format Design

By leveraging the protocol-independent nature of POF, we design the packet format for SR-INT, which inserts an SR-INT header in between the Ethernet and IP headers. Fig. 1 gives an illustrative example on how the fields related to SR and INT are organized in an SR-INT header. Specifically, at the last hop of each path segment, SR-INT replaces a *Segment* field

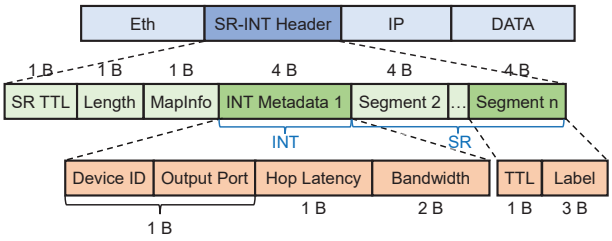


Fig. 1. Our design of the packet format for SR-INT.

with an *INT Metadata* field. Hence, the *Segment* fields in Fig. 1 start from *Segment* 2, because *Segment* 1 has already been replaced with *INT Metadata* 1. Note that, the insertion of the SR-INT header will not affect normal checksum operations on a packet for the following reasons. First of all, as the SR-INT header is placed before an IP header, its insertion will not change the content covered by the checksum fields in IP and TCP/UDP headers. Secondly, for the frame check sequence (FCS) field in Ethernet frame, it should be recalculated and updated after an SR-INT header having been inserted. This is accomplished automatically by the standard Ethernet frame processing procedure implemented in network interface cards (NICs), and thus the FCS field in Ethernet frame will be maintained correctly even with the SR-INT header.

To distinguish a packet with the SR-INT header from regular ones, we will let a POF switch modify its *EtherType* field to $0x0808$ after inserting the SR-INT header. The descriptions of the fields in the SR-INT header are as follows.

- *SR TTL*: This one-byte field records the number of *Segment* fields in the SR-INT header. When a packet has finished the current path segment, we will replace the corresponding *Segment* field with an *INT Metadata* field, and decrease *SR TTL* by 1. If *SR TTL* equals 0, the packet reaches its destination switch. Hence, the switch will first duplicate the packet to send to the data analyzer for INT collection and data analytics, then remove the SR-INT header in it, and forward the packet to its end host.
- *Length*: This is a one-byte field, and it is used to indicate the number of *INT Metadata* fields in the SR-INT header. Note that, as SR-INT time-multiplexes the header fields in each packet for INT and SR, it will replace a *Segment* field with an *INT Metadata* field, when a packet is about to exit the path segment that is represented by the *Segment* field. Hence, each POF switch needs an indicator to locate the first *Segment* field in an SR-INT header, which is exactly the *Length* field. Specifically, we initialize *Length* as 0 at the source of an SR-INT packet, and increase its value by 1 every time when the packet finishes a segment on its routing path. Then, the offset of the first *Segment* field in a packet can be calculated as $(17 + 4 \cdot \textit{Length})$ in bytes, where 14 bytes are for the Ethernet header and 3 bytes are for the *SR TTL*, *Length* and *MapInfo* fields.
- *MapInfo*: This field uses a one-byte bitmap to indicate the types of INT data to collect at the last switch of each segment, *i.e.*, an operator has the flexibility to customize its network monitoring scheme with this field. Specifically, we use each of the four lowest bits in *MapInfo*

to tell whether or not INT should collect the data about *Device ID*, *Output Port*, *Hop Latency*, and *Bandwidth*, respectively, while the remaining bits are unused.

- *INT Metadata*: This field uses 4 bytes to include the INT data about *Device ID*, *Output Port*, *Hop Latency*, and *Bandwidth*. Here, *Device ID* tells the ID of a switch, and *Output Port* stores the output port of the packet on the switch. Each of these two subfields occupies 4 bits, and they share a byte in *INT Metadata*. *Hop Latency* uses one byte to record the processing time (in μs) of the packet in the switch, and *Bandwidth* is a two-byte subfield that stores the bandwidth usage (in Mbps) of the packet's output port. Note that, an SR-INT header can include multiple *INT Metadata* fields, each of which denotes the status of the last switch of a segment.
- *Segment*: This field has the same length as that of *INT Metadata*, and represents the information of a segment. It consists of two subfields, *i.e.*, *TTL* and *Label*. *TTL* uses one byte to identify the position of the packet in the current segment. Specifically, we initialize *TTL* as the hop-count of a segment, and reduce it by one after each hop. When *TTL* becomes 1, the packet reaches the last switch of a segment, and the switch will replace the corresponding *Segment* field with an *INT Metadata* field. On the other hand, *Label* represents a segment on the packet's routing path (*i.e.*, each switch on the segment matches to the *Label* to get the output port for the packet).

Note that, we set the length of an *INT Metadata* field according to the specifications of INT in [26], and as the *Segment* and *INT Metadata* fields should be exchangeable in our SR-INT, they have to use the same length (4 bytes). Because the SR-INT header increases the length of a packet, SR-INT does have scalability issues. However, the same issues also exist in the traditional schemes of INT and SR, while SR-INT actually addresses them better by integrating INT and SR with less overheads. Meanwhile, for a flow, if the original packets without SR-INT headers are relatively long, we can always divide its path into long segments to reduce the number of *INT Metadata*/*Segment* fields in each SR-INT header.

With SR-INT, the control plane first partitions the routing path of a flow into several segments and generates a *MapInfo* field according to the monitoring requirement of the flow, then encodes *Label* fields to represent the segments, and finally installs the corresponding flow tables in all the switches on the flow's routing path. Hence, SR-INT is realized in the data plane for monitoring the flow in realtime. Here, SR and INT actually benefit each other mutually to make the network monitoring more effective and adaptive. Specifically, the control plane can leverage SR to manage the configuration of INT data collection adaptively (*e.g.*, monitoring the most important locations and most relevant INT data for the flow), and in the meantime, the status data collected by INT can in turn assist the control plane to optimize the settings of SR for transmitting the flow in a dynamic network environment.

B. Operation Procedure

The operation procedure in a POF switch to realize SR-INT is shown in *Algorithm* 1. After receiving a packet, the switch checks to see whether there are flow rules installed for it. If no, the switch will send a *PacketIn* message to the POF controller to report the packet, and the controller determines the flow rules for the packet and sets up the corresponding flow tables for it in the related switches by sending *FlowMod* messages to them (*Lines* 1-5). Then, if the packet matches to the flow rules for SR-INT, we first check its *EtherType* field to see whether it equals $0x0808$. If yes, an SR-INT header has already been inserted in the packet and we can proceed to the subsequent procedure. Otherwise, this switch is the source of the packet, and we need to modify its *EtherType* to $0x0808$ and insert an SR-INT header in it (*Lines* 7-10).

Algorithm 1: SR-INT Procedure in POF Switch

```
1  receive a packet Pkt;
2  if there is no flow rule for Pkt then
3      send PacketIn message to controller to report Pkt;
4      set up flow tables for Pkt based on the FlowMod
       message from controller;
5  end
6  if Pkt matches to SR-INT flow rules then
7      if Pkt.EtherType ≠ 0x0808 then
8          SetField(Pkt.EtherType, 0x0808);
9          insert an SR-INT header in Pkt;
10     end
11     locate the first Segment field in Pkt with Length;
12     determine the output port of Pkt based on Label
       in the Segment field;
13     reduce TTL in the Segment field by 1;
14     if TTL equals 1 then
15         collect INT data according to MapInfo and
           encode the data as an INT Metadata field;
16         replace the first Segment field with the INT
           Metadata field;
17         reduce SR TTL by 1 and increase Length by 1;
18     end
19     if SR TTL equals 0 then
20         duplicate Pkt to send to the INT collection
           and data analytics module;
21         remove the SR-INT header in Pkt;
22     end
23     forward Pkt to its output port;
24 end
```

Lines 11-12 leverage the *Length* field in the packet's SR-INT header to locate the first *Segment* field in it, and then match to the *Label* in the *Segment* field to determine the output port of the packet. This is because according to the principle of SR, the first *Segment* field always represents the forwarding scheme of the packet on the current segment. Next, we decrease the *TTL* in the *Segment* field by 1 to denote that one more hop of the current segment has been experienced (*Line* 13). If the *TTL* reaches 1 (*i.e.*, the current segment is finished), we use *Lines* 14-18 to collect INT data according to the *MapInfo* field, encode the data as an *INT Metadata* field to replace the first *Segment* field, and update the values of the *SR*

TTL and *Length* fields accordingly. Then, we check *SR TTL* in *Line* 19. If it equals 0, the packet reaches its destination switch, and thus we invoke the operations for INT data collection and removing the SR-INT header (*Lines* 20-21). Finally, in *Line* 23, we forward the packet to its desired output port.

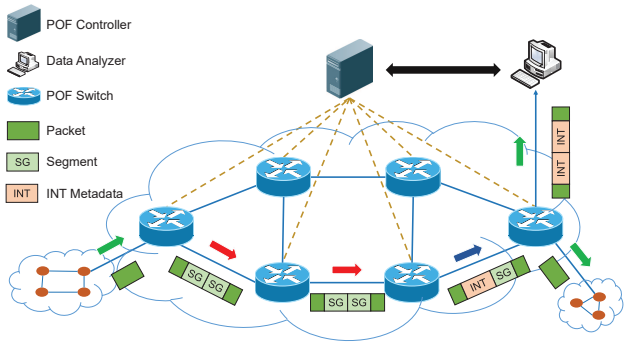


Fig. 2. Architecture of our SR-INT system based on POF.

Fig. 2 shows the system architecture of our proposal, and provides an illustrative example on the operation procedure defined in *Algorithm* 1. Note that, with *Algorithm* 1, each packet only experiences one *AddField* operation and one *DelField* operation in its source and destination switches for inserting and removing its SR-INT header, respectively, while the remaining SR-INT tasks can all be realized with *SetField* and *ModifyField* operations. This will reduce operation complexity in the switches (especially for the software-based ones), because *SetField* and *ModifyField* normally involve fewer memory operations and thus can be processed much faster than *AddField* and *DelField* in switches. We will verify this with experimental results in Section V. Moreover, *Algorithm* 1 makes the length of each SR-INT packet stay constant along its routing path in the POF network. This will not only reduce the bandwidth overheads used for supporting SR and INT simultaneously, but also avoid the hassle of tracking the throughput of each flow along its path (*e.g.*, for traffic engineering or congestion avoidance).

C. Analysis on Feasibility and Adaptability

We hope to point out that even though our SR-INT integrates SR and INT to reduce the overall overheads, its basic operations for SR and INT are the same as those in their traditional schemes, respectively. Therefore, SR-INT inherits the feasibility and adaptability of the traditional SR and INT schemes. For instance, as SR can adapt to different network scenarios well (*e.g.*, the network topology changes, the path length increases, and the segment size and its number varies) in principle [43] and its feasibility in various topologies has been verified in [51–54], SR-INT should have the similar feasibility and adaptability. This is because same as SR, SR-INT encodes each path segment as a label in the *Segment* field, and thus it can always encode a suitable number of *INT Metadata*/*Segment* fields in each SR-INT header by letting the control plane divide each path into segments properly according to the actual network scenario.

IV. System Implementation

In this section, we will explain how to implement the proposed SR-INT system based on POF.

A. System Architecture

Fig. 3 shows the architecture of the system to implement our proposed SR-INT, which involves three major network elements, *i.e.*, the POF controller based on ONOS, the POF switch based on OVS (OVS-POF), and the home-made data analyzer. The POF controller takes care of all the control plane operations for SR-INT. We modify the *Provider&Protocol* module in ONOS to extend the south-bound protocol stack there and add in the support of POF [55]. Therefore, POF-based control messages (*e.g.*, *FlowMod* and *TableMod*), which encode the tuples of <*offset*, *length*> and POF-FIS, can be conveyed between the control and data planes.

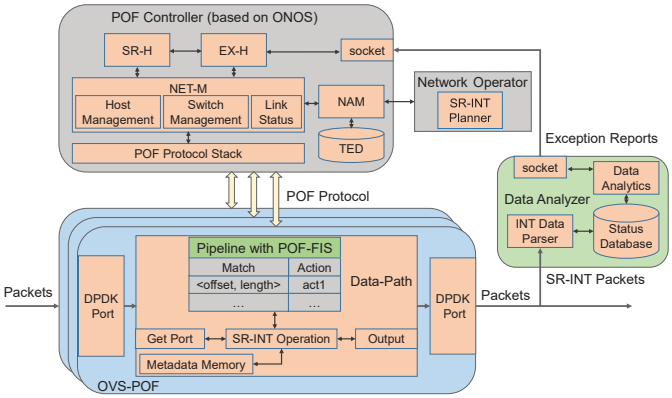


Fig. 3. Architecture of SR-INT system.

The network management module (NET-M) controls the network elements in the data plane and maintains their status. Specifically, it leverages the host management, switch management and link status submodules to manage the hosts, switches and links, respectively. The network abstraction module (NAM) takes the information about network status from the NET-M to obtain a global view about the data plane and store it in the traffic engineering database (TED). The SR handler (SR-H) processes the tasks related to SR (*e.g.*, path computation, creation of segments, and label assignment), while the exception handler (EX-H) is in charge of recovering the network from exceptions (*e.g.*, invoking path switching to bypass congested link(s)). The socket interface is implemented to receive the reports on exceptions from data analyzers.

As for the actual SR-INT scheme of each flow (*i.e.*, how to divide the flow's routing path into segments, and how to collect INT data for the segments), the POF controller determines it according to the policy provided by the network operator (as shown in Fig. 3). Specifically, to get the policy, the network operator leverages an SR-INT planner, which runs an optimization algorithm to divide each flow's path into segments properly and assign a suitable INT data collection scheme to each segment according to the actual network scenario. Note that, as the focus of this paper is the design and

implementation of the SR-INT technique, we will consider the SR-INT planning algorithm in our future work.

Both the OVS-POF and data analyzer are data plane elements. The OVS-POF leverages DPDK [49] to accelerate packet processing, and it uses pipelines built with flow tables that are based on POF-FIS to realize the data path for SR-INT. Meanwhile, the SR-INT operation matches the *Label* in the first *Segment* field in the SR-INT header to determine the output port of each SR-INT packet, and accesses the metadata memory to obtain the required INT data. We will discuss the detailed implementations in OVS-POF in the next subsection. When an SR-INT packet is about the leave the POF network, the egress switch will mirror it to a data analyzer.

Note that, we assume that there can be multiple distributed data analyzers in the network for INT collection and data analytics. The data analyzer is home-made, which is obtained by extending the one in [34]. Specifically, the improvements are as follows. Firstly, we upgrade the INT data parser to adapt to the packet format defined in Section III-A. Secondly, we re-implement the status database based on an open-source time series database platform (*i.e.*, InfluxData [56]), to ensure realtime accessibility and high-throughput data indexing and storage. Finally, we design and implement a data analytics module in it for network monitoring, which keeps analyzing the INT data in the status database during operation. If it detects a network exception, it will generate an exception report to send to the controller through the socket interface.

B. Implementation of SR-INT in OVS-POF

We extend the OVS-POF in [34] to get a high-performance software-based POF switch that supports SR-INT. Specifically, we first leverage the flexibility of POF-FIS to expand the generic *SetField* and *ModifyField* actions for SR-INT-related actions as follows, and then design the flow table in Fig. 4 to ensure that SR-INT can be executed on OVS-POF effectively.

- *Modify_srint_Field* <*offset, length*>: We design this action to modify the *SR TTL* and *Length* fields and *TTL* subfields in an SR-INT header at each hop. As shown in Fig. 4, *Modify_srint_Field* first extracts the *SR TTL* field from a packet and checks its value. If the value equals 0, it means that this is the last hop of the packet, and thus its SR-INT header needs to be deleted with a *DelField* action. Otherwise, the action extracts the *TTL* subfield in the first *Segment* field, with the tuple $<17 + 4 \cdot Length, 1>$. Then, if the *TTL* subfield equals 1, we know that this is the last hop in the current segment. Hence, we will first use *Modify_srint_Field* to update the *SR TTL* and *Length* fields and then call *Set_srint_Field* to replace the first *Segment* field with an *INT Metadata* field. On the other hand, If the *TTL* subfield does not equal 1, we only use *Modify_srint_Field* to decrease its value by 1.
- *Set_srint_Field* <*offset, length*>: This action is designed to replace a *Segment* field with an *INT Metadata* field and accomplish SR-INT on a packet. It first extracts the *TTL* subfield in the first *Segment* field. Then, if the *TTL* subfield equals 1, *Set_srint_Field* replaces the first *Segment* field with an *INT Metadata* field. Specifically, the action extracts the *MapInfo* field from the SR-INT header, composes an *INT Metadata* accordingly, and writes the *INT Metadata* to the location of the first *Segment*.

Note that, other than those mentioned above, we use the generic *AddField* and *DelField* actions in POF-FIS to insert and delete an SR-INT header in one packet, respectively, for realizing the procedure in *Algorithm* 1. Specifically, the SR-INT header can be inserted by calling *AddField* <*offset, length, value*>, where *offset* points to its start location (*i.e.*, right after the Ethernet header), *length* denotes its length, and *value* represents its content with the format in Fig. 1, while calling *DelField* <*offset, length*> removes the SR-INT header.

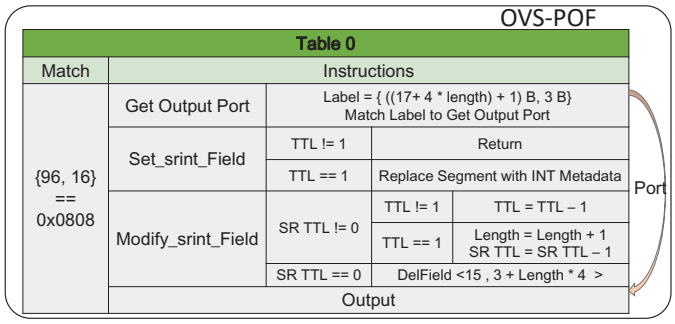


Fig. 4. Flow table on OVS-POF for SR-INT.

C. Adaptive SR-INT

In addition to extending OVS-POF to support SR-INT, we also implement the whole network system in Fig. 3. The network system makes SR and INT benefit each other mutually to achieve highly-efficient and adaptive network monitoring.

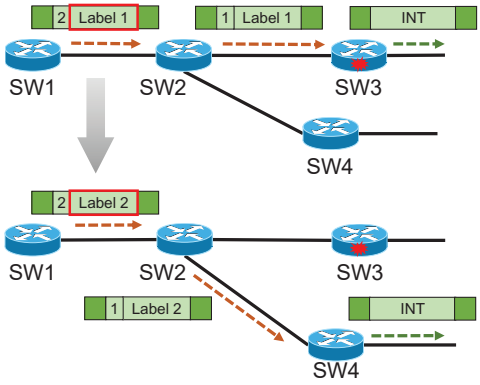


Fig. 5. Fast rerouting based on network status with SR-INT.

1) Fast Rerouting based on Network Status: As shown in Fig. 3, the INT data regarding how a packet is processed in the POF-enabled network is carried out by the packet itself and gets parsed and analyzed by the data analyzer in realtime. If the data analyzer finds any network exceptions, it will report them to the POF controller, which will reconfigure related data plane elements to restore the network from the exceptions. This actually makes the detection of and response to anomalies, especially the soft failures [19], much more timely than that with the conventional polling-based approaches. Meanwhile, with SR, the POF controller can realize fast

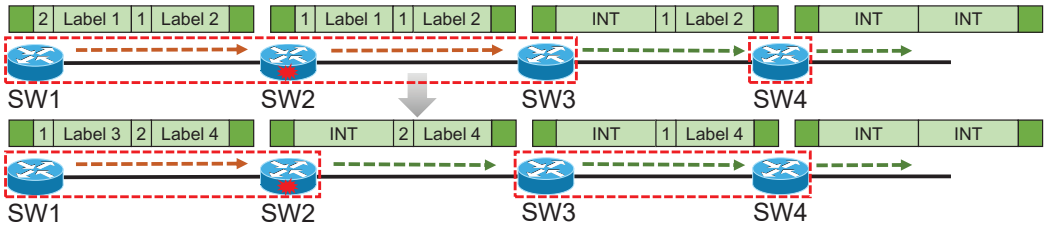


Fig. 6. Self-adaptive network monitoring with SR-INT.

rerouting by only updating the flow tables in the source switch to modify the *Segment* field(s) in packets' SR-INT headers.

For example, Fig. 5 provides an intuitive example of the fast rerouting achieved by SR-INT. Here, the original first path segment of the flow is *SW1→SW2→SW3*, which is represented by the *Label* 1 in the first *Segment* field in packets' SR-INT headers. Hence, after *SW3*, the first *Segment* field is replaced by an *INT Metadata* field, which records how a packet gets processed in *SW3*. Then, at a moment, an exception starts to happen on *SW3* and makes its packet processing latency change abnormally (*e.g.*, congestion). This phenomenon will be quickly detected by our SR-INT-based network monitoring, and then the POF controller invokes a fast rerouting by updating the related flow tables in *SW1* to change the label in the first *Segment* field to *Label* 2. Therefore, the first path segment gets updated to *SW1→SW2→SW4* instantly to bypass the exception, and closed-loop network monitoring and failure recovery can be realized efficiently.

2) Self-adaptive Network Monitoring: The example in Fig. 5 might raise the concern that our SR-INT-based network monitoring only has limited coverage, since the INT data collection is only conducted on the last switch of each path segment. We hope to point out that this is actually not an issue, because the POF controller can leverage SR-INT to change the configuration of INT data collection adaptively. More specifically, the controller can check the reports from a data analyzer and find the key switches and most relevant INT data to monitor for each flow.

Fig. 6 explains how to realize self-adaptive network monitoring with SR-INT. Initially, the routing path of a flow is partitioned into two segments, *i.e.*, *SW1→SW2→SW3* and *SW4*, and thus SR-INT collects INT data on *SW3* and *SW4*. However, during operation, *SW2* has network congestion, which will cause slight packet losses on the flow. The exception can be quickly detected by our SR-INT system, because the flow's throughput measured on *SW3* is less than the pre-known value, while the INT data about *SW3* indicates that it operates normally and thus is not where the exception happens. Hence, the POF controller decides to change the SR scheme of the flow to *SW1→SW2* and *SW3→SW4*, and to monitor *SW2* instead to find the root-cause of the anomaly. Again, this can be done by letting the POF controller update the related flow tables in the source switch (*i.e.*, *SW1*) to modify the *Segment* fields in packets' SR-INT headers. Therefore, self-adaptive network monitoring can be realized to check the status of the switches along a flow's routing path selectively as well as reduce the bandwidth overheads of SR-INT effectively.

V. Experimental Demonstrations

In this section, we discuss the experiments to demonstrate and evaluate our proposed SR-INT system.

A. Feature Validation

We first build a network testbed as shown in Fig. 7 to verify the functionalities of our proposal. The experimental setup consists of six stand-alone POF switches, a POF controller, a data-analyzer, and two hosts. Each of the POF switches is based on OVS-POF, and runs on a high-performance Linux server with 10 GbE linecards. The POF controller is based on the ONOS that contains our extensions to support POF in its south-bound protocol stack. Our experiments send traffic from *Host1* to *Host2*, and apply SR-INT to the flow.

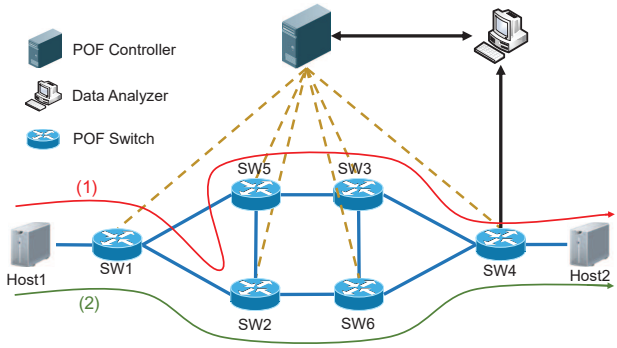


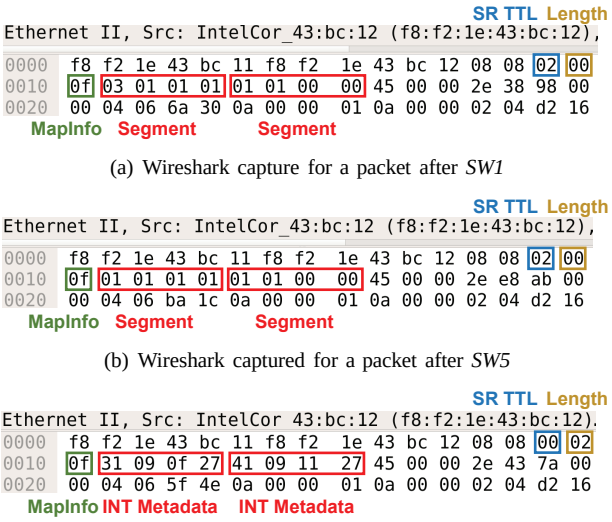
Fig. 7. Experimental setup for demonstrating SR-INT.

1) Verification of Basic SR-INT Functionalities: For the flow from *Host1* to *Host2*, we assign its routing path as *SW1→SW2→SW5→SW3→SW4*, and divide the path into two segments, *i.e.*, *SW1→SW2→SW5→SW3* and *SW4*. Hence, the POF controller installs a flow table in *SW1* to insert an SR-INT header in each packet of the flow, and lets each of *SW3* and *SW4* replace the first *Segment* field in an SR-INT header with an *INT Metadata* field about its own status.

Fig. 8 shows the Wireshark captures of the packets of the flow at different locations. After being processed by *SW1*, the packet in Fig. 8(a) shows that an SR-INT header has been inserted in it correctly. In the SR-INT header, the *SR TTL* equals 2, which means that there are two *Segment* fields corresponding to the two aforementioned segments, respectively, the *Length* is 0, which suggests that there is no *INT Metadata* field in the SR-INT header and thus the first *Segment* field is right after the *MapInfo* field, and the *MapInfo* field equals $0x0f$, which means that INT should collect the data about

Device ID, *Output Port*, *Hop Latency*, and *Bandwidth* (*i.e.*, all the 4 bits in the bitmap are on). Next, the *TTL* subfields in the two *Segment* fields are 3 and 1, *i.e.*, the two path segments still include 3 and 1 switches, respectively.

Then, the Wireshark capture in Fig. 8(b) provides the packet after *SW5*. We can see that the packet is almost the same as that in Fig. 8(a), except that the *TTL* subfield in the first *Segment* field gets decreased to 1. This is because there is still one more hop, which is *SW3*, in the current segment. Finally, Fig. 8(c) shows the Wireshark capture for the packet after *SW4*. Here, as *SW4* is the last switch of the second path segment, both of the *Segment* fields in the original SR-INT header have been replaced with *INT Metadata* fields, which corresponds to the status on *SW3* and *SW4*, respectively. By checking the details in the two *INT Metadata* fields, we can see that the *Device IDs* of the two switches are 3 and 4, respectively, both of the switches output the packet on *Port* 1, their *Hop latencies* are all 9 μs, while the used *Bandwidths* on their corresponding output ports are different because there are background traffic in addition to the considered flow. Hence, the Wireshark captures in Fig. 8 confirm that the basic functionalities of SR-INT have been implemented correctly.



(a) Wireshark capture for a packet after *SW1*

(b) Wireshark captured for a packet after *SW5*

(c) Wireshark captured for a packet after *SW4*

Fig. 8. Experimental results for verifying basic SR-INT function.

2) Fast Rerouting based on Network Status: Then, we continue the experiment to measure the *Hop Latency* on *SW3* for 9 seconds, and after the eighth second, we increase the background traffic through *SW3* to induce slight congestion on it. Fig. 9 shows the *Hop Latency* on *SW3*, which is obtained by the data analyzer. Note that, with SR-INT, each packet is transmitted to a data analyzer by the egress switch (*i.e.*, the last switch on its routing path). For example, for the flow *Host1* to *Host2* in Fig. 7, its packets are duplicated and sent to the data analyzer by *SW4*. The data analyzer runs a process to collect all the packets from the egress switch, and thus it can extract SR-INT headers from the packets, parse the telemetry data in them, and process the data for real-time network monitoring.

It can be seen that the *Hop Latency* stays below 10 μs for 8 seconds, and then suddenly increases to $\sim$13 μs. The data analyzer treats the sudden increase as an exception and sends a report to the POF controller. Specifically, as shown in Fig. 10(a), the data analyzer sets up a TCP connection to report the exception. Then, the controller decides to reroute the flow to go through *SW1*$\rightarrow$*SW2*$\rightarrow$*SW6*$\rightarrow$*SW4* (*i.e.*, the path marked with the green line in Fig. 7), and still divide the path into two segments (*i.e.*, *SW1*$\rightarrow$*SW2*$\rightarrow$*SW6* and *SW4*). Fig. 10(b) shows the Wireshark captures of the packets, which are collected after *SW1*, before and after the rerouting. We can see that after the rerouting, the content of the first *Segment* field is changed while that of the second one stays the same. Specifically, the *TTL* subfield in the first *Segment* field becomes 2 to denote that the first segment still contains *SW2* and *SW6* after *SW1*, and the *Label* subfield is changed to represent the new segment.

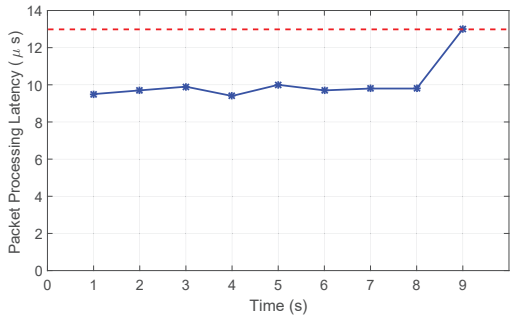
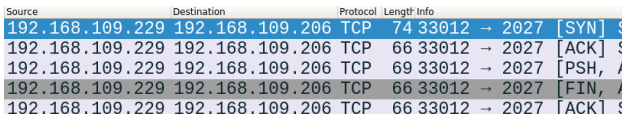
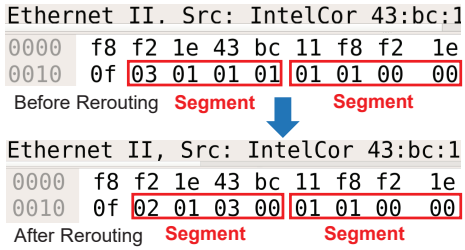


Fig. 9. *Hop Latency* of packets on *SW3*.



(a) Packets captured on POF controller



(b) Wireshark captures for packets after *SW1*

Fig. 10. Results on fast rerouting based on network status.

3) Self-adaptive Network Monitoring: To demonstrate that our SR-INT system can achieve self-adaptive network monitoring, we still let the flow go through *SW1*$\rightarrow$*SW2*$\rightarrow$*SW5*$\rightarrow$*SW3*$\rightarrow$*SW4*, but use the POF controller to change its SR-INT scheme dynamically. Specifically, the POF controller will re-partition the routing path as *SW1*$\rightarrow$*SW2*$\rightarrow$*SW5* and *SW3*$\rightarrow$*SW4* and instruct the switches to only collect the INT data about *Device ID*, *Output Port*, and *Hop Latency*. Fig. 11 shows the Wireshark capture of a packet after SW4, when the new SR-INT scheme mentioned above has been applied. Specifically, we can see that the *MapInfo* field gets changed to $0x03$, and the *Bandwidth* subfields are always 0 since they are excluded from the INT data collection.

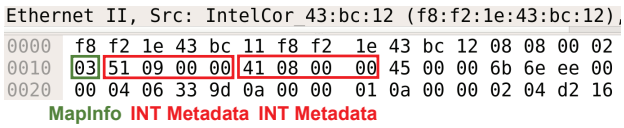


Fig. 11. Wireshark capture for a packet after *SW4* with new SR-INT scheme.

Note that, in addition to readjusting the switches and INT data types to monitor for a flow, self-adaptive network monitoring can also be introduced to balance the tradeoff between the bandwidth overheads and accuracy of network monitoring. Fig. 12 compares the bandwidth overheads of SR-INT when different packet sizes and number of monitoring points are considered. As expected, the bandwidth overheads decrease with packet size and increase with the number of monitoring points (*i.e.*, how many *Segment* fields are included in an SR-INT header). In the worst case, the experiments consider 4 monitoring points on a routing path and use 64-byte packets, and the bandwidth overheads of SR-INT in Fig. 12 is 29.69%.

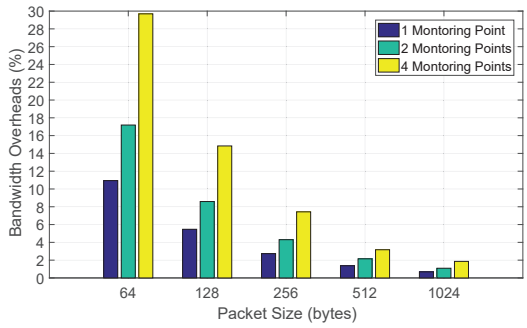


Fig. 12. Bandwidth overheads of SR-INT.

B. Packet Processing Performance of OVS-POF with SR-INT

As we have explained before, our design of SR-INT makes sure that each packet only experiences one *AddField* and one *DelField* throughout its forwarding in the POF network, while the remaining SR-INT operations are all realized with *SetField* and *ModifyField* operations. Hence, its implementation in OVS-POF has reduced operation complexity, because *SetField* and *ModifyField* normally involve fewer memory operations and thus can be processed much faster than *AddField* and *DelField* in software-based switches. To verify this effect, we conduct experiments to compare the throughput of OVS-POF for the cases when SR-INT is used and SR and INT are handled independently (SR+INT). We still let the flow go through *SW1*→*SW2*→*SW5*→*SW3*→*SW4* and divide the path into 2 segments, *i.e.*, *SW1*→*SW2*→*SW5*→*SW3* and *SW4*. Then, we pump in 10 Gbps traffic with different packet sizes at *SW1*, and measure the output throughput at *SW4*.

Fig. 13 shows the results on end-to-end throughput. We observe that for both SR-INT and SR+INT, their throughputs increase with the packet size and reach the linerate (10 Gbps) when $1,024$-byte packets are used. This is because when the packet size becomes smaller, the software-based switches on the routing path need to process more packets per second. Meanwhile, the end-to-end throughput of SR-INT is always higher than that of SR+INT when the packet size ranges within $[64, 512]$ bytes. This verifies the advantage of our proposal.

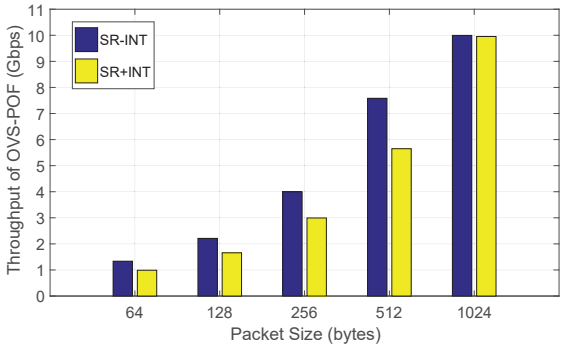


Fig. 13. Comparisons on end-to-end throughput of SR-INT and SR+INT.

Moreover, we measure the processing latency per packet on *SW4* for SR-INT and SR+INT, and the results are plotted in Fig. 14. It can be seen that the packet processing latency of SR-INT is ~ 9.5 μs, while that of SR+INT is ~ 16 μs. As *SW4* is the destination switch, SR-INT applies one *SetField*, one *ModifyField* and one *DelField* on each packet, while for each packet, SR+INT applies one *AddField* to insert an INT-related field in it, and two *DelField* operations (*i.e.*, one is for removing the field for SR, and the other is for converting the packet back to a normal one that does not include any fields related to INT or SR). Therefore, the operations for SR+INT are more complex, which is the reason why its packet processing latency in Fig. 14 is much longer.

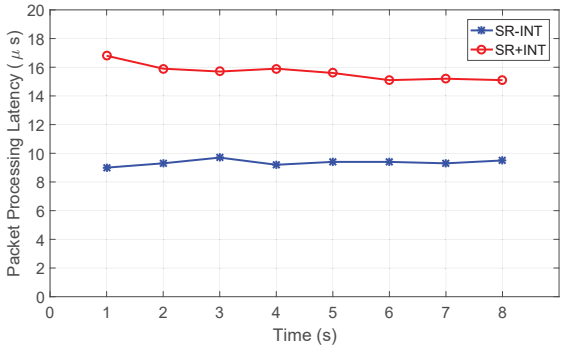


Fig. 14. Comparisons of packet processing latency on *SW4*.

VI. Conclusion

In this paper, we proposed SR-INT, which is a network monitoring system that can combine INT and SR seamlessly to mitigate their overheads. By leveraging POF, we designed the packet format of SR-INT and laid out its packet processing procedure. Moreover, to ensure its adaptivity, we made implementations in both the control and data planes such that the configuration of SR-INT can be adjusted dynamically to adapt to the requirements of network monitoring. The proposed SR-INT was implemented and experimentally demonstrated in a POF-based SDN environment. Our experimental results

showed that SR-INT reduces not only the overheads of using SR and INT simultaneously but also the operation complexity in software-based POF switches, and it achieves highly-efficient and adaptive network monitoring for fault diagnosis and can recover the network from soft-failures quickly.

ACKNOWLEDGMENTS

This work was supported in part by the NSFC projects 61871357 and 61771445, Zhejiang Lab Research Fund 2019LE0AB01, CAS Key Project (QYZDY-SSW-JSC003), SPR Program of CAS (XDC02070300), and Fundamental Funds for Central Universities (WK3500000006).

REFERENCES

[1] Cisco Annual Internet Report (2018-2023). [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[2] 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.

[3] Y. Tian, R. Dey, Y. Liu, and K. Ross, "Topology mapping and geolocating for China's Internet," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, pp. 1908–1917, Sept. 2013.

[4] 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.

[5] 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.

[6] 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.

[7] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–98, Apr. 2014.

[8] Z. Zhu *et al.*, "Demonstration of cooperative resource allocation in an OpenFlow-controlled multidomain and multinational SD-EON testbed," *J. Lightw. Technol.*, vol. 33, pp. 1508–1514, Apr. 2015.

[9] S. Li *et al.*, "Improving SDN scalability with protocol-oblivious source routing: A system-level study," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, pp. 275–288, Mar. 2018.

[10] X. Wang *et al.*, "PNPL: Simplifying programming for protocol-oblivious SDN networks," *Comput. Netw.*, vol. 147, pp. 64–80, Dec. 2018.

[11] W. Fang *et al.*, "Joint spectrum and IT resource allocation for efficient vNF service chaining in inter-datacenter elastic optical networks," *IEEE Commun. Lett.*, vol. 20, pp. 1539–1542, Aug. 2016.

[12] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks," *J. Lightw. Technol.*, vol. 34, pp. 3330–3341, Jul. 2016.

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

[14] Y. Wang, P. Lu, W. Lu, and Z. Zhu, "Cost-efficient virtual network function graph (vNFG) provisioning in multidomain elastic optical networks," *J. Lightw. Technol.*, vol. 35, pp. 2712–2723, Jul. 2017.

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

[16] H. Jiang, Y. Wang, L. Gong, and Z. Zhu, "Availability-aware survivable virtual network embedding (A-SVNE) in optical datacenter networks," *J. Opt. Commun. Netw.*, vol. 7, pp. 1160–1171, Dec. 2015.

[17] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding (LC-VNE) algorithms towards integrated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 3648–3661, Dec. 2016.

[18] Z. Zhu *et al.*, "Jitter and amplitude noise accumulations in cascaded all-optical regenerators," *J. Lightw. Technol.*, vol. 26, pp. 1640–1652, Jun. 2008.

[19] R. Govindan *et al.*, "Evolve or die: High-availability design principles drawn from Google's network infrastructure," in *Proc. of ACM SIGCOMM 2016*, pp. 58–72, Aug. 2016.

[20] B. Niu *et al.*, "Visualize your IP-over-optical network in realtime: A P4-based flexible multilayer in-band network telemetry (ML-INT) system," *IEEE Access*, vol. 7, pp. 82 413–82 423, 2019.

[21] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)," *RFC 1098*, May 1990. [Online]. Available: https://tools.ietf.org/html/rfc1157

[22] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

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

[24] Protocol Indepedent Forwarding. [Online]. Available: https://opennetworking.org/news-and-events/protocol-independent-forwarding/

[25] J. Yu *et al.*, "Forwarding programming in protocol-oblivious instruction set," in *Proc. of ICNP 2014*, pp. 577–582, Oct. 2014.

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

[27] 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.

[28] Z. Zhu, S. Li, and X. Chen, "Design QoS-aware multi-path provisioning strategies for efficient cloud-assisted SVC video streaming to heterogeneous clients," *IEEE Trans. Multimedia*, vol. 15, pp. 758–768, Jun. 2013.

[29] N. Xue *et al.*, "Demonstration of OpenFlow-controlled network orchestration for adaptive SVC video manycast," *IEEE Trans. Multimedia*, vol. 17, pp. 1617–1629, Sept. 2015.

[30] 100G in-band network telemetry with Netcope P4. [Online]. Available: https://www.netcope.com/Netcope/media/content/100G-In-band-Network-Telemetry-With-Netcope-P4.pdf

[31] C. Kim *et al.*, "In-band network telemetry via programmable dataplanes," in *Proc. of ACM SIGCOMM 2015*, pp. 1–2, Aug. 2015.

[32] Y. Kim, D. Suh, and S. Pack, "Selective in-band network telemetry for overhead reduction," in *Proc. of CloudNet 2018*, pp. 1–3, Oct. 2018.

[33] S. Tang, J. Kong, B. Niu, and Z. Zhu, "Programmable multilayer INT: An enabler for AI-assisted network automation," *IEEE Commun. Mag.*, vol. 58, pp. 26–32, Jan. 2020.

[34] S. Tang *et al.*, "Sel-INT: A runtime-programmable selective in-band network telemetry system," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, pp. 708–721, Jun. 2020.

[35] Segment Routing. [Online]. Available: https://www.segment-routing.net/.

[36] B. Pfaff *et al.*, "The design and implementation of Open vSwitch," in *Proc. of USENIX NSDI 2015*, pp. 117–130, May 2015.

[37] ONOS. [Online]. Available: https://onosproject.org/

[38] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proc. of ACM HotSDN 2013*, pp. 127–132, Aug. 2013.

[39] OpenFlow Switch Specifications. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

[40] K. Han *et al.*, "Application-driven end-to-end slicing: When wireless network virtualization orchestrates with NFV-based mobile edge computing," *IEEE Access*, vol. 6, pp. 26 567–26 577, 2018.

[41] J. Hyun, N. Tu, and J. Hong, "Towards knowledge-defined networking using in-band network telemetry," in *Proc. of NOMS 2018*, pp. 1–7, Apr. 2018.

[42] Barefoot deep insight. [Online]. Available: https://www.barefootnetworks.com/products/brief-deep-insight/.

[43] Z. Abdullah, I. Ahmad, and I. Hussain, "Segment routing in software defined networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, pp. 464–486, First Quarter 2019.

[44] A. Iyer, V. Mann, and N. Samineni, "SwitchReduce: Reducing switch state and controller involvement in OpenFlow networks," in *Proc. of NETWORKING 2013*, pp. 1–9, May 2013.

[45] A. Hari, T. Lakshman, and G. Wilfong, "Path switching: Reduced-state flow handling in SDN using path information," in *Proc. of ACM CoNEXT 2015*, pp. 1–7, Dec. 2015.

[46] S. Li *et al.*, "SR-PVX: A source routing based network virtualization hypervisor to enable POF-FIS programmability in vSDNs," *IEEE Access*, vol. 5, pp. 7659–7666, 2017.

[47] 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.

[48] Y. Lin *et al.*, "Netview: Towards on-demand network-wide telemetry in the data center," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2020.

[49] DPDK: Data Plane Development Kit. [Online]. Available: https://www.dpdk.org/
[50] Q. Sun, Y. Xue, S. Li, and Z. Zhu, "Design and demonstration of high-throughput protocol oblivious packet forwarding to support software-defined vehicular networks," *IEEE Access*, vol. 5, pp. 24 004–24 011, 2017.
[51] Z. Pan *et al.*, "Demonstration of variable-length packet contention resolution and packet forwarding in an optical-label switching router," *IEEE Photon. Technol. Lett.*, vol. 16, pp. 1772–1774, Jul. 2004.
[52] A. Liberato *et al.*, "RDNA: Residue-defined networking architecture enabling ultra-reliable low-latency datacenters," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, pp. 1473–1487, Dec. 2018.
[53] M. Shahbaz *et al.*, "Elmo: Source routed multicast for public clouds," in *Proc. of ACM SIGCOMM 2019*, pp. 458–471, Aug. 2019.
[54] C. Dominicini *et al.*, "KeySFC: Traffic steering using strict source routing for dynamic and efficient network orchestration," *Comput. Netw.*, vol. 167, pp. 1–18, Feb. 2020.
[55] H. Huang *et al.*, "Realizing highly-available, scalable and protocol-independent vSDN slicing with a distributed network hypervisor system," *IEEE Access*, vol. 6, pp. 13 513–13 522, 2018.
[56] InfluxDB. [Online]. Available: https://www.influxdata.com/.