SAFE-ME: Scalable and Flexible Policy Enforcement in Middlebox Networks

Journal:	*IEEE/ACM Transactions on Networking*
Manuscript ID	TNET-2021-00358
Manuscript Type:	Original Article
Date Submitted by the Author:	23-Aug-2021
Complete List of Authors:	Xu, Hongli; University of Science and Technology of China, School of Computer Science and Technology Xi, Peng; Anhui Normal University, School of Educational Science Zhao, Gongming; University of Science and Technology of China, School of Computer Science and Technology Liu, Jianchun; University of Science and Technology of China, school of computer science and technology Qian, Chen; University of California Santa Cruz, Department of Computer Science and Engineering Huang, Liusheng; University of Science and Technology of China, School of Computer Science and Technology
Keywords:	Middlebox Networks, Network Function Virtualization, Scalability, Default Path, Tag

SAFE-ME: Scalable and Flexible Policy Enforcement in Middlebox Networks

Hongli Xu, *Member, IEEE,* Peng Xi, Gongming Zhao, *Member, IEEE,*
Jianchun Liu, Chen Qian, *Member, IEEE,* Liusheng Huang, *Member, IEEE,*

Abstract—The past decades have seen a proliferation of middlebox deployment in various scenarios, including backbone networks and cloud networks. Since flows have to traverse specific service function chains (SFCs) for security and performance enhancement, it becomes much complex for SFC routing due to routing loops, traffic dynamics and scalability requirement. The existing SFC routing solutions may consume many resources (*e.g.*, TCAM) on the data plane and lead to massive overhead on the control plane, which decrease the scalability of middlebox networks. Due to SFC requirement and potential routing loops, solutions like traditional default paths (*e.g.*, using ECMP) that are widely used in non-middlebox networks will no longer be feasible. In this paper, we present and implement a scalable and flexible middlebox policy enforcement (SAFE-ME) system to minimize the TCAM usage and control overhead. To this end, we design the smart tag operations for construction of default SFC paths with less TCAM rules in the data plane, and present lightweight SFC routing update with less control overhead for dealing with traffic dynamics in the control plane. We implement our solution and evaluate its performance with experiments on both physical platform (Pica8) and Programming Protocol-independent Packet Processors (P4) based data plane, as well as large-scale simulations. Both experimental and simulation results show that SAFE-ME can greatly improve scalability (*e.g.*, TCAM cost, update delay, and control overhead) in middlebox networks, especially for large-scale clouds. For example, our system can reduce the control traffic overhead by about 85% while achieving almost the similar middlebox load, compared with state-of-the-art solutions.

Index Terms—*Middlebox Networks, Network Function Virtualization, Scalability, Default Path, Tag.*

1 Introduction

Network Functions (NFs) such as firewalls, deep packet inspection, load balancer, *etc.* are provided by specialized network devices or software implementation (*i.e.*, Network Function Virtualization, NFV) [2]. We collectively refer to these specialized/virtualized devices as middleboxes (MBs) for simplicity [2]. MBs have been widely deployed in various networking scenarios including cloud computing environments, campus networks, backbone networks and data centers [3]. Typically, network flows go through several NFs in a specific order to meet its processing requirements, also called Service Function Chaining (SFC) [2]. We call a network with middlebox deployment and fine-grained middlebox policies as a 'middlebox network'.

Recently, with the advantage of the logically centralized control, software defined networking (SDN) has become an emerging technology to cope with complex SFC routing [2] [4] [5]. Under the SDN framework, the switch often forwards data packets by matching the rules in the TCAM-based forwarding table to achieve fast rule lookup or wildcard rule matching. However, TCAMs are 400× more expensive and consume 100× more power per Mbit than the RAM-based storage on switches [6]. Considering the processing speed, price and power consumption of the switch, most of today's commodity switches are equipped with a limited number of TCAM-based entries. Thus, it is essential to save the number of TCAM-based entries used for routing [7] [8]. In fact, besides flow routing, it needs to install extra flow entries on switches to redirect traffic to MBs/NFs for processing [2] [4]. As a result, SFC routing requires more flow entries compared with traditional routing, which accordingly increases significant difficulty for both control plane scalability and data plane scalability.

- *Control Plane Scalability.* Under the SDN framework, the controller is responsible for managing the whole network, *e.g.*, monitoring the network status and determining/updating the SFC routing [9]. Thus, in a large-scale network, when encountering network failures or network performance degradation, the controller needs to update many flow entries for network reconfiguration, which may result in high control overhead [9] [10].
- *Data Plane Scalability.* Due to the limited size of the TCAM-based forwarding table, it is another challenge to accommodate a large number (*e.g.*, 10^6 in a moderate-sized data center

- *Some preliminary results of this paper were published in the Proceedings of IEEE ICNP 2019 [1].*
- *H. Xu, P. Xi, G. Zhao, J. Liu and L. Huang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu, China, 215123. P. Xi is also with School of Educational Science, Anhui Normal University, Wuhu, Anhui, China, 241000. E-mail: xuhongli@ustc.edu.cn, xipeng@mail.ahnu.edu.cn, gmzhao@ustc.edu.cn, lyl617@mail.ustc.edu.cn, lshuang@ustc.edu.cn.*
- *C. Qian is with the Department of Computer Science and Engineering, University of California Santa Cruz, Santa Cruz, CA, USA, 95064. E-mail:cqian12@ucsc.edu.*

[11]) of flows using only a limited size of TCAM-based forwarding entries, especially with SFC requirement. Some SDN switches (*e.g.*, Noviswitch [12]) adopt RAM-based flow tables for flow forwarding. However, RAM-based flow tables may significantly increase the lookup latency [7]. In addition, a large number of flow entries may increase the delay for switches to modify/match entries [8] [13].

Though the traditional solutions, *e.g.*, default paths [14], can achieve better system scalability and deal with a large number of flows in traditional networks, these solutions cannot be applied directly in MB networks for the following reasons. First, SFC routing will cause routing loops, which is the main difference from traditional networks (Section 2.1). However, default paths cannot deal with routing loops. Second, flows with the same egress switch (or destination) will be processed in the same way by default paths, but they may require different SFCs. How to setup default paths for different SFC requirements remains a challenging problem.

To solve complex SFC routing, several works have designed efficient solutions for MB networks [2] [4] [15] [16]. However, these solutions still face several critical disadvantages. First, these solutions often install rules for flows with the granularity of ingress-egress switch pairs. If a network contains several thousands of ingress/egress switches (*e.g.*, NTT Hong Kong Financial Data Center contains more than 7000 racks [17]), there are millions of ingress-egress switch pairs [17] [18]. Consequently, it requires millions of forwarding entries on a switch in the worst case. When encountering network failures or network performance degradation, these solutions will encounter a large response time for network reconfiguration due to the low rule installation speed, which will be validated through experimental testing in Section 6. Second, existing proactive-based solutions (*e.g.*, [2]) usually install rules for flows in advance based on traffic estimation. Thus, they cannot deal with bursty traffic well and significantly decrease the users' QoS [11]. Third, existing reactive-based solutions (*e.g.*, [4]) mainly rely on the controller to perform real-time routing calculation and rule installation for new arrival flows, which may lead to serious per-flow communication/computation overhead on the control plane [9].

To conquer the above challenges, we design the scalable and flexible middlebox policy enforcement system (SAFE-ME). The key idea behind SAFE-ME is to configure three types of tables in the switch's data plane, namely the SFC table, NF table, and Flow table. The SFC table maintains the SFC policy information and assigns tags to packets that match certain policies. The NF table provides the path information to the NFs by checking the packet tags. The Flow table is on a per-switch basis to forward packets to their destinations, similar to those in traditional routers/switches. We design the smart tag operations for construction of default SFC paths, and present lightweight SFC routing update for dealing with traffic dynamics. Both experimental and simulation results show that SAFE-ME reduces flow entries, control overhead, and update delay by >80%, compared with state-of-the-art solutions.

The rest of this paper is organized as follows. Section 2 analyzes the limitations of the previous SFC routing solutions and gives the motivation of our research. We present the overview and workflow of the SAFE-ME system in Section 3. Sections 4 and 5 describe the data plane design and the control plane design of SAFE-ME, respectively. We implement SAFE-ME on a small-scale testbed and large-scale simulations in Section 6. We conclude this paper in Section 7.

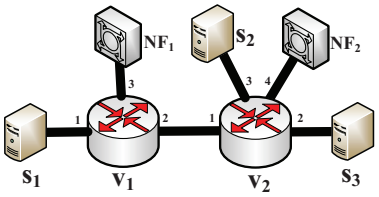

Fig. 1: Traffic from s_1 to s_3 has to traverse NF_1 and traffic from s_2 to s_3 has to traverse NF_1 and NF_2. Different requests with the same egress switch (or destination) will traverse different SFCs. However, the switch-based (or destination-based) default path solution cannot distinguish the traffic from s_1 or s_2.

2 Background and Motivation

In this section, we first introduce the traditional network forwarding mechanism based on the default path and prove its inapplicability for SFC routing in Section 2.1. Then we review the existing SDN-based SFC routing solutions and analyze their limitations in Section 2.2.

2.1 Inapplicability of Traditional Default Path Solutions

A natural strawman solution for flow routing with less forwarding entries is deploying default paths (*e.g.*, using switch-based or destination-based OSPF/ECMP methods [14] [19]). However, in middlebox networks, there may exist routing loops in the forwarding paths due to SFC requirements. In addition, flows with the same egress switch (or destination) may traverse different SFCs, which cannot be satisfied by default paths. Thus, traditional default paths cannot solve the SFC routing problem with fine-grained middlebox policies.

We give an example to illustrate the difference of flow routing between traditional networks and middlebox networks. As shown in Fig. 1, if we forward traffic from server s_1 to server s_3 in the traditional network (*i.e.*, without any SFC requirement), we can install one entry (*i.e.*, $dst = s_3, output = 2$) on each of switches v_1 and v_2, so that traffic will be forwarded through path $s_1 - v_1 - v_2 - s_3$.

However, in middlebox networks, we may require traffic from servers s_1 to s_3 to go through NF_1 for security purposes. The forwarding path is $s_1 - v_1 - NF_1 - v_1 - v_2 - s_3$. We observe that there exists a loop (*i.e.*, one packet will traverse v_1 twice). Thus, we cannot simply adopt the above destination-based routing method for such SFC routing. One may say that this problem can be solved by combining with the port information, *e.g.*, the ingress port of each flow. For example, for switch v_1, we can insert two destination-based entries: 1) $dst = s_3$ and $inport = 1, output = 3$; 2) $dst = s_3$ and $inport = 3, output = 2$. However, this solution does not work in a more complex scenario: the operator may specify all traffic from server s_1 to s_3 to go through NF_1 (*i.e.*, $s_1 - v_1 - NF_1 - v_1 - v_2 - s_3$) and traffic from server s_2 to s_3 through NF_1-NF_2 (*i.e.*, $s_2 - v_2 - v_1 - NF_1 - v_1 - v_2 - NF_2 - v_2 - s_3$). These two flows/requests with the same destination s_3

Schemes	Number of Rules	Control Overhead	Satisfy SFC Policy	Network Performance
Traditional Default Path (*e.g.*, [14] [19] [20] [21])	Few	Low	No	Low
Per-request Routing (*e.g.*, [2] [4] [15] [16])	Many	High	Yes	High
Our Scheme	**Few**	**Low**	**Yes**	**High**

TABLE 1: Comparison of the advantages and disadvantages of existing solutions.

will go through different service function chains. The destination-based routing solution cannot *distinguish* the traffic from s_1 or s_2. Consequently, we cannot determine the proper actions for traffic from v_1 to v_2. Prior work [2] shows that nearly 15% SFC routing paths contain loops. Hence tradition default path methods cannot address the SFC routing challenges.

2.2 Limitations of Prior SFC Routing Solutions

Although packet tags help to solve the routing loops, it may be flow-entry consuming if each 5-tuple flow is attached with a tag. Thus, many works have leveraged the per-request routing strategy to reduce the flow-entry consumption and achieve load balancing [2] [22] [23]. Specifically, a *request* is identified by three elements, ingress switch, egress switch and SFC. That is, all flows with the same ingress switch, egress switch and SFC requirement, will be aggregated into one request. For each newly-arrived request, the corresponding ingress switch reports the packet header information to the controller for requesting forwarding strategy. The controller then computes a proper routing path satisfying the service policy and replies the rule installment instructions to switches along the routing path. Although some 5-tuple flows are aggregated into a request, this solution still requires a large number of entries and leads to massive control overhead even in a moderate-size network. For example, in a practical data center network with 1,000 leaf switches, there may exist $O(1,000 \times 1,000)$ switch pairs. Even if there is only one MB in the network (or one SFC requirement per switch pair), it may require 1M entries on a switch in the worst case, which may violate todays switch capabilities or result in huge energy consumption and long update delay [7]. When multiple SFC requirements are posed for each switch pair, it becomes more serious. Meanwhile, since many rules will be installed and modified under per-request routing scheme, the communication/computation overhead on the control plane is too high, which will be validated in Section 6.

To reduce the TCAM table cost and control overhead, some research attempts to simplify the SFC processing in middlebox networks by constructing a **consolidated platform** [24] [25] [26] [27] [28] [29]. For example, CoMB [26] is a network function consolidation platform, where a flow/request can be processed by all required network functions on a single hardware platform, thus simplifying the SFC routing. Metron [24], MiddleClick [25], and OpenBox [27] also adopt the consolidation conception so as to merge similar packet processing elements into one. Slick [28] and SNF [29] combine the consolidated platform with NF placement and SFC rule decision making. In Fig. 1, they may integrate NF_1 and NF_2 into one mixed NF. All traffic will traverse the mixed NF and be processed automatically by corresponding functions. Though the consolidated platform simplifies the SFC processing, as described in the above paragraph, it still requires a large number of forwarding entries if without proper routing strategies. In fact, our proposed solution tries to optimize the route selection for middlebox networks, and can be combined with the consolidated platform, which will be discussed in Section 4.5.

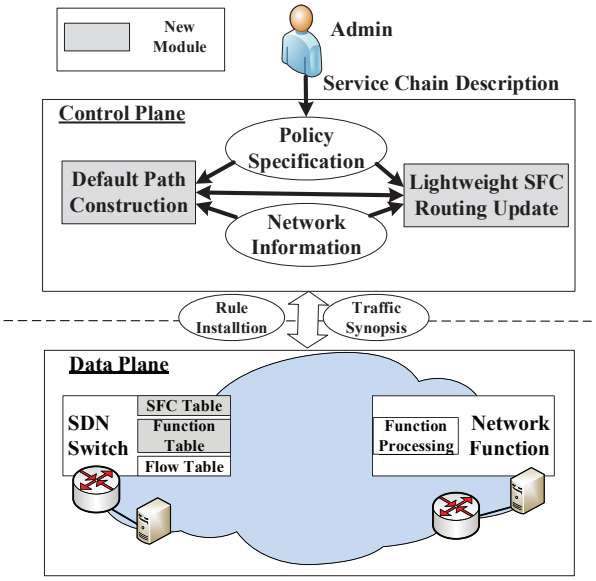

Fig. 2: System Overview. In the data plane, each switch maintains three tables: SFC table, NF table and Flow table. The control plane consists of two new modules, Default Path Construction and Lightweight SFC Routing Update, to implement routing strategies.

From Table 1, we observe that all existing methods can only address partial challenges of the SFC routing in middlebox networks. In other words, it seems that none of them can satisfy SFC requirements with better routing performance, lower control overhead and fewer flow rules consumption. Thus, in this paper, we design an efficient architecture for SFC routing (*i.e.*, SAFE-ME) so as to satisfy the above requirements.

3 System Design

To solve the above challenges in middlebox networks, we propose the scalable and flexible middlebox policy enforcement system (SAFE-ME). We present the overview of SAFE-ME in Section 3.1, describe the packet processing procedure in Section 3.2, and give an example to illustrate SAFE-ME in Section 3.3.

3.1 System Overview

As shown in Fig. 2, SAFE-ME consists of data plane and control plane designs. The proposed architecture addresses the challenges of scalable SFC routing in middlebox networks by embedding the SFC policy (as a tag) into the packet header. We first give an outline of the data plane and the control plane.

Data Plane of middlebox networks consists of SDN switches, NFs, servers and links. The SDN switches are responsible for forwarding packets according to installed rules in switch tables.

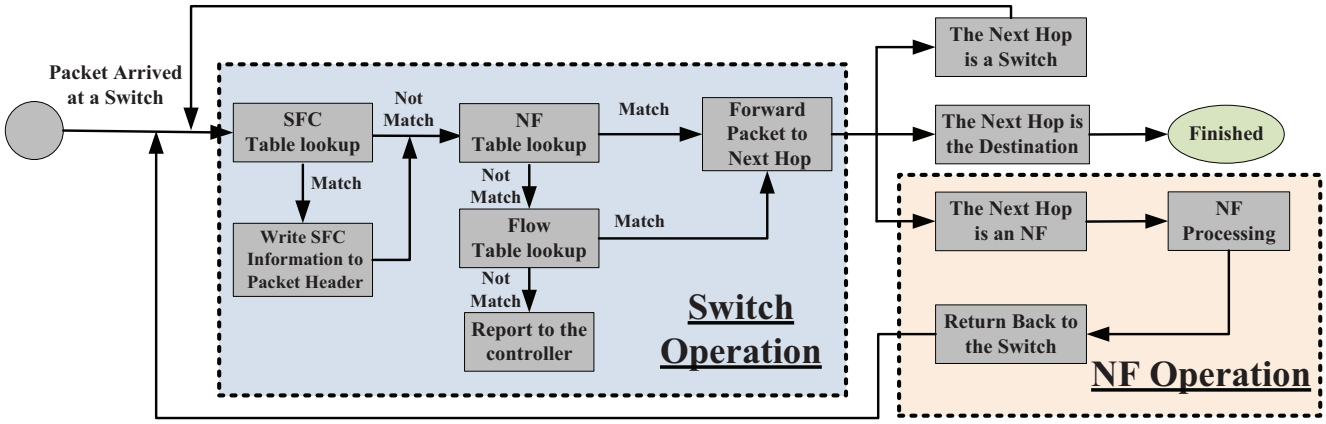

Fig. 3: Illustration of Packet Processing Procedure. When a packet arrives at a switch, the switch matches the header with SFC Table, NF Table and Flow Table in sequence. In this way, the packet will be forwarded to destination while obeying SFC constraints.

Each NF unit processes the received packets. As specified in the OpenFlow standard [30], each SDN switch contains multiple tables. We divide these tables into three parts with different roles, called *SFC Table*, *NF Table* and *Flow Table*, respectively. We will describe the design of the data plane in Section 4.

- *SFC Table* is used to store the SFC policy for each request. When a request arrives at an ingress switch, this switch will match the packet header with the *SFC Table*, and embed the matched SFC policy (as a tag) into the packet header. In other words, the packet header will contain SFC policy through matching *SFC Table* on the ingress switch.
- *NF Table* stores the next-hop information of the path from this switch to each NF. Through matching *NF Table*, the packet will be forwarded to the required NFs in sequence according to the SFC policy.
- *Flow Table* is responsible to store the next-hop information of the path (*e.g.*, default path or per-request path) from this switch to each egress switch in the network. After the packet is processed by all required NFs, it will be forwarded to destination through matching the *Flow Table*.

Control Plane is responsible to manage the whole network. We mainly focus on two new modules in the control plane: *Default Path Construction* (DPC) and *Lightweight SFC Routing Update* (LSRU). Leveraging the network information collected by switches and policy specification issued by network administrator, DPC computes the default paths from each switch to each egress switch or each NF. To avoid the possible congestion due to traffic dynamics, we also design LSRU to periodically re-compute near-optimal routing strategy based on current network conditions. The results will be encapsulated into *Flow-Mod* commands to install corresponding rules on the switches. We will introduce the design of the control plane in Section 5.

3.2 Packet Processing Procedure

We then describe the packet processing procedure of SAFE-ME. As shown in Fig. 3, the controller initially configures the *SFC Table*, *NF Table* and *Flow Table* based on the network information with a proactive manner. When a packet arrives at a switch, the switch first matches the packet header with the *SFC Table*. If there is a match, the switch will write the SFC policy (as a tag) into the packet header, which means the switch is the ingress switch of this packet and the packet is required to be processed by a set of NFs. Next, if there is a match in the *NF Table*, it will be forwarded to next hop from this switch to corresponding NF, which means the packet has to be processed by this matched NF. Otherwise, the packet need not traverse NFs or have traversed all required NFs, and will be forwarded to the destination. Then, the packet follows the traditional processing procedure. There are two cases. If there is a match in the *Flow Table*, this packet will be forwarded to the next hop according to the matching result. Otherwise, no match exists in the *Flow Table*. This packet will be reported to the controller using existing OpenFlow APIs. Note that, the switch connected with NF(s) is responsible to modify the tag in the packet header by tag shifting, which will be introduced in Section 4. In this way, after the packet is processed by the NF and returns to the connected switch, the packet will be forwarded to next required NF through matching another NF entry or forwarded to the egress switch through matching the *Flow Table*.

3.3 Illustration of SAFE-ME Design

We give an example for better understanding the packet processing. The controller initially computes the default path from each switch to each egress switch (or NF) and installs the default paths to egress switches (or NFs) on *Flow Tables* (or *NF Tables*). Besides, the administrator may specify some policies for flows. For example, in Fig. 4, the administrator specifies that traffic from subnet 10.1.1.0/24 should traverse an SFC: Firewall-IDS-Proxy for security benefits. Thus, the controller installs an SFC entry on ingress switch v_1 for this subnet in Fig. 4.

When a request from subnet 10.1.1.0/24 arrives at the ingress switch, v_1 will match this packet header with the *SFC table*, and write the tag (*i.e.*, the SFC policy: "FW-IDS-Proxy") into the packet header. The packet will then be matched with the *NF Table* (*i.e.*, "match=FW"). Since v_1 is connected with a firewall, this switch executes shift operation (which will be introduced in Section 4) to delete the "FW-" information in the tag and then forwards this packet to the firewall through outport 2. After the packet is processed by the firewall and returns to switch v_1, this

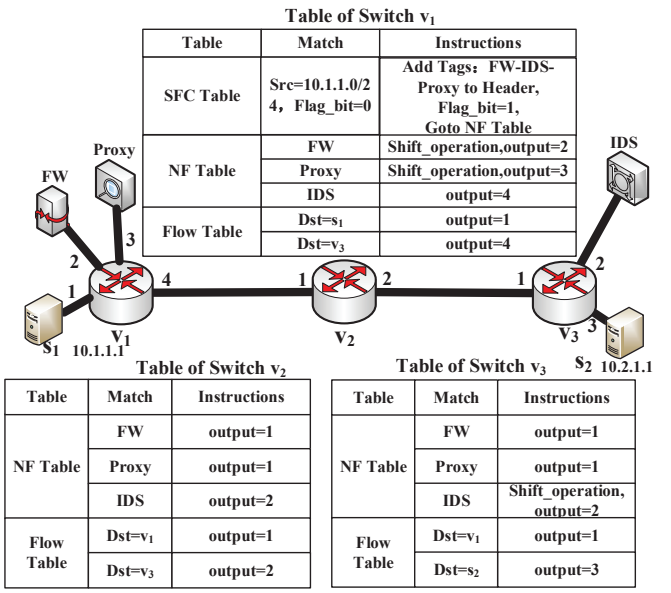

Fig. 4: Illustration of Packet Processing in SAFE-ME and Rule Installment on Switches. The administrator specifies that traffic from subnet 10.1.1.0/24 should traverse a service function chain: Firewall-IDS-Proxy for security benefits. As a result, the packet will be forwarded by path "$s_1 - v_1 -$ FW $- v_1 - v_2 - v_3 -$ IDS $- v_3 - v_2 - v_1 -$ Proxy $- v_1 - v_2 - v_3 - s_2$".

switch will match the NF entry "match=IDS and output=4", and forward this packet to switch v_2, which will then forward it to v_3 by the *NF Table*. Switch v_3 continues to forward this packet to IDS according to the *NF Table*. In this way, this packet will be forwarded through path "$s_1 - v_1 -$ FW $- v_1 - v_2 - v_3 -$ IDS $- v_3 - v_2 - v_1 -$ Proxy $- v_1 - v_2 - v_3 - s_2$". That means, by leveraging the design of the NF/Flow Tables, SAFE-ME can proactively compute the default paths from each switch to each egress switch or each NF and stores this information in NF/Flow Tables proactively. In this way, we only need to install one special SFC entry on the ingress switch for each new arrival request, and the other entries can be shared by different requests. On the contrary, as illustrated in Section 2.2, per-request routing solutions may consume several special entries for each new arrival request. Thus, SAFE-ME will greatly reduce the use of rules and the control overhead. Note that, the process for the return traffic is similar as the incoming traffic. Thus, we only depict the routing tables and describe the process for the incoming traffic for simplicity.

4 Date Plane Design

As described in Section 3.2, multiple entry tables and tag operations are two core issues in the data plane that may affect the forwarding performance. Thus, this section mainly designs the table pipeline process and tag operations (*e.g.*, tag embedding and deleting) on both OpenFlow and P4 [31] switches. Through these operations, SAFE-ME can forward requests using fewer rules and lower control overhead while satisfying SFC policies.

4.1 Pipeline Process in OpenFlow Switches

As specified by the OpenFlow standard [30], the pipeline of every OpenFlow switch consists of multiple forwarding tables.

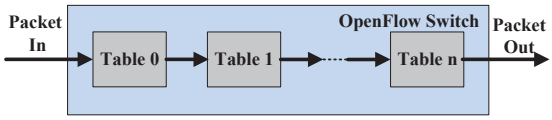

Fig. 5: Pipeline Processing in OpenFlow Switches. Each switch contains multiple tables, which are sequentially numbered. Packets can match theses tables in sequence.

Each table contains a certain number of entries. Many commercial switches (*e.g.*, Pica8 3297 switches [32], barefoot switches [33]) also contain such pipeline of multiple tables. As illustrated in Fig. 5, these tables are sequentially numbered, starting from 0. Each packet is first matched against entries of table 0. If an entry is matched, the corresponding instruction will be executed. These instructions may modify the packet header, forward the packet to a specific port or direct the packet to another table (using the Goto-Table Instruction). If no matched entry is found, a table miss occurs. The optional instructions in the table-miss entry include dropping the packet, passing it to another table, or reporting it to the controller.

We leverage the character of pipeline processing to implement the packet processing in SAFE-ME. Specifically, we define Table 0 as the SFC Table, which stores the SFC policy. When a packet arrives at the ingress switch, it will embed the tag (*i.e.*, the matched SFC policy) into the packet header by matching the SFC table. Then, the ingress switch directs the packet to further match the NF Table. We define Table 1 as the NF Table. If there is a match in the NF Table, the switch will forward the packet to the corresponding port. If there is no match, the switch will direct the packet to the Flow Table for traditional routing (*e.g.*, we define Table 2 as the Flow Table). Leveraging the pipeline processing, the switch will first match the packet header with the SFC table, then with the NF table, and finally with the Flow table in the end (if necessary). We describe the detailed tag operations in the following sections.

4.2 Tag Embedding through SFC Table

In the data plane, there may exist many units of NFs. The controller uses unique identifies (*e.g.*, $1, 2, ..., m$) to distinguish these NFs. Recent studies show that the number of NFs is similar to the number of switches [34] [35] and the length of SFC is usually no more than 5 in a moderate-size network. For the sake of convenience, we use 8 bits to represent an NF and use 40 bits to indicate an SFC. In this way, we add several fields into the packet header as shown in Fig. 6. Specifically, we design (1) *Tag Match Field* to store the first NF in the SFC and (2) *Tag Storage Field* to embed the remaining NF(s) of the SFC in the reverse order. Moreover, we use 1 flag bit to denote whether the packet has been embedded a tag or not. Note that, in data center networks, the average packet size is around 724 bytes (*i.e.*, 5792 bits) [36]. Even in a large-scale network, we may need to use 11 bits to identify 2047 different NFs and the maximum length of SFCs may be 10 [37] (*i.e.*, the cost is $11 \times 10 + 1 = 111$ bits), the bandwidth cost for embedding a tag is still negligible ($< 2\%$).

To illustrate the SFC policy (or tag) embedding process, we revisit the example in Fig. 4. We use 0x01, 0x02, 0x03 to denote the FW, Proxy, IDS units, respectively. In this way, the service function chain can be encoded into 0x01-0x03-0x02. To embed

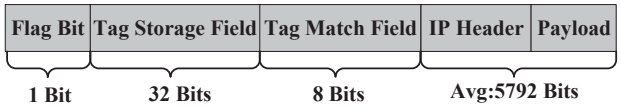

Fig. 6: Tag Storage/Match Fields and Flag Bit in a packet. Flag Bit Field indicates whether the packet has been embedded a tag or not. Tag Match Field stores the first NF in the SFC and Tag Storage Field embeds the remaining NF(s) in the SFC. The overall bandwidth cost for embedding tags is negligible.

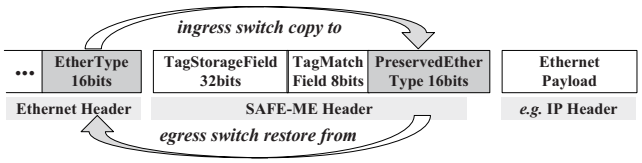

Fig. 7: The SAFE-ME Header Format in P4 Switches.

this tag, *Tag Match Field* records the first NF (*e.g.*, 0x01) and *Tag Storage Field* stores the remaining NFs in the reverse order (*e.g.*, 0x0203). In other words, the SFC entry on switch v_1 can be expressed as "$ip\_src = 10.0.1.0/24, Flag\_bit = 0, actions = \{Tag\_Match\_Field = 0x01, Tag\_Storage\_Field = 0x0203, Flag\_bit = 1, Goto\_Table : NF\_Table\}$". When a packet from subnet 10.0.1.0/24 arrives at switch v_1, it will match the entry in the SFC Table. The actions will modify the *Tag Match Field* to 0x01, *Tag Storage Field* to 0x0203 and set the *Flag Bit* to 1. As a result, the SFC policy is successfully embedded in the packet header.

4.3 Tag Shifting through NF Table

The switch will then match the *Tag Match Field* (*i.e.*, the first NF) in the NF Table, and forward the packet to the corresponding port if there is a matching entry. Moreover, if the switch is directly connected with the NF as specified by the *Tag Match Field*, it will take the following operations: (1) catch the first NF information of the rest SFC from the *Tag Storage Field* (*i.e.*, $shift\_right$ operation); and (2) reset the *Tag Match Field*. We revisit the example in Fig. 4. After embedding a tag, the switch will match the packet header in the NF table about FW (*i.e.*, 0x01). There is a matching NF entry: "$Tag\_Match\_Field = 0x01, actions = shift\_right, output = 2$", which means the switch will shift right the tag and forward the packet to FW through port 2. Note that, the $shift\_right$ operation will bitwise shift right of *Tag Storage+Match Field* by 8 bits. After the $shift\_right$ operation, *Tag Match Field* and *Tag Storage Field* become $0x03$ and $0x02$, which means the packet still has to traverse two NFs (*i.e.*, 0x03-0x02). When the packet returns to switch v_1 from FW, the switch will match the NF table with the match field "$Tag\_Match\_Field = 0x03$", and forward to IDS through port 4. In the end, the packet will be forwarded to the destination while obeying SFC policy.

4.4 SAFE-ME Design with P4 switches

Leveraging the flexibility of $P4_{16}$ programming language [38], SAFE-ME can easily be settled down to the *Programming Protocol-independent Packet Processors* (P4) [31] switches by adopting a new header after the Ethernet header. The new SAFE-ME header format is depicted in Fig 7.

To better fit into the existing network protocol stack (*e.g.*, TCP/IP), a few tunings are taken towards the SAFE-ME header format mentioned in Fig 6. The "*Tag Storage Field*" and "*Tag Match Field*" are not modified. To preserve the "*Ether Type*" value in the Ethernet header of the original packet, a 16-bits "*Preserved Ether Type*" field is appended in the new SAFE-ME header. We also choose an experimental "*Ether Type*" value "*0x0123*" for the Ethernet header to indicate the existence of the following SAFE-ME header. This particular "*Ether Type*" value provides the same usage as the "*Flag bit*" in the original SAFE-ME header. Thus, the "*Flag Bit*" field is removed.

Based on the SAFE-ME scheme, if a packet matches the SFC table at the ingress switch, a SAFE-ME header is generated and inserted after the Ethernet header. The value of "*Ether Type*" field in the Ethernet header is set to be "*0x0123*" to announce that it is followed by a SAFE-ME header. As shown in Fig 7, the switch saves the original Ethernet type of this packet to the "*Preserved Ether Type*" field in the SAFE-ME header for preservation. Then this packet is forwarded by SAFE-ME tag-shifting and NF table matching scheme. When it goes to the egress switch, the SAFE-ME header will be removed, the original Ethernet type will be retrieved and restored from the "*Preserved Ether Type*" field in the deleted SAFE-ME header. Thus, an unmodified packet as the source sent, is finally received by the destination and the SFC rules are obeyed. Please note the field preservation method used in SAFE-ME makes it suitable for any network protocols employing type field to indicate the following header. The matching and forwarding pipeline are implemented strictly following the SAFE-ME work flow described in Fig 3.

4.5 Discussions

Applicability for the Consolidated Platform. NF consolidation (*e.g.*, consolidating several NFs onto a server with containers [39]) reduces the TCAM cost and control overhead through simplifying the SFC processing in middlebox networks [26] [27] [24]. SAFE-ME addresses the complementary problem on how to optimally decide the routing selection for all requests and the rule installment at all switches. Thus, our proposed solution can be combined with the consolidated platform to achieve better network performance. Specifically, under the consolidation architecture, we can regard each consolidated platform as a mixed NF and encode SFC/NF Table(s) based on the mixed NF(s). For example, we assume that a request is required to traverse an SFC: Firewall-IDS-Proxy. Leveraging container technology, we can implement all three NFs onto a server (without loss of generality, denoted as s_c) under the consolidation architecture. Thus, we regard s_c as a mixed NF. On arriving at the network, the ingress switch embeds s_c (*i.e.*, the server for SFC processing) into the packet header of this request through matching SFC entry. Then, we forward this request to s_c for NF processing through matching the NF entry. In the end, this request will be forwarded to destination through matching Flow entry. It means SAFE-ME can be applied for the consolidation architecture with a little modification.

Flexibility of Implementing SAFE-ME. For some programmable switches (*e.g.*, Open vSwitches [40], P4-based barefoot switches [33]), it is not difficult to add new fields, such as *Tag Match Field* and *Tag Storage Field*, to embed the SFC policy into the packet header. In fact, the development of programmable data plane technology such as P4 has reduced the difficulty of implementing a SAFE-ME style data plane and thus, has greatly improved the feasibility of SAFE-ME. For other SDN switches, we can leverage some existing protocol header fields (*i.e.*, VLAN tags, MPLS labels, *etc.*) to embed the SFC policy [2] [22]. Meanwhile, the *shift* operation is a basic and high-speed function [41]. Thus, the tag operations in SAFE-ME can achieve line rate if the switch supports *shift* operation (*e.g.*, Open vSwitches [40], barefoot switches [33]). However, some switches may not support *shift* operation. Under this situation, we can leverage NF units or deploy a software programmable switch (*e.g.*, OVS) on each server running the NFs to fulfill the shift operation. Specifically, when a packet arrives at a required NF, the NF or the software switch will shift the tag in the packet header before returning to the switch so that the packet will match the next required NF. FlowTags [22] has illustrated that these operations are lightweight (<0.5% cost) for an NF to modify the packet header.Thus, SAFE-ME is quite compatible with legacy networks and easy to be implemented.

Applicability for Network Function Virtualization (NFV). In NFV networks, most existing works also adopt the per-request routing strategy [23] [42] or consolidated platform [27] [43] to schedule flows. Thus, similar to MB networks, NFV networks will also encounter routing loops, traffic dynamics and scalability problems, due to the disadvantages of existing solutions as shown in Table 1. SAFE-ME can be applied to NFV networks with some modifications. More specifically, in NFV networks, several virtual network functions (VNF) may be configured on one physical server. It means that packets processed by one VNF may go through the next VNF rather than return to the switch. To be applicable under this situation, SAFE-ME only needs some modifications. For example, if the switch is connected to two following NFs, the switch will delete the first two NFs and write the second NF in the *Tag Store Field* to the *Tag Match Field*. These modifications are easy to implement.

Applicability for Large-Scale Networks. In a large-scale network with a large number of NFs, SAFE-ME will need more bits to identify the NFs of the SFC in its header. To be applicable under this situation, we present two solutions. **(I)** The coarse-grained tag mapping scheme. In this scheme, SAFE-ME can use the tag to identify a set of NFs with the same type (*e.g.*, they are all IDS, etc.) instead of representing a single NF. The packet that matched the NF table entry will be forwarded to one NF selected by a user-defined method from a set of NFs sharing the same type. Since the types of NFs are usually much less compared with the quantity of NFs, this coarse-grained tag mapping scheme can significantly reduce the tag bit length in use, but may degrade the routing performance. Thus, there is a trade-off between the two different tag mapping schemes for different network scales and scenarios. **(II)** The region-based tag reuse scheme. A large-scale SDN network may use a distributed control plane containing several controllers [44]. Each controller may manage a set of switches that forms a local region. In this way, SAFE-ME can reuse the same tag to identify different NFs in the context of different regions. This tag reuse scheme can also significantly reduce the SAFE-ME tags' bit length.

Comparison with Segment Routing over IPv6 (SRv6). To enforce the policy for SFC, SRv6 [45] adopts a source routing mechanism by using an ordered list of tags (each tag is a 128-bit length IPv6 address) in the segment routing header [46]. Compared with SRv6, SAFE-ME is also based on source routing and tag-based forwarding scheme, but has three apparent advantages. **(I)** SAFE-ME introduces much less overhead in the header. For example, SAFE-ME can use an 8-bit length tag to represent an NF when the total number of NFs is 256, while SRv6 needs 128-bit. Thus, SAFE-ME can reduce the header overhead by more than 93% compared with SRv6 in this case. **(II)** SAFE-ME is much more flexible. SRv6 uses the IPv6 header which is a Layer-3 protocol. The SAFE-ME header can be added at any position into the packet header stack based on the need and design. **(III)** The SAFE-ME routing scheme is much more fine-grained and efficient. SRv6 basically uses an IPv6 address to match and forward the packet. However, SAFE-ME can provide dynamic-granularity SFC routing based on various matching rules in the SFC table. Meanwhile, the SAFE-ME control plane introduced in Section 5 also ensures excellent network performance.

5 Control Plane Design

5.1 Default Path Construction (DPC) for SFC Routing

Once the network topology is established, the controller can obtain the topology information, such as the locations/connections of all switches and NFs, through classical OpenFlow APIs. The data plane topology can be modeled as a directed graph $G = (U \cup V \cup N, E)$, where U, V, N and E denote the server set, the switch set, the NF set and the directed link set, respectively. As described in Section 2.1, the traditional default path solution cannot be directly applied for middlebox networks. Thus, we propose novel multi-level (*i.e.*, policy-level, NF-level and switch-level) default paths for SFC routing.

5.1.1 Policy-level Default Path Construction

In the middlebox networks, the network operator usually specifies different sequences of NFs (*i.e.*, SFCs) for different requests. For example, in Fig. 4, the operator may specify that requests from subnet 10.1.1.0/24 (*e.g.*, s_1) have to traverse an SFC: Firewall-IDS-Proxy for security benefits. The DPC module will transform this specification into policy-level default paths and install corresponding entries in the *SFC Table* of the ingress switch.

5.1.2 NF-level Default Path Construction

DPC first leverages classical algorithms (*e.g.*, OSPF or ECMP) to compute default path(s) from each switch to each NF. Each switch then stores the next-hop information on the default path to each NF in the *NF Table* so that each packet will be processed by the required NF(s) in sequence. As a result, each switch will install $|N|$ entries in the *NF Table* for NF-level default paths construction.

5.1.3 Switch-level Default Path Construction

Similarly, DPC leverages classical algorithms to compute default path(s) from each switch to each egress switch. The controller then installs switch-level wildcard rules in the *Flow Table* of each switch through Flow-Mod messages. Moreover, each egress switch will install one rule for each connected destination. For example, in Fig. 4, switch v_1 installs two rules in *Flow Table*: one for egress switch v_3 and the other for the connected destination s_1.

5.1.4 Handling Switch/NF/Link Failures

Although the network topology is expected to be stable to a large extent, we may still encounter switch/NF/link failures in practice. In such cases, the controller needs to construct new default paths and install/update rules on switches. For the single NF/link/switch failure scenario(s), we can address this issue by pre-computing alternative default paths. For multi-NF failures, SAFE-ME only needs to modify the affected SFC entries so as to redirect requests to other available NFs. For multi-link/switch failures, we re-compute default paths and update corresponding rules on switches. With the help of our data plane design, SAFE-ME can deal with various failure events only by modifying a small number of rules, which will be verified in Section 6.2.

5.2 Lightweight SFC Routing Update for Traffic Dynamics

With the help of multi-level default paths, requests will be forwarded to destinations while obeying SFC policies. Default paths help to save TCAM resources and relieve control overhead, but they cannot guarantee the network performance (*e.g.*, NF/link load balancing or throughput maximization), due to traffic dynamics. Thus, we design the Lightweight SFC Routing Update (LSRU) module by joint default paths and per-request paths for network optimization.

5.2.1 Exploration of Feasible SFC Paths

In middlebox networks, each request may have to traverse multiple NFs in sequence. The number of feasible SFC routing paths for each request may be exponential and the network performance will be affected by the selection of SFC routing paths. Thus, we compute a set of feasible paths that satisfy SFC policy for each request. To decrease time complexity, we pre-compute the feasible SFC path set for each request only when topology changes. The feasible path set can be computed by traditional algorithms, such as depth-first search. We give an example to illustrate the exploration of feasible paths. Assume that requests from s_1 to s_2 need go through FW-IDS and there exists 2 FWs and 3 IDSs. We can compute k-shortest paths using Yen's algorithm [47] from s_1 to each FW, from each FW to each IDS, from each IDS to s_2, respectively. This algorithm computes single-source k-shortest loopless paths on a graph with non-negative edge cost. By this way, there exists $2 \cdot k$ feasible paths from s_1 to FWs, $6 \cdot k$ feasible paths from FWs to IDSs, and $3 \cdot k$ feasible paths from IDSs to s_2. As a result, we have explored $36 \cdot k^3$ feasible paths from s_1 to s_2 while satisfying the SFC policy. If there are too many of them, we may include only a certain number (*e.g.*, 3-5) of best ones under a certain performance criterion, such as having the shortest number of hops or having the large capacities. Note that, the exploration of feasible paths is pre-computed and only triggered by topology changes. During the update process (Section 5.2.3), we can select one optimal SFC path from the feasible paths set for each request.

5.2.2 Installment of A Feasible SFC Path

When the controller decides to re-route a request from its default path to another feasible path p, under the traditional wisdom, the controller will totally deploy ω forwarding rules at switches along path p, where ω is the number of times switches appear on path p [2]. However, this scheme will cost many entries and lead to massive control overhead. To reduce the resource cost, we can leverage SAFE-ME to install default rules so as to improve network scalability. Since each ingress switch only maintains one SFC entry in *SFC Table* for each request from this ingress switch, we just consider the forwarding entry cost on the *Flow Table* and the *NF Table*.

Let variables $I^n(f,p,v)$ and $I^f(f,p,v)$ (both initialized to 0) denote the number of required NF entries and the number of required flow entries on switch v, respectively, as the route of request r is updated to the target path p. Assume that the request r has to traverse q NFs, denoted as $NF_1, NF_2, ..., NF_q$, respectively. We determine the values of $I^n(f,p,v)$ and $I^f(f,p,v)$ as follows: 1) We divide the path p into $q+1$ path segments (*i.e.*, ingress switch to NF_1, NF_1 to NF_2, ..., NF_q to egress switch). 2) We use p_d to denote each path segment on path p, where d is the destination of this path segment. For example, p_{NF_1} denotes the path segment from ingress switch to NF_1. 3) For each switch v on p_d, if path segment p_d overlaps with the default path from switch v to d, then there is no need to deploy an entry for this path segment on switch v; otherwise, a flow/NF entry on switch v for this path segment should be deployed. If d is an NF, $I^n(f,p,v) = I^n(f,p,v) + 1$, which means an NF entry should be deployed on the *NF Table*. If d is a server, $I^f(f,p,v) = I^f(f,p,v) + 1$, which means a flow entry should be deployed on the *Flow Table*. After traversing all path segments and all switches on path p, we obtain the values of variables $I^n(f,p,v)$ and $I^f(f,p,v)$.

5.2.3 Problem Definition for SFC Routing Update

We denote the set of switches as $V = \{v_1, ..., v_{|V|}\}$, the set of servers as $U = \{u_1, ..., u_{|U|}\}$, and the set of NFs as $N = \{n_1, ..., n_{|N|}\}$. The data plane topology is modeled as a graph $G = (U \cup V \cup N, E)$, where E is the set of links. Let $c(e)$ (or $c(n)$) be the capacity of a link e (or an NF n) and $l(e)$ (or $l(n)$) be its current load. The switches measure traffic loads on all their ports (*i.e.*, adjacent links) and make the information available to the controller through OpenFlow [48] or other statistics collection mechanisms [49] [50]. Since each middlebox is directly connected with a switch, the switch can also measure the middlebox load through port statistics collection.

When the network performance gets worse (*e.g.*, higher link/NF load), the controller selects a subset Π of the largest requests (reported by the switches) for re-routing so as to achieve various performance optimization. The budget for re-routing execution time constraints the size of Π. More execution time budget

means we can re-route more requests, which can be roughly estimated based on the past executions. The estimated rate of request $f \in \Pi$ is denoted as $r(f)$, which can be obtained through switches. Let $\mathcal{P}(f)$ be the set of feasible paths for request f. $\mathcal{P}(f)$ is determined based on the management policies and performance objectives, as discussed in Section 5.2.1. Note that, $\mathcal{P}(f)$ also contains the path $p^*(f)$ that the request is currently routed through.

Let $T^n(v)$ and $T^f(v)$ be the number of residual entries in the *NF Table* and the *Flow Table*, respectively, at switch v. Let $I^n(f,p,v)$ (or $I^f(f,p,v)$) be the number of required NF entries (or flow entries) on switch v if path p is assigned to request f, which has been discussed in Section 5.2.2. Note that, the number of required SFC entries is related to the number of requests and is independent of update process. Thus, we do not consider the *SFC Table* constraint here. We formalize the lightweight SFC routing update problem in middlebox networks (LSRU-MB) as follows:

$$\min \quad \lambda$$
$$s.t. \begin{cases} b(e) = l(e) - \sum_{f\in\Pi: e\in p^*(f)} r(f), & \forall e \in E \\ b(n) = l(n) - \sum_{f\in\Pi: n\in p^*(f)} r(f), & \forall n \in N \\ \sum_{p\in\mathcal{P}(f)} y_f^p = 1, & \forall f \in \Pi \\ \sum_{f\in\Pi}\sum_{p\in\mathcal{P}(f):v\in p} y_f^p \cdot I^n(f,p,v) \le T^n(v), & \forall v \in V \\ \sum_{f\in\Pi}\sum_{p\in\mathcal{P}(f):v\in p} y_f^p \cdot I^f(f,p,v) \le T^f(v), & \forall v \in V \\ b(e) + \sum_{f\in\Pi}\sum_{p\in\mathcal{P}(f):e\in p} y_f^p r(f) \le \lambda \cdot c(e), & \forall e \in E \\ b(n) + \sum_{f\in\Pi}\sum_{p\in\mathcal{P}(f):n\in p} y_f^p r(f) \le \lambda \cdot c(n), & \forall n \in N \\ y_f^p \in \{0,1\}, & \forall p, f \\ \lambda \le 1. \end{cases} \tag{1}$$

where $y_f^p \in \{0,1\}$ means whether request f will be forwarded through path $p \in \mathcal{P}(f)$ or not. The first and second sets of equations compute the link background traffic load $b(e), \forall e \in E$, and the NF background traffic load $b(n), \forall n \in N$, when the flows in Π are taken out. The third set of equations requires that request $f \in \Pi$ will be forwarded through a single path from $\mathcal{P}(f)$. The fourth set of inequalities describes the NF table size constraint, while the fifth set of inequalities describes the flow table size constraint on switches. The sixth and seventh sets of inequalities indicate that the traffic load on each link e and each NF n, respectively, where λ is called as the load-balancing ratio.

The optimization objective is to minimize λ. Achieving load balance among links/NFs helps prevent large queuing delays that happen when λ approaches towards 1. Moreover, it also leaves room for new requests or allows the existing requests to increase their rates for better network throughput.

Theorem 1. LSRU-MB defined in Eq. (1) is an NP-hard problem.

Proof: We consider a special example of the LSRU-MB problem, in which only one flow needs to traverse one middlebox. This special case of LSRU-MB has been proven to be NP-hardness [51]. Thus, the LSRU-MB problem is NP-Hard too. □

5.2.4 Algorithm Design

In this section, we present an approximate algorithm, called Rounding-based SFC Routing Update (RBSU), to solve this problem. We first relax Eq. (1) by replacing the eighth line of integer constraints with $y_f^p \ge 0$, turning the problem into linear programming. We can solve it with a linear program solver (*e.g.*, CPLEX [52]) and the solution is denoted by $\widetilde{y}$ and $\widetilde{\lambda}$. As the linear program is a relaxation of the LSRU-MB problem, $\widetilde{\lambda}$ is a lower-bound result for LSRU-MB. Using randomized rounding method [53], we obtain an integer solution $\widehat{y}_f^p$. More specifically, variable $\widehat{y}_f^p$ is set as 1 with the probability of $\widetilde{y}_f^p$. The RBSU algorithm is formally described in Algorithm 1.

Algorithm 1 RBSU: Rounding-based SFC Routing Update for Middlebox Networks

Step 1: Solving the Relaxed LSRU-MB Problem
Construct a linear program by replacing the integral constraints with $y_f^p \ge 0$
Obtain the optimal solution $\{\widetilde{y}_f^p\}$
Step 2: Route Update for Middlebox Networks
Derive an integer solution $\{\widehat{y}_f^p\}$ by randomized rounding
for each sampled flow $f \in \Pi$ **do**
  for each SFC route $p \in \mathcal{P}(f)$ **do**
    if $\widehat{y}_f^p = 1$ **then**
      Appoint a path p for flow f
    end if
  end for
end for

5.2.5 Approximation Performance Analysis

We analyze the approximate performance of the proposed RBSU algorithm. Assume that the minimum capacity of all the links and NFs is denoted by $c_{\min}$ and the set of all flows is denoted as Γ. We define a constant α as follows:

$$\alpha = \min\{\min\{\frac{\widetilde{\lambda} c_{\min}}{r(f)}, f \in \Gamma\}, \min\{T^n(v), T^f(v), v \in V\}\} \tag{2}$$

Lemma 2. Taking the optimal solution obtained from the set of the feasible paths exploited in Section 5.2.1 as a benchmark, the proposed RBSU algorithm can achieve the approximation factor of $\frac{3\log n}{2\alpha} + 2$ for link/NF capacity constraints in large networks, where n is the number of switches.

Proof: We denote the traffic load of link $e \in E$ from flow $f \in \Gamma$ as $x_{f,e}$. According to the definition, $x_{f_1,e}$, $x_{f_2,e}$, ... are mutually independent. The expected traffic load on link e is:

$$\begin{aligned} \mathbb{E}\left[\sum_{f\in\Gamma} x_{f,e}\right] &= \sum_{f\in\Pi} [x_{f,e}] + b(e) \\ &= \sum_{f\in\Pi}\sum_{e\in p: p\in\mathcal{P}(f)} \widetilde{y}_f^p \cdot r(f) + b(e) \le \widetilde{\lambda} c(e) \end{aligned} \tag{3}$$

Combining Eq. (3) and the definition of α, we have

$$\begin{cases} \frac{x_{f,e}\cdot\alpha}{\widetilde{\lambda} c(e)} \in [0,1] \\ \mathbb{E}\left[\sum_{f\in\Gamma} \frac{x_{f,e}\cdot\alpha}{\widetilde{\lambda}\cdot c(e)}\right] \le \alpha. \end{cases} \tag{4}$$

Thus, Chernoff bound [14] can be applied. Assume that ρ is a arbitrary positive value. It follows

$$\mathbf{Pr}\left[\sum_{f\in\Gamma}\frac{x_{f,e}\cdot\alpha}{\widetilde{\lambda}\cdot c(e)}\leq(1+\rho)\cdot\alpha\right]\leq e^{\frac{-\rho^2\cdot\alpha}{2+\rho}}$$

$$\Leftrightarrow\mathbf{Pr}\left[\sum_{f\in\Gamma}\frac{x_{f,e}}{\widetilde{\lambda}\cdot c(e)}\leq(1+\rho)\right]\leq e^{\frac{-\rho^2\cdot\alpha}{2+\rho}} \quad (5)$$

Now, we would assume that

$$\mathbf{Pr}\left[\sum_{f\in\Gamma}\frac{x_{f,e}}{\widetilde{\lambda}\cdot c(e)}\geq(1+\rho)\right]\leq e^{\frac{-\rho^2\cdot\alpha}{2+\rho}}\leq\frac{1}{n} \quad (6)$$

We know that $\frac{1}{n}\to 0$ when the network grows. By solving Eq. (6), we have the following result

$$\rho\geq\frac{\log n+\sqrt{\log^2 n+8\alpha\log n}}{2\alpha},\quad n\geq 2$$

$$\Rightarrow\rho\geq\frac{\log n+\sqrt{(2\log n+2\alpha)^2}}{2\alpha}=\frac{3\log n}{2\alpha}+1, n\geq 2 \quad (7)$$

Thus, the approximate factor for link capacity constraints is $\rho+1=\frac{3\log n}{2\alpha}+2$. Similarly, we can obtain the approximation factor for the NF capacity constraints. □

Lemma 3. In most practical scenarios, *e.g.*, $\alpha\geq 3\log n$, the proposed RBSU algorithm can achieve the approximation factor of 2 for link/NF capacity constraints.

Proof: When $\alpha\geq 3\log n$, by solving Eq. (6), we have:

$$\rho\geq\frac{\log n+\sqrt{(\log n-2\alpha)^2-4\alpha^2+12\alpha\log n}}{2\alpha}, n\geq 2$$

$$\Rightarrow\rho\geq\frac{\log n+\sqrt{(\log n-2\alpha)^2}}{2\alpha}\Rightarrow\rho\geq 1 \quad (8)$$

Thus, the bound can be tightened to $\rho+1=2$. □

Lemma 4. Taking the optimal solution obtained from the set of the feasible paths exploited in Section 5.2.1 as a benchmark, after the rounding process, the amount of required flow/NF entries on any switch v will not exceed the number of residual flow/NF entries by a factor of $\frac{3\log n}{2\alpha}+2$ in large networks, where n is the number of switches.

Lemma 5. In most practical scenarios, *e.g.*, $\alpha\geq 3\log n$, the proposed RBSU algorithm can achieve the approximation factor of 2 for flow/NF table size constraints.

The proofs of Lemmas 4 and 5 are similar to that of Lemmas 2 and 3, thus we omit the detailed proofs here.

Approximation Factor: Following from our analysis, the capacity of links/NFs will hardly be violated by a factor of more than $\frac{3\log n}{2\alpha}+2$, and the flow/NF table size constraints will not be violated by a factor of $\frac{3\log n}{2\alpha}+2$. Thus, we can conclude that RBSU can achieve the bi-criteria approximation factor of $(\frac{3\log n}{2\alpha}+2, \frac{3\log n}{2\alpha}+2)$. Under proper assumption (*i.e.*, $\alpha\geq 3\log n$), the bound can be tightened to $(2,2)$. It means that RBSU can minimize the network load ratio to no more than $2\widetilde{\lambda}$ and the flow/NF table size constraints are violated at most by a multiplicative factor 2. Note that, as to the cases where the flow/NF table size constraints are violated, the default entries will take effect to transfer these extra flows, avoiding packets dropping.

6 Performance Evaluation

In this section, we evaluate the scalability and efficiency of SAFE-ME. All code has been publicly available at github[1]. We first give the metrics and benchmarks for performance comparison (*Section 6.1*). We then perform a small-scale test with P4 switches and give the performance analysis (*Section 6.2*). Finally, we evaluate the performance of several middlebox systems with large-scale simulations (*Section 6.3*). Note that, we also implement SAFE-ME with OVS and the conclusion agrees with the above results. Due to the limited space, we omit the description of the experiment results with OVS. The reader can refer [1] for the detailed description.

6.1 Performance Metrics and Benchmarks

Performance Metrics: We adopt the following three sets of metrics to evaluate the scalability and efficiency of our proposed system. 1) SAFE-ME involves tag operations, which may increase the packet transmission delay and decrease the end-to-end throughput. Thus, we adopt *end-to-end delay* and *end-to-end throughput* to evaluate the efficiency of tag operations. Specifically, we use Ping and Qperf [54] tools to measure the delay of ICMP and TCP/UDP protocols between two servers, respectively. In our implementation, some flows are aggregated into one request. We use Packet Generator (PktGen) tool [55] to measure its flow completion time (FCT). Besides, we adopt vnStat tool [56] to measure the end-to-end throughput, which can evaluate the negative impact of tag operations on the packet forwarding rate. 2) Considering traffic dynamics (*e.g.*, request intensity fluctuation), we need re-route flows to better deal with traffic dynamics (*i.e.*, execute the RBSU algorithm). During the update process, we focus on two metrics: *update delay* and *control traffic overhead*. Specifically, we measure the duration of the update procedure as *update delay*. Moreover, we record the total traffic amount between the control plane and the data plane during the update procedure as *control traffic overhead*. Obviously, lower update delay and control traffic overhead represent better network update performance. After route update, we measure the *total number of required entries* on three tables of each switch and the *link/NF Load* on each link/NF. Accordingly, we can obtain the maximum value and CDF performance of these metrics. 3) Network failure is a common scenario in today's networks. Thus, we measure the *failure response time* to deal with various network failures, such as single/multiple NF/link/switch failures. We measure the duration from failure occurrence to failure recovery as *failure response time*.

Benchmarks: We compare SAFE-ME with other two benchmarks for evaluation. The first benchmark is the most related work, SIMPLE [2], which is an SDN-based policy enforcement layer to simplify middlebox traffic steering. To account for both the middlebox processing capacity constraint and the TCAM table size constraint, SIMPLE first pre-computes several feasible physical sequences for each request while tackling the switch resource constraints, and then chooses a physical sequence for each request to minimize the maximum middlebox load. The second benchmark is an online algorithm, called primal-dual-update-algorithm

1. https://github.com/xipeng-ahnu/SAFE-ME/tree/master.

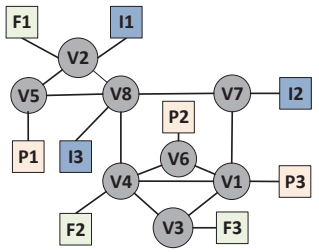

Fig. 8: Telstra Topology. Circles represent switches and squares represent NFs.

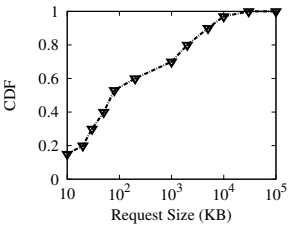

Fig. 9: CDF vs. Request Size generated by Packet Generator on Telstra.

(PDA) [4]. PDA achieves the trade-off optimization between the throughput competitiveness and QoS requirements under both link and NF capacity constraints. Note that, due to the inapplicability of traditional default path solutions as shown in Section 2.1, we have not found prior work in this direction. Thus, we decided to compare SAFE-ME with SIMPLE and PDA, two solutions that schedule and forward traffic at granularity of requests.

6.2 Small-scale Experiments with P4 Software Switch

P4 implementation of SAFE-ME: To enhance our work, we implement SAFE-ME on a software data plane consists of switches based on P4. The P4 software switches named with *simple_switch_grpc* base on the *behavioral model version 2* (bmv2) [57] are employed to construct our small-scale topology called Telstra from the Rocketfuel dataset [58], as depicted in Fig. 8.

Experimental Settings: Since the topology Telstra does not provide NF information, we utilize VNF mechanism [35] to deploy three types of NFs (*i.e.*, Firewall, IDS, and Proxy) and place 3 units for each type of NF for simplicity. In other words, we deploy total 3×3=9 NFs on the Telstra topology. We run each P4 software switch on a Ubuntu 16.04 server (kernel version 4.15.0-142) with Xeon Gold 6152 processor and 128GB of RAM, each NF installed on a high-end workstation with Intel Core i9-10900 processor and 64GB of RAM and each host with Intel Core i5-10400 processor and 16GB of RAM. We write P4 programs for SAFE-ME, PDA and SIMPLE. By utilizing $P4_{16}$ compiler (p4c-bm2-ss) [59], we compile and load them into the P4 software switches to run the data plane. For the control plane, we use a server with the same hardware as the switch to implement the algorithms for the three schemes, and use gRPC [60] protocol to control the P4 data plane. Links between switches or switch-to-NF are 10Gbps, other links including switch to host and switch to controller are 1Gbps.

We use PktGen [55] to generate network traffic, which is a powerful tool also used by [61] [62]. By using PktGen, we can generate requests with various sizes and patterns, and collect FCT, load information through PktGen APIs. In the experiments, we generate DCTCP (data center TCP) pattern requests [55] and the request size distribution is shown in Fig. 9. All requests have to traverse either *Firewall-IDS-Proxy* or *Firewall-IDS*.

FCT and Throughput Performance: In the first set of experiments, we generate TCP requests with a duration of 30s and measure the FCT and end-to-end throughput performance. The end-to-end throughput can be derived by the vnStat tool [56] through measuring the average throughput of one server port per 6-second interval. The results in Figs. 10-13 show that our proposed SAFE-ME system achieves similar FCT and throughput performance compared with other two algorithms. It also fit our previous work using OVS. Thus, we can conclude that the tag operations of SAFE-ME are lightweight and will not significantly impact the network performance.

Update Performance: In the second set of experiments, we conduct the traffic dynamics, which require to dynamically adjust routing paths and update forwarding entries for load balancing. In Fig. 11, FCT of nearly 50% requests is less than 10ms. Thus, if we update routing paths at a low speed, the network performance will be greatly reduced. Figs. 14-15 show that SAFE-ME can reduce update delay and control traffic overhead by about 87% and 85%, respectively, compared with other two solutions. SAFE-ME can achieve lower update delay because it greatly reduces the number of required entries by about 89% for updating compared with other algorithms, as shown in Fig. 16. The rule installation speed is about 0.517 ms/rule on P4 software switches by our test, which is almost twice that of using OVS and will significantly affect update performance. Figs. 17-18 show link/NF load conditions for these three systems. Using Alg. 1, SAFE-ME achieves better link load balancing and similar NF load balancing compared with SIMPLE/PDA. For example, by Fig. 17, SAFE-ME reduces the maximum link load by about 12% and 16% compared with SIMPLE and PDA, respectively. Note that, we can also tweak RBSU to work for the other two schemes and obtain a similar link/NF load balancing performance. However, without the support of the SAFE-ME's data plane, both SIMPLE and PDA would still require a higher number of flow entries, causing larger control overhead and longer update delays, similar to what is shown in Figs. 14-16, even after adopting RBSU.

Dealing with Failures: The network may encounter switch/NF/link failures in practice. We consider four failure scenarios on the Telstra topology: (I) single-NF failure, (II) single-link/switch failure, (III) multi-NF failures, and (IV) multi-link/switch failures. Under all four scenarios, the controller should re-route requests and we focus on the failure response delay to reconfigure the network. When network failure occurs, the controller needs to be aware of failures, compute new rules and install them on switches. For single NF/link/switch failure scenario(s), PDA takes more time to compute and install new rules. SIMPLE pre-computes pruned sets for the single NF failure scenario. Thus, the time cost is mainly for installing rules on switches in SIMPLE. SAFE-ME only adjusts fewer affected SFC entries to embed requests with other available NFs (*e.g.*, nearest available NFs). The number of updated entries for route update of SAFE-ME is less than that of other two benchmarks. Thus, the results in Fig. 19 show that SAFE-ME can reduce the failure response time by about 90% and 85% compared with PDA and SIMPLE, respectively.

From these experimental results, we can conclude that with the support of the SAFE-ME's data plane, SAFE-ME can greatly improve scalability (*e.g.*, TCAM cost, update delay, control overhead and response time) compared with state-of-the-art solutions.

2. I: single-NF failure, II: single-link/switch failure, III: multi-NF failures, IV: multi-link/switch failures

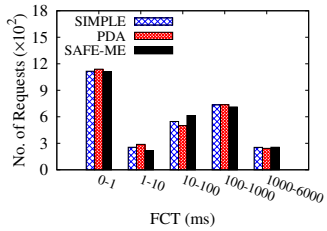

Fig. 10: No. of Requests vs. FCT on Telstra

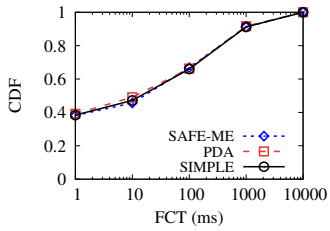

Fig. 11: CDF vs. FCT on Telstra

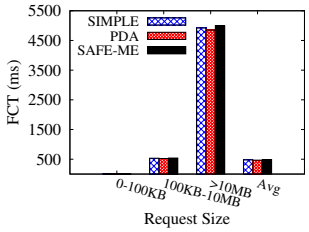

Fig. 12: FCT vs. Request Size on Telstra

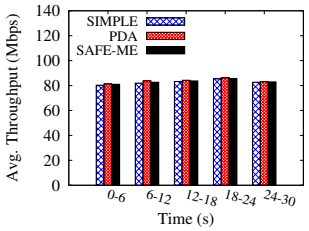

Fig. 13: Avg. Throughput vs. Time on Telstra

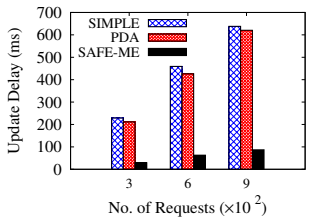

Fig. 14: Update Delay vs. No. of Requests on Telstra

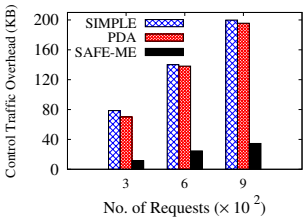

Fig. 15: Control Traffic Overhead vs. No. of Request on Telstra

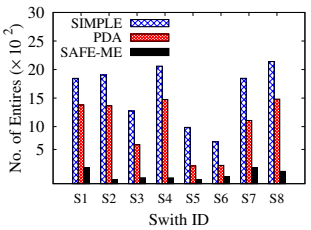

Fig. 16: No. of Entries on Each Switch on Telstra

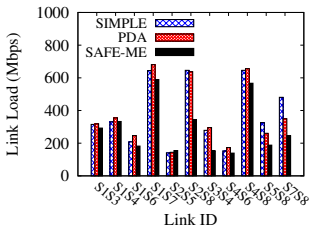

Fig. 17: Link Load of Each Link on Telstra

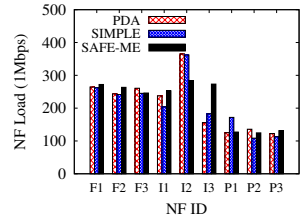

Fig. 18: NF Load of Each NF on Telstra

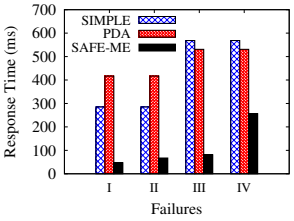

Fig. 19: Response Time vs. Failures on Telstra [2]

6.3 Large-scale Simulations

To complement small-scale testbed experiments, we conduct large-scale simulations to deep-dive into SAFE-ME. Our simulation results further confirm the superior performance of SAFE-ME compared with other two benchmarks.

Simulation Settings: In the large-scale simulations, we use packet traces of our campus network, which is shared at dropbox[3]. We simulate the traces across the Rocketfuel project [58], called Ebone, which contains 87 switches and 348 servers. Since this topology does not provide any NF information, similar to small-scale experiments, we adopt VNF placement scheme [35] to deploy 5 types of NFs (*i.e.*, Firewall, IDS, IPSec, Proxy and WAN-opt) and the number of each type of NF is set as 10 by default. In other words, we deploy totally 5×10=50 NFs on the Ebone topology. There exist four SFCs (*i.e.*, FW-IDS-IPSec, FW-Proxy, FW-IDS-IPSec-WAN-opt and IDS-Proxy), and each request will be assigned with one of SFC requirements. Note that, since the campus network is different from Ebone, we use a gravity model to map requests to ingress/egress switches [58]. We execute each simulation 50 times and average the numerical results.

Flow Entry Resource: We first compare the required entry resources of these three systems. As shown in Figs. 20-21, with the increasing number of requests, the maximum and average number of required entries increases for all systems. In comparison, the proposed SAFE-ME system uses much fewer entries than other two solutions. For example, when there are 36×10^3 requests, SAFE-ME uses a maximum number of 11,900 entries among all switches, while SIMPLE and PDA use 36,500 and 29,500 entries, respectively; SAFE-ME needs 2,600 entries on average, while both SIMPLES and PDA need about 9,000 entries. In other words, SAFE-ME can reduce the maximum number of required entries by about 68% and 60% compared with SIMPLE and PDA, respectively. Meanwhile, SAFE-ME reduces the average number of required entries by about 71% compared with the other two solutions. Fig. 22 shows the CDF of the number of entries under a fixed number (*e.g.*, 36×10^3) of requests. We observe that about 2.2% of switches need more than 8,000 entries by SAFE-ME, while over 45% of switches need more than 8,000 entries by SIMPLE and PDA.

Bandwidth Resource: Figs. 23-25 give the comparisons of bandwidth resource consumption for these algorithms. We claim that SAFE-ME can save bandwidth resources through well-designed routing strategy. For example, when there are 36×10^3 requests, our proposed algorithm can reduce the maximum/average link load by about 23%/28% and 51%/30% compared with SIMPLE and PDA, respectively. Fig. 25 shows the CDF of link load ratio under a fixed number (*e.g.*, 36×10^3) of requests. We observe that over 64% of links undertake load less than 4Gbps while only 53% (or 51%) of links undertake load less than 4Gbps by SIMPLE (or PDA).

NF processing Resource: Figs. 26-27 show the comparisons of NF loads for different algorithms. From these two figures, we observe that SAFE-ME can achieve similar NF load performance compared with both SIMPLE and PDA. Note that, since we assume all requests can be served by the required NFs, the average NF loads of these solutions are the same.

From these simulation results, we can draw some conclusions. First, from Figs. 20-22, SAFE-ME reduces the number of required entries by about 70% on average compared with other two solutions for serving all requests in the network. Second, from Figs.

3. Traces are shared at https://github.com/xipeng-ahnu/SAFE-ME/tree/master/trafficTrace

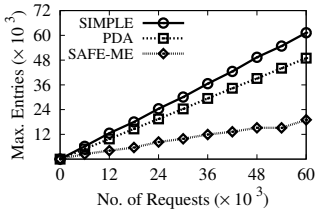

Fig. 20: Max. No. of Entries vs. No. of Requests on Ebone

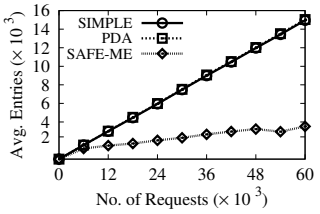

Fig. 21: Avg. No. of Entries vs. No. of Requests on Ebone

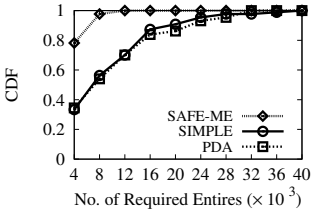

Fig. 22: CDF vs. No. of Required Entries on Ebone

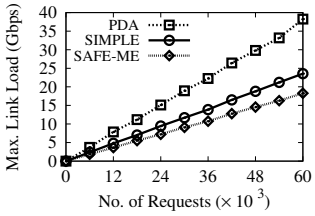

Fig. 23: Max. Link Load vs. No. of Requests on Ebone

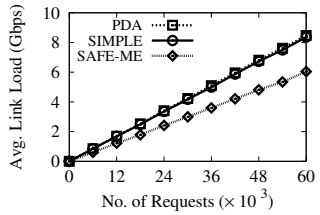

Fig. 24: Avg. Link Load vs. No. of Requests on Ebone

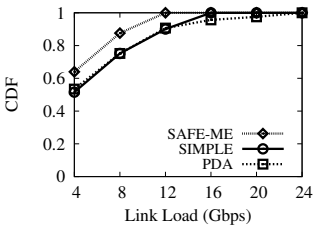

Fig. 25: CDF vs. Link Load on Ebone

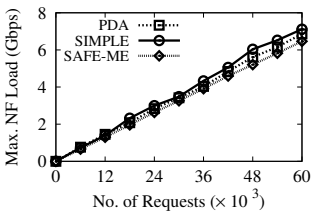

Fig. 26: Max. NF Load vs. No. of Requests on Ebone

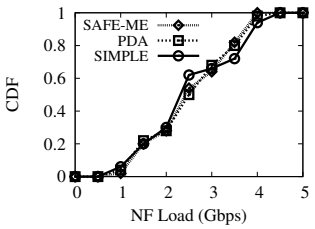

Fig. 27: CDF vs. NF Load on Ebone

23-25, SAFE-ME reduces the link load by about 30% on average compared with SIMPLE and PDA. Finally, from Figs. 26-27, we believe SAFE-ME can achieve similar NF load compared with SIMPLE and PDA, which consume more entry and bandwidth resources than SAFE-ME.

7 Conclusion

In this paper, we leverage the advantages of SDN to proactively deploy NF-based default paths so that requests can be forwarded to destination while obeying SFC policy with less resource (*e.g.*, TCAM) consumption. We further study the joint optimization of default path and per-request routing to update the SFC routing paths. With the help of default paths, we only need modify fewer rules when encountering traffic dynamics or link/switch/NF failures, which means SAFE-ME greatly reduces response time for network failures and update delay for re-route process.

References

[1] G. Zhao, H. Xu, J. Liu, C. Qian, J. Ge, and L. Huang, "SAFE-ME: scalable and flexible middlebox policy enforcement with software defined networking," in *27th IEEE International Conference on Network Protocols, ICNP 2019*. IEEE, 2019, pp. 1–11.

[2] Z. A. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using SDN," in *ACM SIGCOMM 2013 Conference, SIGCOMM 2013*. ACM, 2013, pp. 27–38.

[3] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[4] L. Guo, J. Z. F. Pang, and A. Walid, "Dynamic service function chaining in sdn-enabled networks with middleboxes," in *24th IEEE International Conference on Network Protocols, ICNP 2016*. IEEE Computer Society, 2016, pp. 1–10.

[5] G. Liu, S. Guo, B. Li, and C. Chen, "Joint traffic-aware consolidated middleboxes selection and routing in distributed sdns," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 2, pp. 1415–1429, 2021.

[6] N. P. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking, HotSDN '14*. ACM, 2014, pp. 175–180.

[7] G. Zhao, H. Xu, J. Fan, L. Huang, and C. Qiao, "Achieving fine-grained flow management through hybrid rule placement in sdns," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 3, pp. 728–742, 2021.

[8] H. Song, S. Guo, P. Li, and G. Liu, "FCNR: fast and consistent network reconfiguration with low latency for SDN," *Comput. Networks*, vol. 193, p. 108113, 2021.

[9] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, and Y. Sun, "Minimizing controller response time through flow redirecting in sdns," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 562–575, 2018.

[10] M. A. Togou, D. A. Chekired, L. Khoukhi, and G. Muntean, "A hierarchical distributed control plane for path computation scalability in large scale software-defined networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 16, no. 3, pp. 1019–1031, 2019.

[11] S. Kandula, S. Sengupta, A. G. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference, IMC 2009*. ACM, 2009, pp. 202–208.

[12] Noviflow. (2020) Noviswitch 21100 white paper. [Online]. Available: https://noviflow.com/wp-content/uploads/2019/11/NoviSwitch-21100-Datasheet-400_V5.pdf

[13] J. Liu, H. Xu, G. Zhao, C. Qian, X. Fan, X. Yang, and H. Huang, "Incremental server deployment for software-defined nfv-enabled networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 248–261, 2020.

[14] G. Zhao, H. Xu, S. Chen, L. Huang, and P. Wang, "Joint optimization of flow table and group table for default paths in sdns," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1837–1850, 2018.

[15] Z. Xu, W. Liang, A. Galis, Y. Ma, Q. Xia, and W. Xu, "Throughput optimization for admitting nfv-enabled requests in cloud networks," *Comput. Networks*, vol. 143, pp. 15–29, 2018.

[16] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest path and maximum flow problems under service function chaining constraints," in *2018 IEEE Conference on Computer Communications, INFOCOM 2018*. IEEE, 2018, pp. 2132–2140.

[17] NTT. (2021) NTT Hong Kong Financial Data Center. [Online]. Available: https://datacenter.hello.global.ntt/sites/default/files/apac_hongkong_brochure_jun2020_.pdf

[18] L. Fang, F. M. Chiussi, D. Bansal, V. Gill, T. Lin, J. Cox, and G. R. Ratterree, "Hierarchical SDN for the hyper-scale, hyper-elastic data center and cloud," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR '15*. ACM, 2015, pp. 7:1–7:13.

[19] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2011, pp. 254–265.

[20] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: weighted cost multipathing for improved fairness in

data centers," in *Ninth Eurosys Conference 2014, EuroSys 2014*. ACM, 2014, pp. 5:1–5:14.

[21] H. Xu, H. Huang, S. Chen, G. Zhao, and L. Huang, "Achieving high scalability through hybrid switching in software-defined networking," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 618–632, 2018.

[22] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014*. USENIX Association, 2014, pp. 543–546.

[23] Q. Zhang, Y. Xiao, F. Liu, J. C. S. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017*. IEEE Computer Society, 2017, pp. 731–741.

[24] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. M. Jr., "Metron: NFV service chains at the true speed of the underlying hardware," in *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018*. USENIX Association, 2018, pp. 171–186.

[25] T. Barbette, C. Soldani, R. Gaillard, and L. Mathy, "Building a chain of high-speed vnfs in no time: Invited paper," in *IEEE 19th International Conference on High Performance Switching and Routing, HPSR 2018*. IEEE, 2018, pp. 1–8.

[26] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012*. USENIX Association, 2012, pp. 323–336.

[27] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: A software-defined framework for developing, deploying, and managing network functions," in *Proceedings of the ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 511–524.

[28] B. Anwer, T. Benson, N. Feamster, and D. Levin, "Programming slick network functions," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR '15*. ACM, 2015, pp. 14:1–14:13.

[29] G. P. Katsikas, M. Enguehard, M. Kuzniar, G. Q. M. Jr., and D. Kostic, "SNF: synthesizing high performance NFV service chains," *PeerJ Comput. Sci.*, vol. 2, p. e98, 2016.

[30] O. N. Foundation *et al.*, "Openflow version 1.3.5," 2014.

[31] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[32] Pica8. (2014) Pica8 p3297 switches. [Online]. Available: https://www.pica8.com/wp-content/uploads/pica8-datasheet-48x1gbe-p3297.pdf

[33] Barefoot-networks. (2014) Barefoot tofino switches. [Online]. Available: https://www.barefootnetworks.com

[34] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*. IEEE, 2015, pp. 98–106.

[35] T. Lukovszki, M. Rost, and S. Schmid, "It's a match!: Near-optimal and incremental middlebox deployment," *Comput. Commun. Rev.*, vol. 46, no. 1, pp. 30–36, 2016.

[36] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: enabling network function parallelism in NFV," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017*. ACM, 2017, pp. 43–56.

[37] M. C. Luizelli, D. Raz, and Y. Sa'ar, "Optimizing NFV chain deployment through minimizing the cost of virtual switching," in *2018 IEEE Conference on Computer Communications, INFOCOM 2018*. IEEE, 2018, pp. 2150–2158.

[38] P4.org. (2021) $P4_{16}$-language-specification. [Online]. Available: https://p4lang.github.io/p4-spec/docs/P4-16-v1.2.1.html

[39] Z. Meng, J. Bi, H. Wang, C. Sun, and H. Hu, "Coco: Compact and optimized consolidation of modularized service function chains in NFV," in *2018 IEEE International Conference on Communications, ICC 2018*. IEEE, 2018, pp. 1–7.

[40] OVS. (2018) Open vswitch: open virtual switch. [Online]. Available: http://openvswitch.org/

[41] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC implementation of the AES sboxes," in *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2271. Springer, 2002, pp. 67–78.

[42] L. Guo, J. Z. T. Pang, and A. Walid, "Joint placement and routing of network function chains in data centers," in *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*. IEEE, 2018, pp. 612–620.

[43] C. Tian, A. Munir, A. X. Liu, J. Yang, and Y. Zhao, "Openfunction: An extensible data plane abstraction protocol for platform-independent software-defined middleboxes," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1488–1501, 2018.

[44] D. E. Sarmiento, A. Lebre, L. Nussbaum, and A. Chari, "Decentralized SDN control plane for a distributed cloud-edge infrastructure: A survey," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 1, pp. 256–281, 2021.

[45] Z. N. Abdullah, I. Ahmad, and I. Hussain, "Segment routing in software defined networks: A survey," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 1, pp. 464–486, 2019.

[46] IETF. (2020) Ipv6 segment routing header (SRH). [Online]. Available: https://www.rfc-editor.org/rfc/rfc8754.txt

[47] E. Q. Martins and M. M. Pascoal, "A new implementation of yens ranking loopless paths algorithm," *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 2, pp. 121–133, 2003.

[48] N. McKeown, T. E. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "Openflow: enabling innovation in campus networks," *Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[49] H. Xu, Z. Yu, C. Qian, X. Li, Z. Liu, and L. Huang, "Minimizing flow statistics collection cost using wildcard-based requests in sdns," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3587–3601, 2017.

[50] X. Yu, H. Xu, D. Yao, H. Wang, and L. Huang, "Countmax: A lightweight and cooperative sketch measurement for software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2774–2786, 2018.

[51] S. A. Amiri, K. Foerster, R. Jacob, and S. Schmid, "Charting the algorithmic complexity of waypoint routing," *Comput. Commun. Rev.*, vol. 48, no. 1, pp. 42–48, 2018.

[52] I. I. CPLEX, "V12. 1: Users manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[53] P. Raghavan and C. D. Thompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Comb.*, vol. 7, no. 4, pp. 365–374, 1987.

[54] J. George *et al.* (2018) Qperf. [Online]. Available: https://github.com/linux-rdma/qperf

[55] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016*. USENIX Association, 2016, pp. 537–549.

[56] T. Toivola *et al.* (2018) vnstat. [Online]. Available: https://humdi.net/vnstat/

[57] P4.org. (2015) behavioral-model version 2. [Online]. Available: https://github.com/p4lang/behavioral-model

[58] N. T. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. ACM, 2002, pp. 133–145.

[59] P4·org. (2021) P4-compiler. [Online]. Available: https://github.com/p4lang/p4c

[60] Google. (2021) grpc. [Online]. Available: https://www.grpc.io/

[61] G. Chen, Y. Lu, Y. Meng, B. Li, K. Tan, D. Pei, P. Cheng, L. Luo, Y. Xiong, X. Wang, and Y. Zhao, "Fast and cautious: Leveraging multi-path diversity for transport loss recovery in data centers," in *2016 USENIX Annual Technical Conference, USENIX ATC 2016*. USENIX Association, 2016, pp. 29–42.

[62] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, and P. Cheng, "Clicknp: Highly flexible and high-performance network processing with reconfigurable hardware," in *Proceedings of the ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 1–14.

Dear Editor and Reviewers:

We sincerely appreciate the detailed comments and constructive suggestions by the editor and anonymous reviewers. The manuscript (No. TNET-2020-00607) has been revised according to the comments. Below is our point by point response to all review comments. We hope that the revised manuscript has adequately addressed all concerns from the reviewers.

Sincerely yours

Hongli Xu, Gongming Zhao

Summary of Important Response to Comments:

1) We have added P4-based SAFE-ME implementation and given the experimental results. Specifically, we have implemented SAFE-ME with the P4 switches and described its design in Section 4.4. Moreover, we have replaced the entire content of former Sections 6.2 and 6.3 with a new Section 6.2 containing the detailed description of the P4-based SAFE-ME experiments. The experimental results also strongly support the correctness, feasibility and scalability of SAFE-ME, in response to Comment 1.18.
2) We have improved the description of both Section 3 and Section 4 to enhance the design of SAFE-ME, in response to Comments 1.15, 1.16, 1.19 and 2.1. We have also given detailed illustration for our former OVS-based SAFE-ME implementation in response to Comments 1.20, 1.21, 1.23, 1.24 and 1.25.
3) We have reorganized the introduction section (Section 1) and revised the background section (Section 2) to enhance our motivation, in response to Comments 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.9 and 1.12.
4) We have enhanced the description of the lemmas and the NP-hardness proof in Section 5.2, in response to Comments 2.2 and 2.3.
5) We have done an in-depth study of related works and added some new references in Section 2, in response to Comments 1.10, 1.11, 1.14 and 1.22.
6) We have added a paragraph at the beginning of each section to briefly introduce the overview of that section, in response to Comments 1.8, 1.13 and 1.17.
7) We have updated the content of the Github repository to share the source code, packet trace data and documentation of our SAFE-ME experiments, in response to Comments 2.5 and 2.6.
8) We have carefully checked the manuscript and modified some typo/grammar issues in the revised manuscript.

In the following response sections, we use the red color font to show the revised content and **black color, bold style font** to emphasize the critical information.

Response to Review #1:

[General Comment]

Typically, between journal revisions, when you receive a bunch of comments from a set of reviewers you first address them carefully and then compose a revision document which states how you addressed all these comments and what changes you have done with respect to the previous version of your article. This way, the next reviewer knows what comments where received by previous reviewers and how the authors addressed these comments. I don't see such a revision document, but even worse, I don't see any effort to address my comments from last year's revision process.

Response: We fully agree with your rich, instructive and constructive comments above. We have made some major revisions earnestly and rigorously in response to all comments. We summarize some important responses as below:

1) We have enhanced the implementation and experiments of our SAFE-ME scheme, along with thorough and detailed explanations.
   1.1) To accommodate the reviewer's suggestions, we have implemented SAFE-ME on P4 switches. We have given the design of P4-based SAFE-ME in a new Section 4.4, and replaced the whole content of former Section 6.2 and 6.3 with a new Section 6.2 containing the implementation and experimental results of P4-based SAFE-ME, in response to Comment 1.18.
   1.2) We have given the detailed illustration of the SAFE-ME implementation with OVS, in response to Comment 1.21. We have also enhanced the description of the OVS experiments, in response to Comments 1.20, 1.23, 1.24 and 1.25.
2) We have reorganized the introduction section (Section 1) and revised the background section (Section 2) to enhance our motivation.
   2.1) We have explained the importance and advantages of saving the number of in-used flow entries, in response to Comments 1.2 and 1.9.
   2.2) We have enhanced the description of the scalability challenge in middlebox networks, from the aspects of control plane and data plane, in response to Comments 1.3, 1.4, 1.5 and 1.6.
   2.3) We have reorganized the content of Section 1. We have introduced the structure of the revised Section 1 to make the challenges and our motivation clearer, in response to Comment 1.7.
   2.4) We have enhanced the research and comparison among the traditional default path solutions and the SDN-based per-request routing solutions, in response to Comment 1.12.
3) We have enhanced the description and explanation of SAFE-ME, in response to Comments 1.15, 1.16 and 1.19.
4) We have added a paragraph at the beginning of each section to briefly introduce the content of that section, in response to Comments 1.8, 1.13 and 1.17.
5) We have enhanced the study of state-of-the-art works and added some new references. Also, we have revised some inappropriate descriptions and grammatical errors, in response to Comments 1.10, 1.11, 1.14 and 1.22.

[Comments for Section 1]

Comment 1.1 *"switches forward the data packets by matching rules on the TCAM-based forwarding table" --> not all SDN switch tables are TCAM based, so be careful here*

Response: Sorry for the inappropriate description. We have improved the description in the 2nd paragraph of Section 1 as below.

Under the SDN framework, the switch often forwards data packets by matching the rules in the TCAM-based forwarding table to achieve fast rule lookup or wildcard rule matching.

Comment 1.2 *"Considering the process speed, price and power consumption of the switch, most of today's commodity switches only support just 2-20K entries [5] [6]"*

I reviewed your previous TNET submission last year as well and I had explained you my concerns on this sentence last year. Specifically, these are my words from last year's review:

*"This is partially true. A single table may indeed be quite small (e.g., 4000 entries), but typically OF switches may possess **tens of tables**. Our old (since 2014) NoviFlow switch provides 55 tables, each with 4096 entries. I assume that recent models may potentially provide much more capacity. You should fix this in the introduction."*

You did not address my comment.

Response: We agree with the reviewer and believe that continuous advances in switch design will undoubtedly lead to more and more entries in a flow table. However, **the price and power requirements for TCAMs may limit the size of TCAMs.** According to [1], TCAMs are 400×more expensive and consume 100× more power per Mbit than the RAM-based storage in switches. Moreover, the lookup speed and insertion speed are related to the size of TCAMs. For example, based on the TCAM power model [2], a single search on a 1 megabit (Mb) TCAM chip takes 17.8 nanojoules (nJ) and 2.1 nanoseconds (ns), while the same search on a 36 Mb TCAM chip takes 483.4 nJ and 46.9ns, respectively. Thus, it is a significant tradeoff between the size of a TCAM chip and its operation performance (e.g., power consumption and search speed).

With the increase of the switch memory, some high-end SDN switches (e.g., Noviswitch 21100 [3]) adopt RAM-based flow tables to support a large number of exact match entries (e.g., up to 6 million). However, RAM-based flow tables may significantly increase the lookup latency [4]. In addition, **a large number of flow entries may cause a long delay in entry modification/search** [5].

In fact, the limited number of flow entries should be shared by several functions (e.g., routing/performance/measurement/security) implemented on the same chip. For example, if the controller expects some flows to be processed by middleboxes, there should install additional rules for these flows on switches [6]. Thus, the available number of flow entries for flow routing is less than its flow-table size.

Hence, we believe that **managing the usage of flow entries will benefit energy consumptions or improve the scalability of the network.**

We have enhanced the reason of flow table size constraints in the 2nd paragraph of Section 1 in the revised manuscript.

However, TCAMs are 400× more expensive and consume 100× more power per Mbit than the RAM-based storage on switches [5]. Considering the processing speed, price and power consumption of the switch, most of today's commodity switches are equipped with a limited number of TCAM-based entries. Thus, it is essential to save the number of TCAM-based entries used for routing [8] [9].

[1] N. P. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in Proceedings of the third workshop on Hot topics in software defined networking, HotSDN '14. ACM, 2014, pp. 175–180.

[2] B. Agrawal and T. Sherwood, "Modeling TCAM power for next generation network devices," in 2006 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS. IEEE Computer Society, 2006, pp. 120–129.

[3] https://noviflow.com/wp-content/uploads/2019/11/NoviSwitch-21100-Datasheet-400_V5.pdf

[4] G. Zhao, H. Xu, J. Fan, L. Huang, and C. Qiao, "Achieving fine-grained flow management through hybrid rule placement in sdns," IEEE Trans. Parallel Distributed Syst., vol. 32, no. 3, pp. 728–742, 2021.

[5] H. Song, S. Guo, P. Li, and G. Liu, "FCNR: fast and consistent network reconfiguration with low latency for SDN," Comput. Networks, vol. 193, p. 108113, 2021.

[6] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," ACM SIGCOMM computer communication review, vol. 43, no. 4, pp. 27–38, 2013.

Comment 1.3 *"However, with more flows arriving at the network, it will lead to serious per-flow communication/computation overhead on the control plane" This is another comment you did not address. Here is my comment on this line from last year's review:*

"Actually, this is not how SDN-based networks work in practice. Typically, the ingress zone of a network may involve classifiers that redirect specific traffic types (e.g., HTTP/HTTPS) to specific pools of servers (i.e., web servers). This way, the SDN switches towards that pool of servers may have pre-installed rules, which match HTTP traffic, thus no or very little controller intervention is required."

If you look at real SDN deployments, you will find that nobody in practice sends every new flow to the controller. This kills network performance for no reason. A network operator knows that a given SFC is deployed, thus he/she has pre-installed rules to steer traffic across the NF of this SFC.

Response: We agree with the reviewer that per-flow routing is impractical for large-scale networks. Existing SFC routing solutions often install rules for flows with the granularity of ingress-egress switch pairs [7][8]. The large-scale network may contain thousands of ingress/egress switches, thus it may require millions of forwarding entries on a switch in the worst case. In this case, when encountering network failures or network performance degradation, **the controller needs to update many flow entries for network reconfiguration, which may result in high control overhead** [9][10]. Thus, we believe that the middlebox network faces the control plane scalability challenge.

We have enhanced the description of control plane scalability in the 2nd paragraph of Section 1 in the revised manuscript.

Under the SDN framework, the controller is responsible for managing the whole network, e.g., monitoring the network status and determining/updating the SFC routing [9]. Thus, in a large-scale network, when encountering network failures or network performance degradation, the controller needs to update many flow entries for network reconfiguration, which may result in high control overhead [9][10].

[7] Z. A. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using SDN," in ACM SIGCOMM 2013 Conference, SIGCOMM 2013. ACM, 2013, pp. 27–38.
[8] L. Guo, J. Z. F. Pang, and A. Walid, "Dynamic service function chaining in sdn-enabled networks with middleboxes," in 24th IEEE International Conference on Network Protocols, ICNP 2016. IEEE Computer Society, 2016, pp. 1–10.
[9] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, and Y. Sun, "Minimizing controller response time through flow redirecting in sdns," IEEE/ACM Trans. Netw., vol. 26, no. 1, pp. 562–575, 2018.
[10] M. A. Togou, D. A. Chekired, L. Khoukhi and G. Muntean, "A Hierarchical Distributed Control Plane for Path Computation Scalability in Large Scale Software-Defined Networks," in IEEE Transactions on Network and Service Management, vol. 16, no. 3, pp. 1019-1031, Sept. 2019, doi: 10.1109/TNSM.2019.2913771.

Comment 1.4 *"Moreover, SFC routing update, especially in a large-scale network, also encounters serious control overhead." Any reference for this?*
Response: The previous work [10] points out: "solving the synchronization problem in large-scale SDNs, particularly when topology changes are frequent or even moderate, puts computational pressure on the controllers and introduces high overhead due to the large number of control messages to be exchanged. This impacts the efficiency of path computation, which in turn affects the performance of various SDN applications such as traffic engineering solutions". So, it has been proved that **the control plane will face high overhead in large-scale networks.**

We have added reference [10] to the description of control plane scalability in the 2nd paragraph of Section 1.

Thus, in a large-scale network, when encountering network failures or network performance degradation, the controller needs to update many flow entries for network reconfiguration, which may result in high control overhead [9] [10].

Comment 1.5 *"If a network contains several thousands of ingress/egress switches, there are millions of ingress-egress switch pairs." This sounds too dramatic and philosophical. What is the scale of such a network? Do you have a particular example in mind?*

Response: Thanks for the concern. Existing literature has mentioned the "Hyper-Scale Data Centers". As described in [11]: "By hyper-scale, we mean that the underlay network infrastructure has to scale to support tens of millions of physical endpoints (i.e., bare metal or virtualized servers) and middleboxes (i.e., physical and virtualized network functions and appliances)." We take NTT Financial Data Center in Hong Kong [12] as an example. It contains more than 7000 racks in two buildings reported in June, 2020. Thus, it is reasonable to believe that the hyper-scale data centers will have several thousand ingress and egress switches.

We have added these references in the 4th paragraph of Section 1 in the revised manuscript.

If a network contains several thousands of ingress/egress switches (e.g., NTT Hong Kong Financial Data Center contains more than 7000 racks [12]), there are millions of ingress-egress switch pairs [11] [12].

[11] L. Fang, F. M. Chiussi, D. Bansal, V. Gill, T. Lin, J. Cox, and G. R. Ratterree, "Hierarchical SDN for the hyper-scale, hyper-elastic data center and cloud," in Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR '15. ACM, 2015, pp. 7:1–7:13.
[12] NTT. (2021) NTT Hong Kong Financial Data Center. [Online]. Available: https://datacenter.hello.global.ntt/sites/default/files/apac_hongkong_brochure_jun2020_.pdf

Comment 1.6 *"(no matter using proactive mode (e.g., [2]) or using reactive mode (e.g., [7]))"*
Nested parentheses are a bad practice. Also, you did not explain what proactive and reactive modes are.
Response: Sorry for the inappropriate description. There are two forwarding modes in SDNs: **proactive mode** [7] and **reactive mode** [8]. Specifically, the proactive mode usually installs rules for flows in advance based on traffic estimation [7]. The reactive mode mainly relies on the controller to perform real-time routing calculation and rule installation for new arrival flows [8]. Both proactive mode (e.g., SIMPLE [7]) and reactive mode (e.g., PDA [8]) have been widely adopted in real-world networks.

We have improved the description of proactive mode and reactive mode in the 4th paragraph of Section 1 in the revised manuscript as follows.

Second, existing proactive-based solutions (e.g., [7]) usually install rules for flows in advance based on traffic estimation. Thus, they cannot deal with bursty traffic well and significantly decrease the users' QoS [13]. Third, existing reactive-based solutions (e.g.,[8]) mainly rely on the controller to perform real-time routing calculation and rule installation for new arrival flows, which may lead to serious per-flow communication/computation overhead on the control plane [9].

[13] S. Kandula, S. Sengupta, A. G. Greenberg, P. Patel, and R. Chaiken, “The nature of data center traffic: measurements & analysis,” in Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference, IMC 2009. ACM, 2009, pp. 202–208.

Comment 1.7 *Last year I proposed some general directions on how to structure your Introduction in order to be more convincing. Here are my words:*

"I am not thrilled with the way your introduced your work (Section 1). Some of the challenges you mention do not seem to be that hard to solve (or may already be solved). Maybe a better way of introducing these challenges is through the state of the art solutions. You may put a graph that shows how "broken" state of the art solutions are and explain why that happens. With a nice graph, you may convince the reader."

You did not attempt to address any of my comments and this is a problem.

Response: Sorry for the confusing description of our motivation and challenges. SFC routing brings more critical scalability challenges on both data plane and control plane in middlebox networks. To accommodate these suggestions, we have introduced these challenges of SFC routing through the state-of-the-art solutions. Moreover, we have given Table 1 to show how "broken" state-of-the-art solutions are.

Specifically, Section 1 of the revised manuscript has been organized as follows. Paragraph 1 gives the introduction of middlebox networks. Paragraph 2 analyzes the two major challenges in SDN-based middlebox networks: control plane scalability and data plane scalability. We show the inapplicability of traditional default path solutions for SFC routing in Paragraph 3. Paragraph 4 illustrates the limitations of the previous SFC routing solutions. In Paragraph 5, we give a brief description of the proposed SAFE-ME system. Paragraph 6 gives the paper organization structure.

[Comments for Section 2]

Comment 1.8 *There is no text between Section 2 and Section 2.1, please introduce Section 2.1.*

Response: We have added the following text before Section 2.1 to introduce Section 2.

In this section, we first introduce the traditional network forwarding mechanism based on the default path and prove its inapplicability for SFC routing in Section 2.1. Then we review the existing SDN-based SFC routing solutions and analyze their limitations in Section 2.2.

Comment 1.9 *"Even if there is only one MB in the network (or one SFC requirement per switch pair),it may require 1M entries on a switch in the worst case, which may violate today’s switch capabilities [6]." See my comments from last year:*

"Pardon me, but you use a 6 years old reference to argue about SDN switches capacity. You should be looking into modern (2018 and later) datacenter switches (e.g., TOFINO-2) and more recent works. Even today's Network cards, such as those provided by Mellanox,

*offer capacity in the order of several *millions* of flow rules. This is another piece of evidence that the way you argue is not very accurate."*

Again no effort to address this comment.

Response: We agree with the reviewer and believe that continuous advances in switch design will undoubtedly lead to more and more entries in a flow table. However, the price and power requirements for TCAMs may limit the size of TCAMs as illustrated in Response to Comment 1.2. Although some SDN switches (e.g., Noviswitch 2000 Series [14]) adopt RAM-based flow tables to support a large number of flow entries, it may significantly increase the lookup latency [4].

Even if we assume that some high-end switches can contain over 1 million TCAM-based entries in the future, installing too many entries may result in huge energy consumption and long update delay. For example, the rule installation speed is about 0.517 ms/rule on P4 software switches by our test in Section 6.2. Thus, we believe it is meaningful to reduce the occupation of flow entries on switches.

We have modified the description in the 1st paragraph of Section 2.2 as below.

Even if there is only one MB in the network (or one SFC requirement per switch pair), it may require 1M entries on a switch in the worst case, which may violate today's switch capabilities or result in huge energy consumption and long update delay [8].

[14] NoviFlow. (2021). NoviWare NoviSwitches 2000 Series. [Online]. Available: https://noviflow.com/noviswitch/.

Comment 1.10 *"to simplify the SFC processing in middlebox networks by constructing consolidated platform [14] [15] [16]." Another comment from last year: [14] [15] [16] these are only a few efforts that propose NFV consolidation. You should carefully study the following efforts too:*

[1] Barbette, T., Soldani, C., Gaillard, R., and Mathy, L. Building a chain of high-speed VNFs in no time. In Proceedings of the IEEE International Conference on High Performance Switching and Routing (2018), HPSR'18.
https://www.tombarbette.be/wp-content/uploads/2018/10/barbette.pdf.

[2] Katsikas, G. P., Enguehard, M., Kuźniar, M., Maguire J r., G. Q., and Kostić, D. SNF: Synthesizing high performance NFV service chains. PeerJ Computer Science 2 (Nov. 2016), e98. http://dx.doi.org/10.7717/peerj-cs.98.

[3] Anwer B., Benson T., Feamster N., Levin D. 2015. Programming slick network functions. In Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research, SOSR'15. New York, N Y, USA: ACM, 14:1–14:13.

Again no effort to address this comment.

Response: Thanks for the suggestions. We have carefully studied these related works that the reviewer gives above. These works focus on the development of consolidated platforms for simplifying the SFC processing. **SAFE-ME addresses the complementary problem on how to optimally determine the routing selection for all requests and the rule installment at all switches.** Thus, our proposed solution can be combined with the consolidated platform to

achieve better network performance. We give the detailed study of the above three related works.

A) MiddleClick [16] aims to run a pipeline of VNFs with a high level of parallelism to handle many flows. It analyzes the various classification needs of the VNFs in the chain and synthesizes a single, non-redundant classifier. This consolidation scheme can avoid multiple reclassification of packets.

B) Slick [19] allows the programmer to write a single application that describes a sequence of processing elements for a given part of flow space, and designs a Slick runtime to manage the details of how those elements are replicated, consolidated, and placed throughout the network.

C) SNF [20] focuses on synthesizing chains of NFs to provide minimal I/O interactions with the NFV platform, applying single-read-single-write operations on the packets, and early-discarding irrelevant traffic classes.

Thus, we conclude that the above consolidated platforms can simplify the SFC processing, but they still require a large number of forwarding entries if without proper routing strategies. In fact, our proposed SAFE-ME solution tries to optimize the route selection for middlebox networks, and can be combined with the consolidated platform.

We have revised the 2nd paragraph of Section 2.2 as below to add and classify these works.

To reduce the TCAM table cost and control overhead, some research attempts to simplify the SFC processing in middlebox networks by constructing a consolidated platform [15] [16] [17] [18] [19][20]. For example, CoMB [17] is a network function consolidation platform, where a flow/request can be processed by all required network functions on a single hardware platform, thus simplifying the SFC routing. Metron [15], MiddleClick [16], and OpenBox [18] also adopt the consolidation conception so as to merge similar packet processing elements into one. Slick [19] and SNF [20] combine the consolidated platform with NF placement and SFC rule decision making. In Fig.1, they may integrate NF_1 and NF_2 into one mixed NF. All traffic will traverse the mixed NF and be processed automatically by corresponding functions. Though the consolidated platform simplifies the SFC processing, as described in the above paragraph, it still requires a large number of forwarding entries if without proper routing strategies. In fact, our proposed solution tries to optimize the route selection for middlebox networks, and can be combined with the consolidated platform, which will be discussed in Section 4.5.

[15] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. M. Jr., “Metron: NFV service chains at the true speed of the underlying hardware,” in 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018. USENIX Association, 2018, pp. 171–186.

[16] T. Barbette, C. Soldani, R. Gaillard, and L. Mathy, “Building a chain of high-speed vnfs in no time: Invited paper,” in IEEE 19th International Conference on High Performance Switching and Routing, HPSR 2018. IEEE, 2018, pp. 1–8.

[17] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, “Design and implementation of a consolidated middlebox architecture,” in Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012. USENIX Association, 2012, pp. 323–336.

[18] A. Bremler-Barr, Y. Harchol, and D. Hay, “Openbox: A software-defined framework for developing, deploying, and managing network functions,” in Proceedings of the ACM SIGCOMM 2016 Conference. ACM, 2016, pp. 511–524.
[19] B. Anwer, T. Benson, N. Feamster, and D. Levin, “Programming slick network functions,” in Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR ’15. ACM, 2015, pp. 14:1–14:13.
[20] G. P. Katsikas, M. Enguehard, M. Kuzniar, G. Q. M. Jr., and D. Kostic, “SNF: synthesizing high performance NFV service chains,” PeerJ Comput. Sci., vol. 2, p. e98, 2016.

Comment 1.11 *"TABLE 1: Comparison of the advantages and disadvantages of existing solutions." This table is not convincing enough as it provides only 3 references. Normally, you should do the grouping of research efforts in such a way that you include several references per entry. Your table as it s now gives the impression that you compare your work with only 3 reference efforts. Moreover, last year your table had one more entry titled "Consolidated Platform", why did you remove it? We seek for more content in this table, not less.*

Response: Thanks for the suggestion. We have studied many related works and divided them into two categories: **the traditional default path solution** and **the SDN-based per-request routing solution**, as illustrated in Section 2.

The traditional default path solutions [21][22][23][24] can achieve better system scalability and deal with traffic dynamics in traditional networks. However, they cannot be applied directly in middlebox networks because it cannot deal with routing loops. Besides, the flows with the same egress switch (or destination) will be processed in the same way by default paths, but these flows may require different SFCs.

The existing SDN-based per-request SFC routing solutions [25][26][27][28] often install a large number of rules for the middlebox routing. Thus, when encountering network failures or network performance degradation, the control plane needs to update many entries for network reconfiguration, which may result in high control overhead. Moreover, a large number of flow entries may significantly increase the entry lookup latency in the data plane, leading to high forwarding delay.

We observe that all existing methods can only address partial challenges of the SFC routing in middlebox networks. To conquer the above challenges, including routing loops, traffic dynamics, and scalability requirements, we design our SAFE-ME scheme.

Note that, as illustrated in Response to comment 1.10, our proposed solution tries to optimize the route selection for middlebox networks, and can be combined with the consolidated platform. Specifically, the NFV consolidated-based platforms [15][16][17][18][19][20] can simplify the SFC processing, but they still require a large number of forwarding entries if without proper routing strategies. That means, **the NFV consolidated-based platforms are not our competitors, but collaborators.** Thus, we have removed "Consolidated Platform" in Table 1.

We have studied more related works and added them into Table 1 in the revised manuscript.

Schemes	Number of Rules	Control Overhead	Satisfy SFC Policy	Network Performance
Traditional Default Path (*e.g.*, [21][22][23][24])	Few	Low	No	Low
Per-request Routing (*e.g.*, [25][26][27][28])	Many	High	Yes	High
Our Scheme	**Few**	**Low**	**Yes**	**High**

TABLE 1: Comparison of the advantages and disadvantages of existing solutions.

[21] G. Zhao, H. Xu, S. Chen, L. Huang, and P. Wang, "Joint optimization of flow table and group table for default paths in sdns," IEEE/ACM Trans. Netw., vol. 26, no. 4, pp. 1837–1850, 2018.

[22] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. ACM, 2011, pp. 254–265.

[23] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: weighted cost multipathing for improved fairness in data centers," in Ninth Eurosys Conference 2014, EuroSys 2014. ACM, 2014, pp. 5:1–5:14.

[24] H. Xu, H. Huang, S. Chen, G. Zhao, and L. Huang, "Achieving high scalability through hybrid switching in software-defined networking," IEEE/ACM Trans. Netw., vol. 26, no. 1, pp. 618–632, 2018.

[25] Z. A. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simplefying middlebox policy enforcement using SDN," in ACM SIGCOMM 2013 Conference, SIGCOMM 2013. ACM, 2013, pp. 27–38.

[26] L. Guo, J. Z. F. Pang, and A. Walid, "Dynamic service function chaining in sdn-enabled networks with middleboxes," in 24th IEEE International onference on Network Protocols, ICNP 2016. IEEE Computer Society, 016, pp. 1–10.

[27] Z. Xu, W. Liang, A. Galis, Y. Ma, Q. Xia, and W. Xu, "Throughput optimization for admitting nfv-enabled requests in cloud networks," Comput. Networks, vol. 143, pp. 15–29, 2018.

[28] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest path and maximum flow problems under service function chaining constraints," in 2018 IEEE Conference on Computer Communications, INFOCOM 2018. IEEE, 2018, pp. 2132–2140.

Comment 1.12 *"From Table 1, we observe that all existing methods can only address partial challenges of the SFC routing in middlebox networks. In other words, it seems that none of them can satisfy SFC requirements with better routing performance, lower control overhead and fewer flow rules consumption. Thus, in this paper, we design an efficient architecture for SFC routing (i.e., SAFE-ME) so as to satisfy the above requirements."*

Here you use Table 1 as a way to convince the reader, but as I said above, Table 1 does not provide enough information. You should further study the related work. I suggested the very same thing last year, but you did not take it into account.

Response: Thanks for the suggestions. We have added more related works in Table 1 to give sufficient examples and competitors for readers. Now we have divided the related works into two categories and given a total of 14 references as illustrated in Response to Comment 1.11.

[Comments for Section 3]

Comment 1.13 *There is no text between Section 3 and Section 3.1, please introduce Section 3.1.*

Response: We have added an introductive paragraph before Section 3.1 as below:

To solve the above challenges in middlebox networks, we propose the scalable and flexible middlebox policy enforcement system (SAFE-ME). We present the overview of SAFE-ME in Section 3.1, describe the packet processing procedure in Section 3.2, and give an example to illustrate SAFE-ME in Section 3.3.

Comment 1.14 *"should be traversed a SFC" --> "should traverse an SFC"*

Response: We have revised this sentence in the 1st paragraph of Section 3.3 as below:

For example, in Fig. 4, the administrator specifies that traffic from subnet 10.1.1.0/24 should traverse an SFC.

Comment 1.15 *"That means, we only need one special SFC entry on the ingress switch for this request, and the other entries are shared by different requests. Consequently, SAFE-ME will greatly reduce the use of rules and the control overhead." This very same sentence was commented by me last year, as follows:*

"In order to state that SAFE-ME will greatly reduce the use of rules and the control overhead, you need to show us how existing state of the art solutions solve the same problem. This way we will see how many rules SAFE-ME saves compared to the state of the art. You should update your example and Figure 4 accordingly, including at least one state of the art work."

Response: Sorry for the unclear description of the specific implementation of existing solutions. SAFE-ME divides switch tables into three parts with different roles, called SFC Table, NF Table, and Flow Table, respectively. Leveraging the newly designed Default Path Construction (DPC) module, SAFE-ME computes the default paths from each switch to each egress switch or each NF and stores these information in NF/Flow Tables proactively. In this way, **we only need to install one special SFC entry on the ingress switch for each new arrival request**, and the other entries can be shared by different requests. We should note that, although SAFE-ME proactively installs some default rules, the entire network and forwarding rules can be managed and dynamically modified by the SDN controller. Thus, SAFE-ME can respond quickly to network failures when encountering traffic dynamics or link/switch/NF failures.

Existing non-SDN-based solutions cannot dynamically adjust routing rules according to the current network status and can't adapt to the network dynamics well. Thus, in this paper, we mainly compare SAFE-ME with SDN-based solutions. Specifically, to realize the manageability of the middlebox network, existing **SDN-based solutions usually adopt a per-request routing strategy.** As illustrated in Section 2.2, per-request routing solutions may consume several special entries for each new arrival request. Thus, we say that **SAFE-ME will greatly reduce the use of rules and the control overhead**.

Since we have discussed the existing works in Section 2.2, and Section 3.3 is used for better understanding of the packet processing in SAFE-ME through the example shown in Fig. 4, plus, considering the limitation of space, we omit the detailed discussion about per-request solutions in Section 3.3.

We have enhanced the description at the end of Section 3.3 in the revised manuscript.

That means, by leveraging the design of the NF/Flow Tables, SAFE-ME can proactively compute the default paths from each switch to each egress switch or each NF and stores this information in NF/Flow Tables proactively. In this way, we only need to install one special SFC entry on the ingress switch for each new arrival request, and the other entries can be shared by different requests. On the contrary, as illustrated in Section 2.2, per-request routing solutions may consume several special entries for each new arrival request. Thus, SAFE-ME will greatly reduce the use of rules and the control overhead.

Comment 1.16 *Apart from this, you did not mention how the return traffic is processed by SAFE-ME. Presumably, the existing rules are not enough. Imagine that all packets at the return path must traverse the same set of NFs and update Figure 4 and the discussion accordingly." Again no effort to address this comment.*
Response: SAFE-ME focuses on the installment of flow entries for efficient SFC processing. In fact, the process of the return traffic in the SAFE-ME scheme is the same as that of the incoming traffic, as described in Sections 3.2 and 3.3. If we add the flow entries for the return traffic in Fig. 4, it will become much complex and hard for understanding. Thus, we only depict the routing tables and describe the process for the incoming traffic for simplicity.

We have added some description about return traffic in the last paragraph of Section 3.3 in the revised manuscript.

Note that, the process for the return traffic is similar as the incoming traffic. Thus, we only depict the routing tables and describe the process for the incoming traffic for simplicity.

[Comments for Section 4]
Comment 1.17 *There is no text between Section 4 and Section 4.1, please introduce Section 4.1.*
Response: We have added the following introductive paragraph before Section 4.1 in the revised manuscript.

As described in Section 3.2, multiple entry tables and tag operations are two core issues in the data plane that may affect the forwarding performance. Thus, this section mainly designs the table pipeline process and tag operations (e.g., tag embedding and deleting) on both OpenFlow and P4 [30] switches. Through these operations, SAFE-ME can forward requests using fewer rules and lower control overhead while satisfying SFC policies.

Comment 1.18 *Figure 6 shows that your tagging approach requires 41 bits which can be placed right before the IP header. Then you write: "For other SDN switches, we can leverage either VLAN tags, MPLS labels, or other unused fields in the IP header to embed the SFC policy [2] [12]" Both Figure 6 and this sentence are still here, despite the fact I made these comments last year:*

*"Figure 6 states that you use a custom header before the IP header to accommodate the SFC information. However, OpenFlow-based switches *won't be able to parse and understand such custom header*. Only P4 switches could do that (as a programmer can program the parser). In Section 4.4 you state that: "For other SDN switches, we can leverage either VLAN tags, MPLS labels, or other unused fields in the IP header to embed the SFC information [1][14]."*

This is a major issue of your design as VLAN tags and MPLS labels are not unused fields, but rather very commonly used fields. I wonder how come this was not raised by the ICNP reviewers. In fact:

(i) you cannot use 802.1Q VLAN as the VLAN header is only 32-bits long (while your header is 41). You could use double-tagging as per 802.1ad which provides 64 bits but still these VLAN headers are placed between the SRC MAC and EtherType fields of the Ethernet header (so after the VLAN headers follows part of the Ethernet header and not the IP header as you state in Figure 6).

(ii) The MPLS header is also 32 bits, so you cannot encode your tags there either. Even if you could, a packet with SFC info encoded into MPLS header goes through an MPLS-based switch, the forwarding outcome is rather unpredictable.

That said, you must re-design your approach from scratch if you want to make it functional for current OpenFlow-based switches and networks that heavily use existing tagging schemes such as MPLS or VLAN.

Response: Sorry for the inappropriate description of the implementation of SAFE-ME. We agree with the reviewer that the VLAN tags and MPLS labels are frequently used in the network, and a single VLAN tag or MPLS label is not enough to accommodate the SAFE-ME header with 41 bits. We give a detailed response to the above doubts below.

In fact, **SAFE-ME is flexible for different types of data planes.** We can deploy SAFE-ME according to specific network scenarios. **(I)** If the data plane is composed of the programmable switches (e.g., Open vSwitches, P4 switches, etc.), we can implement the SAFE-ME scheme by adding a new SAFE-ME header or header fields. **(II)** If the SDN switches in the data plane do not support adding or parsing new protocol headers, then we can only deploy SAFE-ME with the help of existing protocol headers, which these SDN switches can process. We will choose the unused or specific protocol header to map the SAFE-ME header according to specific network scenarios. We give MPLS labels and VLAN tags only to illustrate the applicability of SAFE-ME.

Here, we take an MPLS label-based SAFE-ME deployment as an example. To implement the SAFE-ME scheme with 41 bit or longer header length in the SDN switches which do not support adding or parsing new protocol headers, we can use multiple MPLS headers that form an MPLS header stack to store the SAFE-ME header fields with a specific design. If this network heavily uses MPLS protocol, extra flow entries and actions should also be designed

to distinguish these MPLS headers' different uses. The solution is similar to the one described in the response of Comment 1.21 by using a specific value of the "EXP" field and pre-deployed flow entries, which makes the normal MPLS forwarding compatible with SAFE-ME.

Although the SDN switches which do not support adding or parsing new protocol headers can run the SAFE-ME scheme, using the programmable switches will be more convenient. The OVS can support adding and parsing new headers by adopting a few modifications to the source code. However, after studying the P4 programming language and switches recommended by the reviewer, we found that it is much easier to deploy SAFE-ME with the P4 data plane. Thus, **we have designed and implemented SAFE-ME with P4 switches, and make the following two revisions.**

i) *We have added a new Section 4.4 to describe the SAFE-ME design based on P4 switches and have given Fig. 7 to show the format of the SAFE-ME header implemented with P4:*

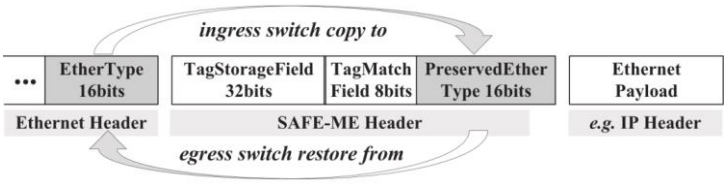

Fig. 7: The SAFE-ME Header Format in P4 Switches

Leveraging the flexibility of $P4_{16}$ programming language [29], SAFE-ME can easily be settled down to the Programming Protocol-independent Packet Processors (P4) [30] switches by adopting a new header after the Ethernet header. The new SAFE-ME header format is depicted in Fig. 7.

To better fit into the existing network protocol stack (e.g., TCP/IP), a few tunings are taken towards the SAFE-ME header format mentioned in Fig. 6. The "Tag Storage Field" and "Tag Match Field" are not modified. To preserve the "Ether Type" value in the Ethernet header of the original packet, a 16-bits "Preserved Ether Type" field is appended in the new SAFE-ME header. We also choose an experimental "Ether Type" value "0x0123" for the Ethernet header to indicate the existence of the following SAFE-ME header. This particular "Ether Type" value provides the same usage as the "Flag bit" in the original SAFE-ME header. Thus, the "Flag Bit" field is removed.

Based on the SAFE-ME scheme, if a packet matches the SFC table at the ingress switch, a SAFE-ME header is generated and inserted after the Ethernet header. The value of the "Ether Type" field in the Ethernet header is set to be "0x0123" to announce that it is followed by a SAFE-ME header. As shown in Fig. 7, the switch saves the original Ethernet type of this packet to the "Preserved Ether Type" field in the SAFE-ME header for preservation. Then this packet is forwarded by SAFE-ME tag-shifting and NF table matching scheme. When it goes to the egress switch, the SAFE-ME header will be removed, and the original Ethernet type will be retrieved and restored from the "Preserved Ether Type" field in the deleted SAFE-ME header. Thus, an unmodified packet as the source sent is finally received by the destination and the SFC rules are obeyed. Please note the field preservation method used in SAFE-ME makes it suitable for any network protocols employing type field to indicate the

following header. The matching and forwarding pipeline are implemented strictly following the SAFE-ME workflow described in Fig. 3.

ii) We have revised the description in the 2nd paragraph of Section 4.5 as below.

For other SDN switches, we can leverage some existing protocol header fields (e.g., VLAN tags, MPLS labels, etc.) to embed the SFC policy [6] [31].

[29] P4.org. (2021) P4 16 -language-specification. [Online]. Available: https://p4lang.github.io/p4-spec/docs/P4-16-v1.2.1.html

[30] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: programming protocol-independent packet processors," Comput.Commun. Rev., vol. 44, no. 3, pp. 87–95, 2014.

[31] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014. USENIX Association, 2014, pp. 543–546.

Comment 1.19 *Apart from this, the number of bits being used for an NF is 8, which is very low. There may be not so many different types of NFs in total, but have you considered how to discriminate instances of the same NF? For example, you may use a given 8-bit number for FW, but 100 instances of this firewall might be deployed in a network. Then another 200 instances of the IDS may exist. Each FW or IDS instance requires a unique 8-bit number in order to be uniquely distinguishable. This quickly adds up to more than 256 NFs. Although you discuss this issue later:*

"Even in a large-scale network, we may need to use 11 bits to identify 2047 different NFs and the maximum length of SFCs may be 10 [25] (i.e., the cost is 11 × 10 + 1 = 111 bits), the bandwidth cost for embedding a tag is still negligible (< 2%)." this is not enough. Your current design should readily support the requirements of real large scale networks and apparently it does not. This is an important scalability aspect. "No further comments for this section, your approach is still flawed and you made no effort to fix it.

You just mention that P4 could be used to "reduce the difficulty of implementing a SAFE-ME style data plane" (these are your words in Section 4.4), but you did not consider using P4 in the end, although this is what you need.

Response: Thanks for the comment. SAFE-ME uses a tag to identify a single NF. When the number of different NFs significantly increases in large-scale networks, SAFE-ME needs more bits to identify the NFs in the SAFE-ME header. Thus, how to reduce the overhead of the SAFE-ME header and how to implement SAFE-ME into the data plane of the large-scale networks are the two major applicability issues. To this end, SAFE-ME only needs some modifications to address these issues. Here, we present two solutions: **the coarse-grained tag mapping scheme** and **the region-based tag reuse scheme.**

- In the coarse-grained tag mapping scheme, **SAFE-ME uses the tag to identify a set of NFs with the same type instead of representing a single NF.** Since the types of NFs are usually much less compared with the quantity of NFs, SAFE-ME can significantly reduce the tag bit length in use and deploy its header with low overhead. When a packet

matches an entry in the NF table, it will be forwarded to one NF selected from a set of NFs sharing the same type. The method of choosing the next NF can be various or user-defined. For example, we can choose the shortest path NF as the next one or let the controller dynamically decide based on the workload of NFs to achieve load balancing. Please note, if we use the tag to identify a single NF, the flow control is fine-grained which may offer better routing performance but a large overhead of in the SAFE-ME header. If we choose the coarse-grained tag mapping scheme, we can reduce the deployment overhead and difficulty, but may degrade the routing performance. Thus, there is a trade-off between the different tag mapping schemes for different network scales and scenarios.

- In the region-based tag reuse scheme, a large-scale SDN network may use a distributed control plane containing several controllers [29]. Each controller may manage a set of switches that forms a local region. In this way, SAFE-ME can reuse the same tag to identify different NFs in the context of different regions. This tag reuse scheme can significantly reduce the SAFE-ME tags' bit length and is applicable for large-scale networks.

We have added discussions about the applicability of SAFE-ME towards large-scale networks in Section 4.5 as below.

Applicability for Large-Scale Networks. In a large-scale network with a large number of NFs, SAFE-ME will need more bits to identify the NFs of the SFC in its header. To be applicable under this situation, we present two solutions. (I) The coarse-grained tag mapping scheme. In this scheme, SAFE-ME can use the tag to identify a set of NFs with the same type (e.g., they are all IDS, etc.) instead of representing a single NF. The packet that matched the NF table entry will be forwarded to one NF selected by a user-defined method from a set of NFs sharing the same type. Since the types of NFs are usually much less compared with the quantity of NFs, this coarse-grained tag mapping scheme can significantly reduce the tag bit length in use, but may degrade the routing performance. Thus, there is a trade-off between the two different tag mapping schemes for different network scales and scenarios. (II) The region-based tag reuse scheme. A large-scale SDN network may use a distributed control plane containing several controllers [32]. Each controller may manage a set of switches that forms a local region. In this way, SAFE-ME can reuse the same tag to identify different NFs in the context of different regions. This tag reuse scheme can also significantly reduce the SAFE-ME tags' bit length.

[32] D. E. Sarmiento, A. Lebre, L. Nussbaum, and A. Chari, “Decentralized SDN control plane for a distributed cloud-edge infrastructure: A survey,” IEEE Commun. Surv. Tutorials, vol. 23, no. 1, pp. 256–281, 2021.

[Comments for Section 6]

Comment 1.20 *My 3 comments from last year's revision are still valid:*

- The topology of your testbed (shown in Figure 7) is really a toy topology. Even your simple example in Figure 4 was more complex than that.

Response: Thanks for the concern. This test-bed is merely for a confirmatory test. **From this test, we just expect to observe how the network performance will be impacted by the tag operations on hardware switches.**

Although shift operation is high-speed for ASIC [33], some commodity switch chips may not fully support this function. Under this situation, we can leverage NF to fulfill shift operations. So it is also necessary to evaluate the cost of tag operations for the VNF.

The results shown in Table 2 have confirmed that these tag operation costs are negligible. The test shows the advantage that SAFE-ME is light-weighted. If some NF vendors want to introduce SAFE-ME into their productions, they will not worry about the performance impact. Thus, even if this test is small, the outcomes are still helpful.

We should note that the best use scenario for SAFE-ME is based on a fully programmable data plane, such as P4 switches consisted data plane. Otherwise, some compromises need to be made, such as using unused or specific fields of existing protocol headers (if the switch does not support new field or header addition) or adopt additional software or VNFs to implement the shift operations (if the switch does not support tag shifting). In the revised manuscript, we have adopted the reviewer's recommendations to implement SAFE-ME based on P4 switches. Considering the limitation of space, we have replaced this part with the P4-based implementation as illustrated in Response to comment 1.18.

[33] J. Wolkerstorfer, E. Oswald, and M. Lamberger, “An asic implementation of the aes sboxes,” in Cryptographers Track at the RSA Conference. Springer, 2002, pp. 67–78.

Comment 1.21 - *Sections 6.2 and 6.3 report some results taken from the "real" testbed using PICA8 switches and an emulated testbed using OVS. However, given my comment above, how did you encode the tagging info into those OpenFlow switches? What existing headers did you use (VLAN/MPLS or something else)? How many bits did you finally use for tagging, given that both MPLS and 802.1Q VLAN provide only 32 bits (with a given structure of fields)?*

This is important information that is missing. One could not reproduce your results without this information.

Response: Sorry for the unclear description. **We have replaced the whole Sections 6.2 and 6.3 with a new experiment of SAFE-ME on P4 software switches** (see response to Comment 1.18 for detailed description). Due to the space limitation, we are not able to add the implementation details of the OVS version in the paper. So, we give the OVS-based SAFE-ME implementation details here.

In the original SAFE-ME implementation with Open vSwitch or OpenFlow-based switches (e.g., PICA8 switches), basically, the length of SAFE-ME header should be 41 bits, and we need to use multiple MPLS headers to deploy it. However, since we use Telstra, a small-scale topology in which the total NF quantity is 9, we only use 4 bits to represent the NFs, from 0b0001 to 0b1001 for simplicity. So the SAFE-ME header length for representing an SFC with at most 5 NFs is 21 bits ($4bits \times 5$ for tags plus $1bit$ for flag). Thus, we can use only

one 32 bits MPLS header to deploy the SAFE-ME header for system feasibility and performance test. We will give a thorough description of our former experiment implementation below.

First, it is important to show how we map the SAFE-ME header into an MPLS header. We store the Tag Storage Field and Tag Match Field of the SAFE-ME header together into the 20-bit length "Label field" and the Flag Bit into the last bit of the "Experimental Use field" of the MPLS header. Here we give a **Fig. 1** below to show this tag mapping relationship.

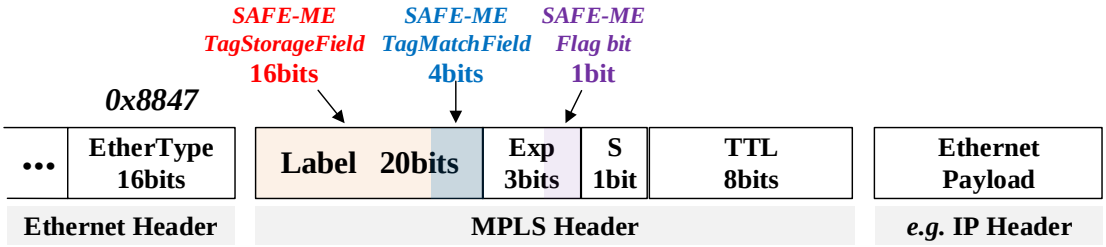

Fig. 1 SAFE-ME header mapping with an MPLS header in OpenFlow-based SDNs

Next, we will explain the implementation details based on the matching process of the three tables: the SFC, NF and Flow table. In the experiment, the flow table with id 0 is the SFC table, id 1 is the NF table, and id 2 is the Flow table as described in Section 3.2 and 4.1 of our paper.

[For the SFC table:]

When a packet arrives at a switch, it first comes to the SFC table. In the SFC table, the entries containing the match field "**mpls_tc**" (this is the match field name for the Experimental Use field defined in OpenFlow Specifications 1.3.5 and RYU controller [34]) will have higher priority. The packet matching and forwarding depend on the following situations:

- **The packet contains an MPLS header and its mpls_tc field = 1**. This means it has a valid SAFE-ME header and should be forwarded based on the NF table. The action will be GOTO_TABLE 1.
- **The packet contains an MPLS header and its mpls_tc field = 0.** This means the packet has finished all the NF traveling and will do normal forwarding based on the Flow table to the destination, the action will be GOTO_TABLE 2; There is a special case, if this switch is the egress switch directly connected to the destination host, the MPLS header with mpls_tc field = 0 will be removed with POP_MPLS action and GOTO_TABLE 2. This is where we remove the SAFE-ME header and restore the original packet.
- **The packet contains an MPLS header and its mpls_tc field = 2 (0b010).** This means it is a normal MPLS packet. It will be treated as a normal packet and directly GOTO_TABLE 2. Setting this particular value of the mpls_tc field ensures the compatibility between the SAFE-ME scheme and normal MPLS packet forwarding. Note that the OpenFlow-based MPLS Label Switching Router (LSR), responsible for adding the normal MPLS header, will set the mpls_tc field with value 2 to distinguish with the MPLS headers used by SAFE-ME. Besides, normal MPLS headers can follow the SAFE-ME header which is also deployed with other MPLS headers.

- **The packet does not contain the MPLS header**. In one situation, it has to obey one SFC rule. Then it will certainly match a pre-deployed entry in the SFC table to add an MPLS header by PUSH_MPLS action, and then set the mpls_label and mpls_tc fields by the SET_FIELD action for deploying the new SAFE-ME header fields. At last, it will GOTO_TABLE 1 to do NF Table matching and forwarding. In another situation, it has no SFC rules to obey. Then it will not match any entry in the SFC table, except the Table-miss entry. The action of the Table-miss entry is GOTO_TABLE 1.

[For the NF table:]

- In the NF table, the packet will try to match the flow entries with both mpls_tc field = 1 and the same last 4 bits of the MPLS label field. In one situation, if the current switch is the closest one to an NF and the packet matches the flow entry successfully, the action of this entry will be SET_FIELD with the tag shifting results into the packet's mpls_label field, and the packet will be forwarded to the NF. In another situation, the packet will not be changed and will be output through the specific port defined in the action of the flow entry matched. Finally, the action of the Table-miss entry for the NF table is GOTO_TABLE 2.
- The NF table contains a flow entry with the highest priority to match the packet whose "mpls_tc = 1 and mpls_label = 0", this means the packet has finished traversing through all the NFs, but its mpls_tc did not set to 0. The matched flow entry will use SET_FIELD action to make the packet's mpls_tc field = 0 and then GOTO_TABLE 2. This action is important to correct the status of the SAFE-ME header fields represented by the MPLS header.

[For the Flow table:]

- Flow Table with id 2 is the basic forwarding table. Here the packets will do traditional forwarding or drop based on the non-SAFE-ME headers (e.g., ipv4, TCP/UDP, etc.). Note that, if the packet has an MPLS header and the mpls_tc field = 2, it will be treated as a normal MPLS header in the Flow table. This step completes the normal MPLS packet forwarding process in the SAFE-ME scheme.

Here, we have explained how we deploy the SAFE-ME header with an MPLS header and how the actual forwarding pipeline is implemented in our former OVS-based experiment. This deployment scheme also applies to other OpenFlow-based switches.

[34] Ryu. (2021). [Online]. Available: https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html

Comment 1.22 - *Finally, the number of references is very low. Journals such as ToN typically require thorough related work studies. You should consider reading and discussing the efforts I mentioned above and even more.*

Response: Thanks for the suggestion. We have studied many related works and technical documents during this round of revision, including P4 technical papers, SFC routing and consolidation platform literature, etc. We have added 23 and removed 4 related works to the reference. In the revised paper, we have used red color to highlight them.

Comment 1.23 *"Our proposed SAFE-ME method implements shift operation on VNF to forward requests through path* $s_1 - v_1\ (add\ tags) - \text{IDS}\ (shift\ operation) - v_1 - v_2\ (delete\ tags) - s_2$*." If I were a network operator, I would never buy your solution. Despite the applicability issues I mentioned in Sections 3 and 4, there is one more reason that your solution is bad. You perform the shift operation in the VNF. This means that your solution is not transparent to the VNFs being used, thus one needs to modify each VNF to perform tag shifting. Tag shifting is something that YOU introduce, thus it is YOUR system that must deal with it, not the VNFs.*

Response: Thanks for the comment. We should note that the best application scenario for SAFE-ME is based on a fully programmable data plane, such as P4 switches consisted data plane. In this situation, **the tag operations are performed not by VNFs, but by our SAFE-ME system.** Thus, no modification is required for the VNFs. Otherwise, if the switch does not support tag shifting, adopt additional software or VNFs to implement the shift operations is needed for the SAFE-ME scheme. The situations are two folds:

- If the VNFs can be modified, they will implement the tag shifting function and help the SFC routing. The SAFE-ME scheme can be deployed without change. Programmable switches can support our systems and operations on tags.
- If the VNFs cannot be modified. It is easy to deploy a software programmable switch (e.g., OVS) on the server running the VNF. The tag shifting operation can be done by this software switch. Thus, the SAFE-ME scheme can still work without the modifications of the VNFs.

Note that, in the revised manuscript, we have replaced this part with P4-based SAFE-ME implementation due to space limitation. In this situation, the tag operations are performed not by VNFs, but by our SAFE-ME system.

Comment 1.24 *"Each test is executed 50 times and the average testing results are listed in Table 2." You should modify the caption of Table 2 to explain that the reported values are average values across 50 repetitions of the experiment.*

Response: We have revised the caption of Table 2 as the reviewer suggested.

TABLE 2: Tag operations on switch/NF only cost < 5% performance loss (results from average values in 50 times run).

Comment 1.25 *"Note that, the rule installation speed of a physical switch (e.g., 50.25ms/rule on HP 5130 switches [16]) is substantially much slower than that of OVS, which means the gap of update delay is even larger among these solutions on the physical platform." This is not necessarily true as the reference HP 5130 switch is very slow. There are much faster OF switches than this crappy one.*

Response: In the revised paper, we have removed these descriptions since we replace Section 6.2 and 6.3 with the P4-base SAFE-ME implementation. Instead, we have tested the average

flow rule installation speed of the P4 software switch – "*simple_switch_grpc*". The result is about 0.517 ms/rule.

We have revised the 5th paragraph of Section 6.2 as below.

The rule installation speed is about 0.517 ms/rule on P4 software switches by our test, which is almost twice that of using OVS and will significantly affect update performance.

Response to Review #2:

Comment 2.1 *Regarding the larger comment, one of the recent developments in middlebox traversal is segment routing via IPv6, now standard e.g., in the Linux kernel, and it strikes me as odd that this technology is not discussed here, not even in 2.2. Why is it omitted? I realize there is not much fine-grained control, but as a basic routing that guarantees middlebox traversal, it should still be discussed. What are the upsides/downsides in comparison to your solution? How does it compare conceptionally and also in practice? You do not need to run new simulations, but an extended discussion would be good.*

Response: Thanks for the comment. Segment Routing over IPv6 (SRv6) [32] is frequently employed to enforce the SFC policy. By using an ordered list of tags (each tag is a 128-bit length IPv6 address) in the segment routing header, SRv6 adopts a source routing mechanism to implement the SFC routing [33]. SAFE-ME is also based on source routing and tag-based forwarding, but it has three main advantages compared with SRv6. **(I) SAFE-ME introduces less overhead in the header.** SAFE-ME can use dynamic bit length to identify an NF according to the number of the total NFs in the network. For example, when the number of the total NFs is less than 256, SAFE-ME uses an 8-bit length tag to represent an NF. However, SRv6 uses 128-bit length IPv6 address to act as a tag. If an SFC contains 5 NFs, then the SRv6 header will be more than $5 \cdot 128 = 640$ bits. SAFE-ME only needs 41 bits. Thus, SAFE-ME can reduce the header overhead by 93.6% compared with SRv6 in this case. **(II) SAFE-ME is more flexible.** SRv6 extends the IPv6 header to fulfill the segment routing, which is a Layer-3 protocol. The SAFE-ME header can be added to any position in the packet header stack according to the need and design. Thus, the deployment and usage of the SAFE-ME header are more flexible and universal than that of SRv6. **(III) The SAFE-ME routing scheme is more fine-grained and efficient.** SRv6 basically uses an IPv6 address to match and forward packets. However, with the rich and even programmable matching fields in the flow entries, SAFE-ME can provide dynamic-granularity SFC routing based on various matching rules. Meanwhile, the design of the SAFE-ME control plane introduced in Section 5 also ensures excellent network performance.

We have appended these discussions in a new paragraph at the end of Section 4.5.

Comparison with Segment Routing over IPv6 (SRv6). To enforce the policy for SFC, SRv6 [45] adopts a source routing mechanism by using an ordered list of tags (each tag is a 128-bit length IPv6 address) in the segment routing header [46]. Compared with SRv6, SAFE-ME is also based on source routing and tag-based forwarding scheme, but has three apparent advantages. (I) SAFE-ME introduces much less overhead in the header. For example, SAFE-ME can use an 8-bit length tag to represent an NF when the total number of NFs is 256, while SRv6 needs 128-bit. Thus, SAFE-ME can reduce the header overhead by more than 93% compared with SRv6 in this case. (II) SAFE-ME is much more flexible. SRv6 uses the IPv6 header which is a Layer-3 protocol. The SAFE-ME header can be added at any position into the packet header stack based on the need and design. (III) The SAFE-ME routing scheme is much more fine-grained and efficient. SRv6 basically uses an IPv6 address to match and forward the packet. However, SAFE-ME can provide dynamic-granularity SFC routing based

on various matching rules in the SFC table. Meanwhile, the SAFE-ME control plane introduced in Section 5 also ensures excellent network performance.

[35] Z. N. Abdullah, I. Ahmad, and I. Hussain, "Segment routing in software defined networks: A survey," IEEE Commun. Surv. Tutorials, vol. 21, no. 1, pp. 464–486, 2019.
[36] IETF. (2020) Ipv6 segment routing header (SRH). [Online]. Available: https://www.rfc-editor.org/rfc/rfc8754.txt

Comment 2.2 *Regarding the smaller technical comments, I would like to dive into Section 5. The feasible routing paths are just a heuristic, and hence Lemma 2/3 cannot guarantee approximation guarantees over all solutions, but just for those leveraging the paths in 5.2.1. This must be mentioned and clarified: when plugging in all paths into P(f), your formulation grows exponentially. Please keep the statements but clarify them regarding the limited paths that are used.*

Moreover, please separate the "practical scenarios" statements from the Lemmas 2 and 3 into separate corollaries, where you clarify that the bound is 2.

Response: Sorry for the inappropriate description. We have clarified that the approximation performance analysis is based on the set of feasible routing paths exploited in Section 5.2.1. Moreover, we have improved the description of Lemmas 2 and 3 and separate each lemma into two lemmas according to different scenarios in Section 5.2.6 as follows.

Lemma 2. Taking the optimal solution obtained from the set of the feasible paths exploited in Section 5.2.1 as a benchmark, the proposed RBSU algorithm can achieve the approximation factor of $\frac{3\log n}{2\alpha}+2$ for link/NF capacity constraints in large networks, where n is the number of switches.

Lemma 3. In most practical scenarios, e.g., $\alpha \geq 3\log n$, the proposed RBSU algorithm can achieve the approximation factor of 2 for link/NF capacity constraints.

Lemma 4. Taking the optimal solution obtained from the set of the feasible paths exploited in Section 5.2.1 as a benchmark, after the rounding process, the amount of required flow/NF entries on any switch v will not exceed the number of residual flow/NF entries by a factor of $\frac{3\log n}{2\alpha}+2$ in large networks, where n is the number of switches.

Lemma 5. In most practical scenarios, e.g., $\alpha \geq 3\log n$, the proposed RBSU algorithm can achieve the approximation factor of 2 for flow/NF table size constraints.

Comment 2.3 *Moreover, regarding NP-hardness, routing already through one middlebox under load minimization is NP-hard already for 1 flow, please see*

https://dl.acm.org/doi/10.1145/3211852.3211859 (here the setting of directed graphs can be obtained with background flows and the question of capacity violation is equivalent to asking if a load of at most 100% can be achieved).

Response: Sorry for the cumbersome certification process regarding NP-hardness proof. We have improved the description of the NP-hardness proof of Theorem 1 in Section 5.2.4 as follows.

We consider a special example of the LSRU-MB problem, in which only one flow needs to traverse one middlebox. This special case of LSRU-MB has been proven to be NP-hardness [37]. Thus, the LSRU-MB problem is NP-Hard too.

[37] S. A. Amiri, K. Foerster, R. Jacob, and S. Schmid, "Charting the algorithmic complexity of waypoint routing," Comput. Commun. Rev., vol. 48, no. 1, pp. 42–48, 2018.

Comment 2.4 - *please take a pass over the references and unify the style, e.g. [1] lists the abbreviation and others do not etc. A suggestion would be to use e.g., DBLP for a unified style. For another example, 19 lists "B. Switches" as the author*

Response: Thanks for the comment. We have carefully revised all of the references with the unified style followed by DBLP. We give some examples as follows.

[1] G. Zhao, H. Xu, J. Liu, C. Qian, J. Ge, and L. Huang, "SAFEME: scalable and flexible middlebox policy enforcement with software defined networking," in 27th IEEE International Conference on Network Protocols, ICNP 2019. IEEE, 2019, pp. 1–11.

[16] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest path and maximum flow problems under service function chaining constraints," in 2018 IEEE Conference on Computer Communications, INFOCOM 2018. IEEE, 2018, pp. 2132–2140.

[38] Barefoot-networks. (2014) Barefoot Tofino Switches. [Online]. Available: https://www.barefootnetworks.com

[44] D. E. Sarmiento, A. Lebre, L. Nussbaum, and A. Chari, "Decentralized SDN control plane for a distributed cloud-edge infrastructure: A survey," IEEE Commun. Surv. Tutorials, vol. 23, no. 1, pp. 256–281, 2021.

Comment 2.5 - *Your packet traces are shared on Dropbox, but this is not a "permanent" solution. Please consider using some data sharing service that makes the data available on the long term (either "professionally" e.g., with IEEE or at least on Github/Bitbucket etc.)*

Response: Thanks for the suggestion. We have uploaded the packet trace data to Github [39]. Since the trace file is larger than 100 MB which has exceeded the file size limitation for a single file in Github, we have compressed it into several small volumes.

[39] packet traffic trace. (2021). Accessed: Aug. 16, 2021. [Online]. "SAFE-ME experiment packet traffic trace". Available: https://github.com/xipeng-ahnu/SAFE-ME/tree/master/trafficTrace

Comment 2.6 - *Would you consider making your implementations publicly available as well to help future research? Thank you.*

Response: Thanks for your suggestion. We have already uploaded and shared the P4-based SAFE-ME experiment environment on Github [40]. Please check the "README.html" file in the repository to get the instructions about how to run it.

[40] SAFE-ME research group. (2021). Accessed: Aug. 16, 2021. [Online]. "SAFE-ME experiment environment with P4 data plane implementation". Available: https://github.com/xipeng-ahnu/SAFE-ME /tree/master

Response to Review #3:

Comment 3.1 *However, in my opinion, the additional contributions over the previous ICNP2019 pare are minimal and not enough to recommend the publication in TNET. The sections 1 to 5 are basically the same in both papers. The secretion 6 (performance evaluation) only includes a few additional results and doesn't overcome the main limitation of the paper that is the limited scale of the evaluation scenarios. The large scale simulations section, for instance, is almost equal to the one in the ICNP paper, and doesn't provide enough details about the simulation model, traffic dynamics, service chain dynamic behavior and performance results to support the claimed scalability of the proposed solution*

Response: Thanks for the concern. We have made a lot of changes and updates in this revision. We believe that the revised manuscript adds a lot of extra contributions compared with the ICNP19 version, including the organization of Sections 1, the in-depth study of related works in Section 2, a clearer illustration of SAFE-ME in Section 3.3, the illustration of P4-based SAFE-ME in Section 4.4, and the P4-based SAFE-ME experiments in Section 6.3. We summarize the important updates as below:

1) We have added P4-based SAFE-ME implementation and given the experimental results. Specifically, we have implemented SAFE-ME with the P4 switches and described its design in Section 4.4. Moreover, we have replaced the entire content of former Sections 6.2 and 6.3 with a new Section 6.2 containing the detailed description of the P4-based SAFE-ME experiments. The experimental results also strongly support the correctness, feasibility and scalability of SAFE-ME.
2) We have improved the description of both Section 3 and Section 4 to enhance the design of SAFE-ME. We have also given detailed illustration for our former OVS-based SAFE-ME implementation.
3) We have reorganized the introduction section (Section 1) and revised the background section (Section 2) to enhance our motivation.
4) We have enhanced the description of the lemmas and the NP-hardness proof in Section 5.2.
5) We have done an in-depth study of related works and added some new references in Section 2.
6) We have added a paragraph at the beginning of each section to briefly introduce the overview of that section.
7) We have updated the content of the Github repository to share the source code, packet trace data and documentation of our SAFE-ME experiments.
8) We have carefully checked the manuscript and modified some typo/grammar issues in the revised manuscript.

Dear Editor:

Here within enclosed is our paper for consideration to be published on "IEEE/ACM Transactions on Networking". We sincerely appreciate the detailed comments and constructive suggestions by the editor and anonymous reviewers. The manuscript (No. TNET-2020-00607) has been revised according to the comments. The further information about the paper is in the following:

The Title:
SAFE-ME: Scalable and Flexible Policy Enforcement in Middlebox Networks

The Authors:
Hongli Xu, Peng Xi, Gongming Zhao, Jianchun Liu, Chen Qian, Liusheng Huang

The authors claim that none of the material in the paper is under consideration for publication elsewhere. I am the corresponding author and my address and other information is as follows:

Address: School of Computer Science and Technology,
University of Science and Technology of China, Hefei, Anhui, 230027, P.R.China
E-mail: xuhongli@ustc.edu.cn
Tel: 86-551-3602445
Fax: 86-512-87161305

Thank you very much for consideration!
Sincerely Yours.

Prof. Hongli Xu

Some preliminary results of this paper were published in the Proceedings of IEEE ICNP 2019. In this version, we have made the following main improvements compared to the ICNP version.

(1) We have enhanced the performance evaluation in the revised version.

(1.1) We have replaced the entire content of former Sections 6.2 and 6.3 with a new Section 6.2 containing the detailed description of the P4-based SAFE-ME experiments. The experimental results also strongly support the correctness, feasibility and scalability of SAFE-ME.

(1.2) We have given a more comprehensive description of the large-scale simulations in Section 6.4. Specifically, we have added Figure 21 to show the average number of required entries by changing the number of requests and plotted Figure 24 to present the average link load performance by varying the number of requests.

(2) We have reorganized the introduction section (Section 1) and revised the background section (Section 2) to make the research motivation clearer.

(3) We have enhanced the illustration of SAFE-ME system in the revised version.

(3.1) We have implemented SAFE-ME with the P4 switches and described its design in Section 4.4.

(3.2) To better illustrate the system overview of SAFE-ME, we have added Figure 2 in Section 3.1.

(3.3) We have added Section 4.1 and Figure 5 to illustrate the pipeline process in OpenFlow Switch, which will help us to better understand the data plane design of SAFE-ME.

(3.4) We have shown how SAFE-ME deal with various network failure scenarios in Section 5.1.4.

(3.5) We have given an example to illustrate the exploration of feasible paths in Section 5.2.1.

(4) We have illustrated the applicability of SAFE-ME, including how to combine it with the consolidated platform, and how to apply it to large-scale networks in Section 4.5. We also have compared SAFE-ME with SRv6 in Section 4.5.

(5) We have enhanced the description of the lemmas and the NP-hardness proof in Section 5.2.

(6) We have updated the content of the Github repository to share the source code, packet trace data and documentation of our SAFE-ME experiments.

(7) We have modified some language errors in the revised version.

SAFE-ME: Scalable and Flexible Middlebox Policy Enforcement with Software Defined Networking

Gongming Zhao[1,3] Hongli Xu*[1,3] Jianchun Liu[1,3] Chen Qian[2] Juncheng Ge[3] Liusheng Huang[1,3]
[1]School of Computer Science and Technology, University of Science and Technology of China, China
[2]Department of Computer Science and Engineering, University of California Santa Cruz, USA
[3]Suzhou Institute for Advanced Study, University of Science and Technology of China, China

***Abstract*—The past decades have seen a proliferation of middlebox deployment in various networks, including backbone networks and datacenters. Since network flows have to traverse specific service function chains (SFCs) for security and performance enhancement, it becomes much complex for SFC routing due to routing loops, traffic dynamics and scalability requirement. The existing SFC routing solutions may consume many resources (*e.g.*, TCAM) on the data plane and lead to massive overhead on the control plane, which decrease the scalability of middlebox networks. Due to SFC requirement and potential routing loops, solutions like traditional default paths (*e.g.*, using ECMP) that are widely used in non-middlebox networks will no longer be feasible. In this paper, we present and implement a scalable and flexible middlebox policy enforcement (SAFE-ME) system to minimize the TCAM usage and control overhead. To this end, we design the smart tag operations for construction of default SFC paths with less TCAM rules in the data plane, and present lightweight SFC routing update with less control overhead for dealing with traffic dynamics in the control plane. We implement our solution and evaluate its performance with experiments on both physical platform (Pica8) and Open vSwitch (OVS), as well as large-scale simulations. Both experimental and simulation results show that SAFE-ME can greatly improve scalability (*e.g.*, TCAM cost, update delay, and control overhead) in middlebox networks. For example, our system can reduce the control traffic overhead by about 83% while achieving almost the similar middlebox load, compared with state-of-the-art solutions.**

***Index Terms*—Software Defined Networks, Network Function, Middlebox, Default Path, Tag.**

I. Introduction

Network functions (NFs) such as firewalls, deep packet inspection, load balancer, etc. are provided by specialized network devices called middleboxes (MB) [1]. They have been widely deployed in various networking scenarios including campus networks, backbone networks, data centers and cloud computing environments [2]. Typically, network flows go through several NFs in a specific order to meet its processing requirements, also called Service Function Chaining (SFC) [1]. Thus routing with SFC (or SFC routing) becomes much more complex than the traditional network routing. We call a network with considerable middlebox deployment and fine-grained middlebox policies as a 'middlebox network'.

Due to the features of middlebox networks, there are two critical challenges for SFC routing: 1) *Routing loops*. The flows processed by the MBs will be forwarded to the MB, and then back to switch after NF processing. Thus, there exist *routing loops* in the middlebox networks [1], which is the main difference from the traditional routing solutions. 2) *Traffic dynamic* is a common issue, especially for large-scale networks. In practice, the network will experience highly dynamic flows. Moreover, even for a flow, its intensity will fluctuate with time. Thus, it is required to handle unexpected bursts experienced at the switches under dynamic traffic conditions [3].

Recently, with the advantage of the centralized control, software defined networking (SDN) [4] has become an emerging technology to conquer the above challenges of complex SFC routing. Under the SDN framework, switches forward the data packets by matching rules on the TCAM-based forwarding table. However, TCAMs are 400X more expensive and consume 100X more power per Mbit than the RAM-based storage on switches [5]. Thus, most today's commodity switches only support 2-20K entries [5] (*e.g.*, 16K entries on high-end Broadcom Trident2 switches [6]). In an SDN-based middlebox network, we should consider the scalability issue in two aspects: 1) *Control Plane Scalability.* Under the SDN framework, a newly arrival flow will be reported to the controller for route selection. However, with more flows arriving at the network, it will lead to serious per-flow communication/computation overhead on the control plane [7]. Moreover, SFC routing update, especially in a large-scale network, also expects less control overhead. 2) *Data Plane Scalability.* Due to the limited size of the TCAM-based forwarding table, it is another challenge to accommodate a large number (*e.g.*, 10^6 in a moderate-sized data center [8]) of flows using only a limited size (*e.g.*, several thousands) of forwarding entries.

To solve complex SFC routing, several works have designed efficient solutions for MB networks [1] [10]. However, these solutions still face several critical disadvantages. First, these solutions often install rules for flows with the granularity of ingress-egress switch pairs. If a network contains several thousands of ingress/egress switches, there are millions of ingress-egress switch pairs. Consequently, it may require millions of forwarding entries on a switch in the worst case, which far exceeds the forwarding table size. Moreover, these solutions (no matter using proactive mode (*e.g.*, [1]) or using reactive mode (*e.g.*, [10])) will encounter lager response time (when encountering network failures) and larger network update delay (when network performance decreases) due to the low rule installation speed, which will be validated through experimental testing in Section VI.

Though the traditional solutions, *e.g.*, default paths [12],

978-1-7281-2700-2 / 19/$31.00

Schemes	No. of Rules	Control Overhead	SFC Policy	Network Performance	Hardware Support
Traditional Default Path (*e.g.*, [9])	Few	Low	No	Low	No
Per-request Routing (*e.g.*, [1] [10])	Many	High	Yes	High	No
Consolidated Platform (*e.g.*, [11])	Few	Low	Yes	High	Yes
Our Scheme	**Few**	**Low**	**Yes**	**High**	**No**

TABLE I: Comparison of the advantages and disadvantages of existing solutions.

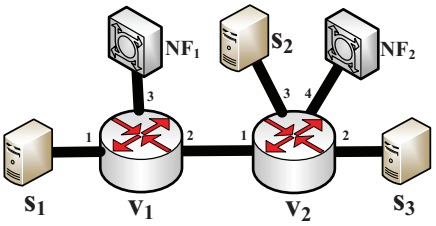

Fig. 1: Traffic from s_1 to s_3 has to traverse NF_1 and traffic from s_2 to s_3 has to traverse NF_1 and NF_2. Different requests with the same egress switch (or destination) will traverse different SFCs. However, the switch-based (or destination-based) default path solution cannot distinguish the traffic from s_1 or s_2.

can achieve better scalability and deal with traffic dynamics in traditional networks, these solutions cannot be applied directly in middlebox networks for the following reasons. First, SFC routing will cause routing loops, which is the main difference from the traditional network (Section II-A). However, default paths cannot deal with routing loops. Second, flows with the same egress switch (or destination) will be processed in the same way by default paths, but they may require different SFCs. How to setup default paths for different SFC requirements remains a challenging problem.

To conquer the above challenges, we design the scalable and flexible middlebox policy enforcement system (SAFE-ME). SAFE-ME installs three types of tables in the switch's data plane, namely the SFC table, NF table, and Flow table. The SFC table maintains the SFC policy information and assigns tags to packets that match certain policies. The NF table provides the path information to the NFs by checking the packet tags. The Flow table is on a per-switch basis to forward packets to their destinations, similar to those in traditional routers/swtiches. We design the smart tag operations for construction of default SFC paths, and present lightweight SFC routing update for dealing with traffic dynamics. Although the switch implements more logic than classic SDN switches, our implementation on Pica8 3297 switches shows that SAFE-ME reduces flow entries, control overhead, and update delay by >80%, while increasing packet forwarding delay by 3.3% to 4.8%, compared with state-of-the-art solutions.

II. Background and Motivation

A. Inapplicability of Traditional Default Path Solutions

A natural strawman solution for flow routing with less forwarding entries is deploying default paths (*e.g.*, using switch-based or destination-based OSPF/ECMP methods) [12] [9]. However, in middlebox networks, there may exist routing loops in the forwarding paths due to SFC requirements. In addition, flows with the same egress switch (or destination) may traverse different SFCs, which cannot be satisfied by default paths. Thus, traditional default paths cannot solve the SFC routing problem with fine-grained middlebox policies.

We give an example to illustrate the difference of flow routing between traditional networks and middlebox networks. As shown in Fig. 1, if we forward traffic from server s_1 to server s_3 in the traditional network (*i.e.*, without any SFC requirement), we can install one entry (*i.e.*, $dst = s_3, output = 2$) on each of switches v_1 and v_2, so that traffic will be forwarded through path $s_1 - v_1 - v_2 - s_3$.

However, in MB networks, the operator may specify all traffic from s_1 to s_3 to go through NF_1 (*i.e.*, $s_1 - v_1 - NF_1 - v_1 - v_2 - s_3$) and traffic from server s_2 to s_3 through NF_1-NF_2 (*i.e.*, $s_2 - v_2 - v_1 - NF_1 - v_1 - v_2 - NF_2 - v_2 - s_3$). These two requests with the same egress switch will go through different SFCs. The switch-based (or destination-based) routing solution cannot *distinguish* the traffic from s_1 or s_2. Thus, we cannot determine the proper actions for traffic on v_2. Prior work [1] shows that for some network configurations, 15% of the SFC routing paths using the proposed approach in [1] contain loops. Hence traditional default-path methods cannot fully address the SFC routing issue.

B. Limitations of Prior Work

Though packet tags help to solve the routing loop, it may be flow-entry consuming if each 5-tuple flow is attached with a tag. Thus, many works have leveraged the per-request routing strategy to reduce the flow-entry consumption and achieve load balancing [1] [13] [14] [15]. Specifically, a *request* is identified by three elements, ingress switch, egress switch and SFC. That is, all flows with the same ingress switch, egress switch and SFC requirement, will be aggregated into one request. For each newly arrival request, the corresponding ingress switch reports the packet header information to the controller for requesting forwarding strategy. The controller then computes a proper routing path satisfying the service policy and replies the rule installment instructions to switches along the routing path. Though some 5-tuple flows are aggregated into a request, this solution still requires a large number of entries and leads to massive control overhead even in a moderate-size network. For example, in a practical data center network with 1,000 edge switches, there may exist $O(1,000 \times 1,000)$ switch pairs. Even if there is only one SFC requirement per switch pair, it may require 1M entries on a switch in the worst case, which violates today's switch capabilities [5]. When multiple SFC requirements are posed for each switch pair, it becomes more serious. Meanwhile, since many flow rules should be installed and modified under per-request routing scheme, the communication/computation overhead on the control plane is too high, which will be validated in Section VI.

To reduce the TCAM table cost and controller overhead, some research attempts to simplify the SFC routing problem in middlebox networks by constructing **consolidated platform** [11] [16] [17]. CoMB [11] is a network function consolida-

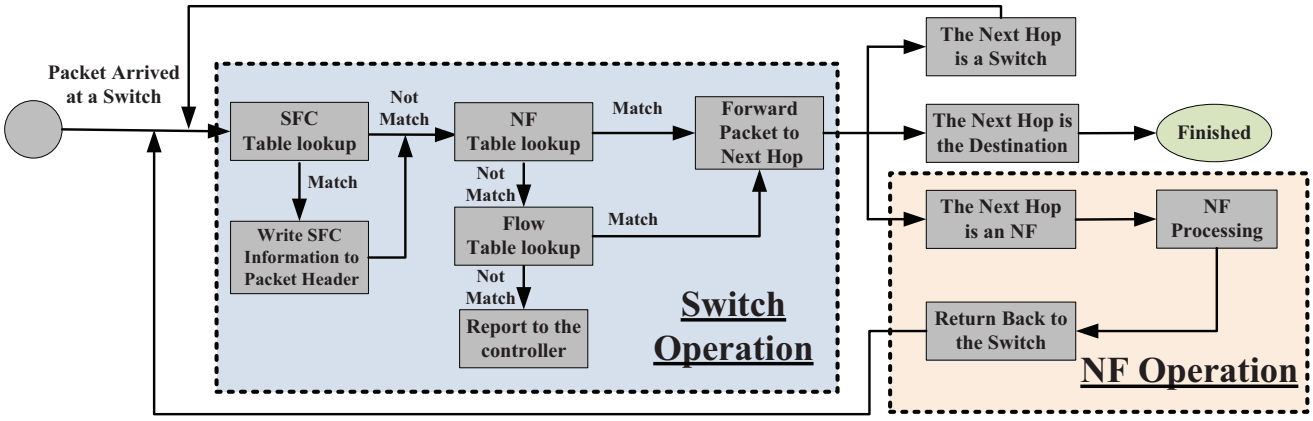

Fig. 2: Illustration of Packet Processing Procedure. When a packet arrives at a switch, the switch matches the header with SFC Table, NF Table and Flow Table in sequence. In this way, the packet will be forwarded to destination while obeying SFC constraints.

tion platform, where a flow/request can be processed by all required NFs on a single hardware platform, thus simplifying the SFC routing. OpenBox [16] and Metron [17] also adopt the consolidation conception so as to merge similar packet processing elements into one. In Fig. 1, they may integrate NF_1 and NF_2 into one mixed NF. All traffic will traverse the mixed NF and be processed automatically by corresponding functions. The consolidated platform simplifies the flow routing, helps reduce the number of required forwarding entries on the switches, and also relieves the control overhead. However, it requires specific hardwares to build the consolidated platform. Moreover, different NFs may be provided by different vendors, which prevents it to be consolidated.

From Table I, we observe that all existing methods can only address partial challenges of SFC routing in middlebox networks. In other words, none of them can achieve better routing performance with fewer flow rules, lower control overhead and SFC requirements under existing hardware platforms. Thus, in this paper, we design an efficient architecture for SFC routing so as to satisfy the above characteristics.

III. System Architecture

A. System Overview

SAFE-ME consists of data plane and control plane designs. The proposed architecture addresses the challenges of scalable SFC routing in middlebox networks by embedding the SFC information (as a tag) into the packet header. We first give an outline of the data plane and the control plane.

Data Plane of middlebox networks consists of SDN switches, NFs, servers and links. The SDN switches are responsible for forwarding packets according to installed rules in switch tables. Each NF unit processes the received packets. As specified in the OpenFlow standard [18], each SDN switch contains multiple tables. We divide these tables into three parts with different roles, called *SFC Table*, *NF Table* and *Flow Table*, respectively. We will describe the design of the data plane in Section IV.

- *SFC Table* is used to store the SFC information for each request. When a request arrives at an ingress switch, this switch will match the packet header with the *SFC Table*, and embed the matched SFC policy (as a tag) into the packet header. In other words, the packet header will contain SFC information through matching *SFC Table* on the ingress switch.
- *NF Table* stores the next-hop information of the path from this switch to each NF. Through matching *NF Table*, the packet will be forwarded to the required NFs in sequence according to the SFC information.
- *Flow Table* is responsible to store the next-hop information of the path (*e.g.*, default path or per-request path) from this switch to each egress switch in the network. After the packet is processed by all required NFs, it will be forwarded to destination through matching the *Flow Table*.

Control Plane is responsible to manage the whole network. We mainly focus on two new modules in the control plane: *Default Path Construction* (DPC) and *Lightweight SFC Routing Update* (LRU). Leveraging the network information collected by *OpenFlow API* and policy specification issued by network administrator, DPC computes the default paths from each switch to each egress switch or each NF. To avoid the possible congestion due to traffic dynamics, we also design LRU to periodically re-compute near-optimal routing strategy based on current network conditions. The results will be encapsulated into *Flow-Mod* commands to install corresponding rules on the switches. We will introduce the design of the control plane in Section V.

B. Packet Processing Procedure

We then describe the packet processing procedure of SAFE-ME. As shown in Fig. 2, the controller initially configures the *SFC Table*, *NF Table* and *Flow Table* based on the network information with a proactive manner. When a packet arrives at a switch, the switch first matches the packet header with the *SFC Table*. If there is a match, the switch will write the SFC information (as a tag) into the packet header, which means the switch is the ingress switch of this packet and the packet is required to be processed by a set of NFs. Next, if there is a match in the *NF Table*, it will be forwarded to next hop from this switch to corresponding NF, which means the packet has to be processed by this matched NF. Otherwise, the packet need not traverse NFs or have traversed all required NFs, and will be forwarded to the destination. Then, the packet follows the traditional processing procedure. There are two cases. If there is a match in the *Flow Table*, this packet will

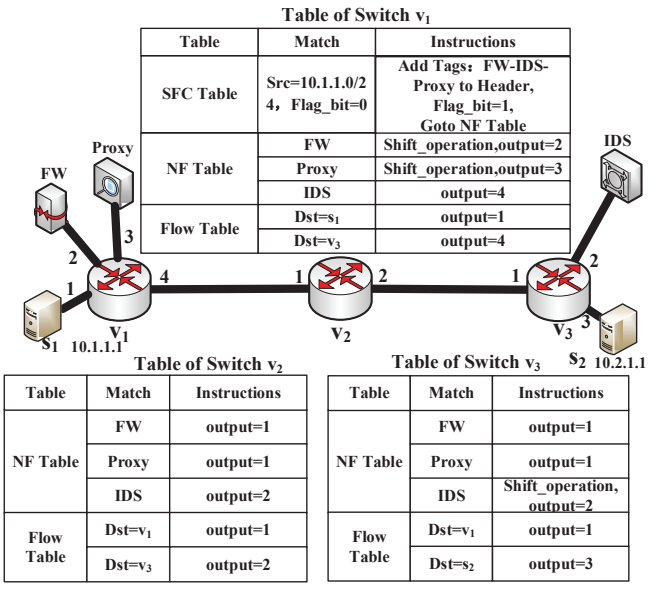

Fig. 3: Illustration of Packet Processing in SAFE-ME and Rule Installment on Switches. The administrator specifies that traffic from subnet 10.1.1.0/24 should be traversed a service function chain: Firewall-IDS-Proxy for security benefits. As a result, the packet will be forwarded by path "$s_1 - v_1 -$ FW $- v_1 - v_2 - v_3 -$ IDS $- v_3 - v_2 - v_1 -$ Proxy $- v_1 - v_2 - v_3 - s_2$".

be forwarded to the next hop according to the matching result. Otherwise, no match exists in the *Flow Table*. This packet will be reported to the controller using existing OpenFlow APIs. Note that, the switch connected with NF(s) is responsible to modify the tag in the packet header by tag shifting, which will be introduced in Section IV. In this way, after the packet is processed by the NF and returns to the connected switch, the packet will be forwarded to next required NF through matching another NF entry or forwarded to the egress switch through the *Flow Table*.

C. Illustration of SAFE-ME Design

We give an example for better understanding of packet processing in SAFE-ME. The controller initially computes the default path (*e.g.*, shortest path) from each switch to each egress switch (or NF) and installs the default paths to egress switches (or NFs) on *Flow Tables* (or *NF Tables*). Besides, the administrator may specify some policies for flows. For example, in Fig. 3, the administrator specifies that traffic from subnet 10.1.1.0/24 should be traversed a SFC: Firewall-IDS-Proxy for security benefits. Thus, the controller installs an SFC entry on ingress switch v_1 of this subnet in Fig. 3.

When a request from subnet 10.1.1.0/24 arrives at the ingress switch, v_1 will match this packet header with the *SFC table*, and write the tag (*i.e.*, the SFC information: "FW-IDS-Proxy") into the packet header. The packet will then be matched with the *NF Table* (*i.e.*, "match=FW"). Since v_1 is connected with a firewall, this switch executes shift operation (which will be introduced in Section IV) to delete the "FW-" information in the tag and then forwards this packet to the firewall through outport 2. After the packet is processed by the firewall and returns to switch v_1, this switch will match the NF entry "match=IDS and output=4", and forward this packet to switch v_2, which will then forward it to v_3 by the

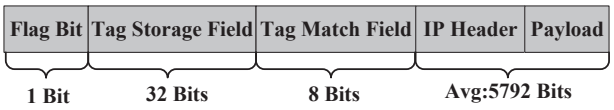

Fig. 4: Tag Storage/Match Fields and Flag Bit in a packet. Flag Bit Field indicates whether the packet has been embedded a tag or not. Tag Match Field stores the first NF in the SFC and Tag Storage Field embeds the rest NF(s) in the SFC. The overall bandwidth cost for embedding tags is negligible.

NF Table. Switch v_3 continues to forward this packet to IDS according to the *NF Table*. In this way, this packet will be forwarded through path "$s_1 - v_1 -$ FW $- v_1 - v_2 - v_3 -$ IDS $- v_3 - v_2 - v_1 -$ Proxy $- v_1 - v_2 - v_3 - s_2$". That means, we only need one special SFC entry on the ingress switch for this request, and the other entries are shared by different requests. Consequently, SAFE-ME will greatly reduce the use of rules and the control overhead.

IV. Date Plane Design

As specified by the OpenFlow standard [18], each SDN switch consists of multiple forwarding tables. We divide these tables into three parts with different roles, called *SFC Table*, *NF Table* and *Flow Table*, respectively. Leveraging the pipeline processing, the switch will first match the packet header with the SFC table, then with the NF table (if necessary), and finally with the Flow table.

A. Tag Embedding through SFC Table

In the data plane, there may exist many units of NFs. The controller uses unique identifies (*e.g.*, $1, 2, ..., m$) to distinguish these NFs. Recent studies show that the number of NFs is similar to the number of switches [19] [20] and the length of SFC is usually no more than 5 in a moderate-size network. For the sake of convenience, we use 8 bits to represent an NF and use 40 bits to indicate a SFC. In this way, we add several fields into the packet header as shown in Fig. 4. Specifically, we design (1) *Tag Match Field* to store the first NF in the SFC and (2) *Tag Storage Field* to embed the rest NF(s) of the SFC in the reverse order. Moreover, we use 1 flag bit to denote whether the packet has been embedded a tag or not. Note that, in data center networks, the average packet size is around 724 bytes (*i.e.*, 5792 bits) [21]. Even in a large-scale network, we may need to use 11 bits to identify 2047 different NFs and the maximum length of SFCs may be 10 [22] (*i.e.*, the cost is $11 \times 10 + 1 = 111$ bits), the bandwidth cost for embedding a tag is still negligible ($< 2\%$).

To illustrate the SFC information (or tag) embedding process, we revisit the example in Fig. 3. We use 0x01, 0x02, 0x03 to denote the FW, Proxy, IDS units, respectively. In this way, the service function chain can be encoded into 0x01-0x03-0x02. To embed this tag, *Tag Match Field* records the first NF (*e.g.*, 0x01) and *Tag Storage Field* stores the rest NFs in the reverse order (*e.g.*, 0x0203). In other words, the SFC entry on switch v_1 can be expressed as "$ip\_src = 10.0.1.0/24, Flag\_bit = 0, actions = \{Tag\_Match\_Field = 0x01, Tag\_Storage\_Field = 0x0203, Flag\_bit = 1, Goto\_Table : NF\_Table\}$".

When a packet from subnet 10.0.1.0/24 arrives at switch v_1, it will match the entry in the SFC Table. The actions

will modify the *Tag Match Field* to 0x01, *Tag Storage Field* to 0x0203 and set the *Flag Bit* to 1. As a result, the SFC information is successfully embedded in the packet header.

B. Tag Shifting through NF Table

The switch will then match the *Tag Match Field* (*i.e.*, the first NF) in the NF Table, and forward the packet to the corresponding port if there is a matching entry. Moreover, if the switch is directly connected with the NF as specified by the *Tag Match Field*, it will take the following operations: (1) catch the first NF information of the rest SFC from the *Tag Storage Field* (*i.e.*, $shift\_right$ operation); and (2) reset the *Tag Match Field*. We revisit the example in Fig. 3. After embedding a tag, the switch will match the packet header in the NF table about FW (*i.e.*, 0x01). There is a matching NF entry: "$Tag\_Match\_Field = 0x01, actions = shift\_right, output = 2$", which means the switch will shift right the tag and forward the packet to FW through port 2. Note that, the $shift\_right$ operation will bitwise shift right of *Tag Storage+Match Field* by 8 bits. After the $shift\_right$ operation, *Tag Match Field* and *Tag Storage Field* become $0x03$ and $0x02$, which means the packet still has to traverse two NFs (*i.e.*, 0x03-0x02). When the packet returns to switch v_1 from FW, the switch will match the NF table with the match field "$Tag\_Match\_Field = 0x03$", and forward to IDS through port 4. In the end, the packet will be forwarded to the destination.

C. Discussions

Flexibility of Implementing SAFE-ME. For some programmable switches (*e.g.*, Open vSwitches [23], P4-based barefoot switches [24]), it is not difficult to add new fields, such as *Tag Match Field* and *Tag Storage Field*, to embed the SFC information into the packet header. In fact, the development of programmable data plane technology such as P4 has reduced the difficulty of implementing a SAFE-ME style data plane and thus, has greatly improved the feasibility of SAFE-ME. For other SDN switches, we can leverage either VLAN tags, MPLS labels, or other unused fields in the IP header to embed the SFC information [1] [13]. Meanwhile, the *shift* operation is a basic and high-speed function [25]. Thus, the tag operations in SAFE-ME can achieve line rate if the switch supports *shift* operation (*e.g.*, Open vSwitches [23], barefoot switches [24]), which are also testified in Section VI-C. However, some switches may not support *shift* operation. Under this situation, we can leverage NF units to fulfill *shift* operation. Specifically, when a packet arrives at a required NF, the NF will shift the tag in the packet header before returning to the switch so that the packet will match the next required NF. FlowTags [13] has illustrated that these operations are lightweight (<0.5% cost) for an NF to modify the packet header, which is also testified by our test in Section VI-B. Thus, SAFE-ME is quite compatible with legacy networks and easy to be implemented.

Applicability for Network Function Virtualization (NFV). Similar to MB networks, NFV networks will also encounter routing loops, traffic dynamics and scalability problems, due to the disadvantages of existing solutions as shown in Table I. SAFE-ME can be applied to NFV networks with some modifications. For example, if the switch is connected to two following NFs, the switch will delete the first two NFs and write the second NF in the *Tag Store Field* to the *Tag Match Field*. These modifications are easy to implement.

V. Control Plane Design

A. Default Path Construction (DPC) for SFC Routing

As described in Section II-A, the traditional default path solution cannot be directly applied for middlebox networks. Thus, we propose novel multi-level (*i.e.*, policy-level, NF-level and switch-level) default paths for SFC routing.

1) Network Model: Once the network topology is established, the controller can obtain the topology information, such as the locations/connections of all switches and NFs, through classical OpenFlow APIs. The data plane topology can be modeled as a directed graph $G = (U \cup V \cup N, E)$, where U, V, N and E denote the terminal set, the switch set, the NF set and the directed link set, respectively.

2) Policy-level Default Path Construction: In the middlebox networks, the network operator usually specifies different sequences of NFs (*i.e.*, SFCs) for different requests. For example, in Fig. 3, the operator may specify that requests from subnet 10.1.1.0/24 (*e.g.*, s_1) have to traverse a SFC: Firewall-IDS-Proxy for security benefits. The DPC module will transform this specification to policy-level default path and install corresponding entries in the *SFC Table* of the ingress switch.

3) NF-level Default Path Construction: DPC first leverages classical algorithms (*e.g.*, OSPF or ECMP) to compute default path(s) from each switch to each NF. Each switch then stores the next-hop information on the default path to each NF in the *NF Table* so that each packet will be processed by the required NF(s) in sequence. We should note that even though one switch lies on more than one default path to each NF, it requires to install one NF entry and at most one group entry. Due to space limit, we omit the description of the group table installment in this paper. As a result, each switch will install $|N|$ entries in the *NF Table* for NF-level default paths construction.

4) Switch-level Default Path Construction: Similarly, DPC first leverages classical algorithms to compute default path(s) from each switch to each egress switch. The controller then let each switch to install switch-level wildcard rules in the *Flow Table* through Flow-Mod messages. Moreover, each egress switch has to install one rule for each connected destination. For example, in Fig. 3, switch v_1 installs two rules in *Flow Table:* one wildcard rule for egress switch v_3 and one rule for connected destination s_1.

B. Lightweight SFC Routing Update

With the help of multi-level default paths, requests will be forwarded to destinations while obeying SFC policies. Default paths help to save TCAM resources and relieve controller overhead, but they cannot guarantee the network performance (*e.g.*, NF/link load balancing or network throughput), due

to traffic dynamics. Thus, we design the Lightweight SFC Routing Update (LRU) module by joint default paths and per-request paths for network optimization.

1) Exploration of Feasible SFC Paths: Each request may have to traverse multiple NFs in sequence. The number of feasible SFC paths for each request may be exponential and the network performance will be affected by the selection of routing paths. Thus, we compute a set of feasible paths that satisfy SFC policy for each request. To decrease time complexity, we pre-compute the feasible SFC path set for each request only when topology changes. The feasible path set can be computed by traditional algorithms, such as depth-first search. If there are too many feasible paths, we may only choose a certain number (*e.g.*, 3-5) of best ones under a some performance criterion, such as having the large capacities or having the shortest number of hops. In this way, during the update process (Section V-B3), we can select one optimal SFC path from the feasible path set for each request.

2) Installment of A Feasible SFC Path: When the controller decides to re-route a request from its default path to another path p, under the traditional wisdom, the controller will deploy ω forwarding rules at every switch on p, where ω is the number of its appearance times on path p [1]. However, this scheme will cost many entries and lead to massive control overhead. To reduce the resource cost, we can leverage SAFE-ME to install default rules so as to improve network scalability. Since each ingress switch only maintains one SFC entry for each related request in *SFC Table*, we just consider the forwarding entry cost on the *Flow Table* and the *NF Table*.

Let variables $I^n(f,p,v)$ and $I^f(f,p,v)$ (both initialized to 0) denote the number of required NF entries and the number of required flow entries on switch v, respectively, as the route of request r is updated to the target path p. Assume that the request r has to traverse q NFs, denoted as $NF_1, NF_2, ..., NF_q$, respectively. We determine the values of $I^n(f,p,v)$ and $I^f(f,p,v)$ as follows: 1) We divide the path p into $q+1$ path segments (*i.e.*, source to NF_1, NF_1 to NF_2, ..., NF_q to egress switch). 2) We use p_d to denote each path segment on path p, where d is the destination of this path segment. For example, p_{NF_1} denotes the path segment from source to NF_1. 3) For each switch v on p_d, if path segment p_d overlaps with the default path from switch v to d, then there is no need to deploy an entry for this path segment on switch v; otherwise, a flow/NF entry on switch v for this path segment should be deployed. If d is an NF, $I^n(f,p,v) = I^n(f,p,v)+1$, which means an NF entry should be deployed on the *NF Table*. If d is a terminal, $I^f(f,p,v) = I^f(f,p,v)+1$, which means a flow entry should be deployed on the *Flow Table*. After traversing all path segments and all switches on path p, we obtain the values of variables $I^n(f,p,v)$ and $I^f(f,p,v)$.

3) Problem Definition for SFC Routing Update: We denote the set of switches as $V = \{v_1, ..., v_{|V|}\}$, the set of terminals as $U = \{u_1, ..., u_{|U|}\}$, and the set of NFs as $N = \{n_1, ..., n_{|N|}\}$. The data plane topology is modeled as a graph $G = (U \cup V \cup N, E)$, where E is the set of links. Let $c(e)$ (or $c(n)$) be the capacity of a link e (or an NF n) and $l(e)$ (or $l(n)$) be its current load. Note that these information can be obtained through OpenFlow [4] or other statistics collection mechanisms [26]. Since each middlebox is connected with a switch, the switch can also measure the middlebox load through port statistics collection.

When the network performance gets worse (*e.g.*, higher link/NF load ratio), the controller selects a subset Π of the largest requests (reported by the switches) for re-routing so as to achieve better network performance. The budget for re-routing execution time constraints the size of Π; more execution time budget means we can re-route more requests, which can be roughly estimated based on the past executions. The estimated rate of request $f \in \Pi$ is denoted as $r(f)$, which can be obtained through edge switches. Let $\mathcal{P}(f)$ be the set of feasible paths for request f. $\mathcal{P}(f)$ is determined based on the management policies and performance objectives, which has been discussed in Section V-B1. Note that, $\mathcal{P}(f)$ also contains the path $p^*(f)$ that the request is currently routed through.

Let $T^n(v)$ and $T^f(v)$ be the number of available entries in the *NF Table* and the *Flow Table*, respectively, at switch v. Let $I^n(f,p,v)$ (or $I^f(f,p,v)$) be the number of required NF entries (or flow entries) on switch v if path p is assigned to request f, which has been discussed in Section V-B2. Note that, the number of required SFC entries is related to the number of requests and is independent of update process. Thus, we do not consider the *SFC Table* constraint in here.

We formalize the load balancing routing (LBR-MBN) problem in middlebox networks as follows:

$$\min \quad \lambda$$
$$s.t. \begin{cases} b(e) = l(e) - \sum_{f\in\Pi: e\in p^*(f)} r(f), & \forall e \in E \\ b(n) = l(n) - \sum_{f\in\Pi: n\in p^*(f)} r(f), & \forall n \in N \\ \sum_{p\in\mathcal{P}(f)} y_f^p = 1, & \forall f \in \Pi \\ \sum_{f\in\Pi}\sum_{p\in\mathcal{P}(f): v\in p} y_f^p \cdot I^n(f,p,v) \le T^n(v), & \forall v \in V \\ \sum_{f\in\Pi}\sum_{p\in\mathcal{P}(f): v\in p} y_f^p \cdot I^f(f,p,v) \le T^f(v), & \forall v \in V \\ b(e) + \sum_{f\in\Pi}\sum_{p\in\mathcal{P}(f): e\in p} y_f^p r(f) \le \lambda \cdot c(e), & \forall e \in E \\ b(n) + \sum_{f\in\Pi}\sum_{p\in\mathcal{P}(f): n\in p} y_f^p r(f) \le \lambda \cdot c(n), & \forall n \in N \\ y_f^p \in \{0,1\}, & \forall p, f \\ \lambda \le 1. \end{cases} \tag{1}$$

where $y_f^p \in \{0,1\}$ means whether request f will be forwarded through path $p \in \mathcal{P}(f)$ or not. The first and second sets of equations compute the link background traffic load $b(e)$, $\forall e \in E$, and the NF background traffic load $b(n)$, $\forall n \in N$, when the flows in Π are taken out. The third set of equations requires that request $f \in \Pi$ is not splittable; it will be forwarded through a single path from $\mathcal{P}(f)$. The fourth set of inequalities describes the NF table size constraint, while the fifth set of inequalities describes the flow table size constraint on switches. The sixth and seventh sets of inequalities state the traffic load on each link e and each NF n, respectively, where λ is called as the network load ratio.

The optimization objective is determined by the users' requirement (*e.g.*, throughput maximization, load balancing). We choose network load ratio minimization, *i.e.*, $\min \lambda$, as the objective in this section for simplicity.

Theorem 1: LBR-MBN is an NP-hard problem.

We can show that the multi-commodity flow with minimum congestion problem [27] is a special case of our problem. Thus, the LBR-MBN problem is NP-Hard too. Due to space limit, we omit the detailed proof here.

4) Algorithm Design and Performance Analysis: We present an approximate algorithm, called Rounding-based SFC Routing Update (RBSU), to solve this problem. We first relax Eq. (1) by replacing the eighth line of integer constraints with $y_f^p \geq 0$, turning the problem into linear programming. We can solve it with a linear program solver (*e.g.*, CPLEX) and the solution is denoted by $\widetilde{y}$ and $\widetilde{\lambda}$. As the linear program is a relaxation of the LBR-MBN problem, $\widetilde{\lambda}$ is a lower-bound result for LBR-MBN. Using randomized rounding method [28], we obtain an integer solution $\widehat{y}_f^p$. More specifically, variable $\widehat{y}_f^p$ is set as 1 with the probability of $\widetilde{y}_f^p$. The RBSU algorithm is formally described in Algorithm 1.

Algorithm 1 RBSU: Rounding-based SFC Routing Update for Middlebox Networks

1: **Step 1: Solving the Relaxed LBR-MBN Problem**
2: Construct a linear program by replacing the integral constraints with $y_f^p \geq 0$
3: Obtain the optimal solution $\{\widetilde{y}_f^p\}$
4: **Step 2: Route Update for Middlebox Networks**
5: Derive an integer solution $\{\widehat{y}_f^p\}$ by randomized rounding
6: **for** each sampled flow $f \in \Pi$ **do**
7: **for** each SFC route $p \in \mathcal{P}(f)$ **do**
8: **if** $\widehat{y}_f^p = 1$ **then**
9: Appoint a path p for flow f

To analyze the proposed RBSU algorithm performance, we first assume that the minimum capacity of all the NFs and links is denoted by $c_{\min}$ and the whole flow set is denoted by Γ. We define a variable α as follows:

$$\alpha = \min\{\min\{\frac{\widetilde{\lambda} c_{\min}}{r(f)}, f \in \Gamma\}, \min\{T^n(v), T^f(v), v \in V\}\} \tag{2}$$

Lemma 2: RBSU can achieve the approximation factor of $\frac{\log n}{\alpha}+3$ for link capacity constraints in large networks, where n is the number of switches. Moreover, *the bound can be tightened to 2 in practice*.

Proof: We denote the traffic load of link $e \in E$ from flow $f \in \Gamma$ as $x_{f,e}$. Thus, the expected traffic load on e is

$$\begin{aligned}\mathbb{E}\left[\sum_{f\in\Gamma} x_{f,e}\right] &= \sum_{f\in\Pi}[x_{f,e}] + b(e) \\ &= \sum_{f\in\Pi}\sum_{e\in p: p\in\mathcal{P}(f)} \widetilde{y}_f^p \cdot r(f) + b(e) \leq \widetilde{\lambda} c(e)\end{aligned} \tag{3}$$

Combining Eq. (3) and the definition of α, we have

$$\begin{cases} \frac{x_{f,e}\cdot\alpha}{\widetilde{\lambda} c(e)} \in [0,1] \\ \mathbb{E}\left[\sum_{f\in\Gamma} \frac{x_{f,e}\cdot\alpha}{\widetilde{\lambda}\cdot c(e)}\right] \leq \alpha. \end{cases} \tag{4}$$

Thus, Chernoff bound [12] can be applied. Assume that ρ is a arbitrary positive value. It follows

$$\mathbf{Pr}\left[\sum_{f\in\Gamma} \frac{x_{f,e}\cdot\alpha}{\widetilde{\lambda}\cdot c(e)} \geq (1+\rho)\cdot\alpha\right] \leq e^{\frac{-\rho^2\cdot\alpha}{2+\rho}} \tag{5}$$

Now, we would assume that

$$\mathbf{Pr}\left[\sum_{f\in\Gamma} \frac{x_{f,e}}{\widetilde{\lambda}\cdot c(e)} \geq (1+\rho)\right] \leq e^{\frac{-\rho^2\cdot\alpha}{2+\rho}} \leq \frac{1}{n} \tag{6}$$

By solving Eq. (6), we have the following result

$$\rho \geq \frac{\log n + \sqrt{\log^2 n + 8\alpha\log n}}{2\alpha} \Rightarrow \rho \geq \frac{\log n}{\alpha} + 2 \tag{7}$$

In most practical scenarios, according to the definition of α, we can assume $\alpha \geq 3\log n$. Under this assumption, we have:

$$\rho \geq \frac{\log n + \sqrt{(\log n - 2\alpha)^2 - 4\alpha^2 + 12\alpha\log n}}{2\alpha} \Rightarrow \rho \geq 1 \tag{8}$$

Thus, the approximate factor for link capacity constraints is $\rho+1 = \frac{\log n}{\alpha}+3$. Under proper assumption (*i.e.*, $\alpha \geq 3\log n$), the bound can be tightened to $\rho+1=2$. ∎

Lemma 3: The proposed RBSU algorithm can achieve the bi-criteria approximation factor of $(\frac{\log n}{\alpha}+3, \frac{\log n}{\alpha}+3)$. Under proper assumption (*i.e.*, $\alpha \geq 3\log n$), the bound can be tightened to $(2,2)$. It means that RBSU can minimize the network load ratio to no more than $2\widetilde{\lambda}$ and the table size constraints are violated at most by a multiplicative factor 2.

The proof is similar to that of Lemma 2. Due to space limit, we omit the detailed proof here.

VI. Performance Evaluation

In this section, we evaluate the scalability and efficiency of our SAFE-ME system. All our code has been publicly available at github[1].

A. Performance Metrics and Benchmarks

Performance Metrics: We adopt the following three sets of metrics to evaluate the scalability and efficiency of our proposed system. 1) SAFE-ME involves tag operations, which may increase the packet transmission delay and decrease the end-to-end throughput. Thus, we adopt *end-to-end delay* and *end-to-end throughput* to evaluate the efficiency of tag operations. Specifically, we use Ping and Qperf [29] tools to measure the delay of ICMP and TCP/UDP protocols between two terminals, respectively. In our implementation, some flows are aggregated into one request. We use Packet Generator (PktGen) tool [30] to measure its flow completion time (FCT). Besides, we adopt vnStat tool [31] to measure the end-to-end throughput, which can evaluate the negative impact of tag operations on the packet forwarding rate. 2) Considering traffic dynamics (*e.g.*, request intensity fluctuation), we need re-route flows to better deal with traffic dynamics (*i.e.*, execute the RBSU algorithm). During the update process, we focus on two metrics: *update delay* and *control traffic overhead*. Specifically, we measure the duration of the update procedure as *update delay*. Moreover, we record the total traffic amount

[1]https://github.com/sdntest/Middlebox_Routing.

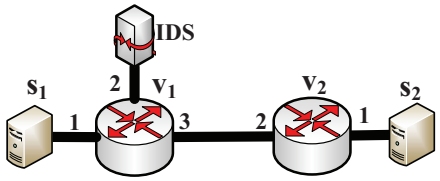

Fig. 5: Topology of the pica8 testbed. The testbed consists of two pica8 3297 switches, one VNF (IDS) and two servers (s_1, s_2). We generate traffic, from s_1 to s_2, which has to traverse IDS.

	Ping Delay	Iperf3 throughput	Qperf Delay (TCP)	Qperf Delay (UDP)
PDA	1.25 ms	814 Mbps	513 us	429 us
SIMPLE	1.27 ms	812 Mbps	519 us	436 us
SAFE-ME	1.316 ms	809 Mbps	530 us	449 us

TABLE II: Tag operations only cost $< 5\%$ performance loss.

between the control plane and the data plane during the update procedure as *control traffic overhead*. Obviously, lower update delay and control traffic overhead represent better network update performance. After route update, we measure the *total number of required entries* on three tables of each switch and the *link/NF Load* on each link/NF. Accordingly, we can obtain the maximum value and CDF performance of these metrics. 3) Network failure is a common scenario in today's networks. Thus, we measure the *failure response time* to deal with various network failures, such as single/multiple NF/link/switch failures. We measure the duration from failure occurrence to failure recovery as *failure response time*.

Benchmarks: We compare SAFE-ME with other two benchmarks for evaluation. The first benchmark is the most related work, SIMPLE [1], which is an SDN-based policy enforcement layer to simplify middlebox traffic steering. To account for both the middlebox processing capacity constraint and the TCAM table size constraint, SIMPLE first pre-computes several feasible physical sequences for each request while tackling the switch resource constraints, and then chooses a physical sequence for each request to minimize the maximum middlebox load. The second benchmark is an online algorithm, called primal-dual-update-algorithm (PDA) [10]. PDA achieves the trade-off optimization between the throughput competitiveness and QoS requirements under both link and NF capacity constraints. Note that, due to the inapplicability of traditional default path solutions as shown in Section II-A, we have not found prior work in this direction. Thus, we decided to compare SAFE-ME with SIMPLE and PDA, two solutions that schedule and forward traffic at granularity of requests (as shown in Table I).

B. System Implementation with Pica8 and Evaluations

As described in Section IV-C, although *shift* operation is high-speed for ASIC [25], some commodity switch chips may not fully support this function. Under this situation, we can leverage NF to fulfill *shift* operation. This section shows that tag operations in SAFE-ME are lightweight for NF processing.

As shown in Fig. 5, the servers (s_1 and s_2) and VNF are connected to the Pica8 3297 switches [32] through 1Gbps links. The VNF is an open source IDS, called Snort [33], running on a server with a core i5-3470 processor and 8GB of RAM. To test the efficiency of tag operations on NFs, we route traffic from s_1 to s_2 using three different solutions. (1)

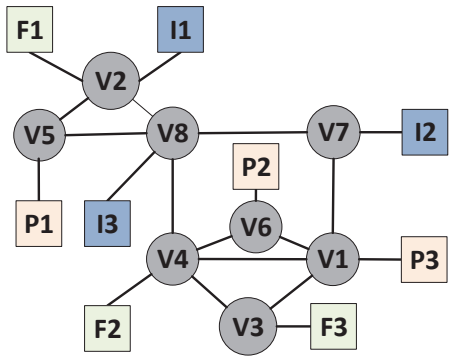

Fig. 6: Telstra Topology for small-scale testing. Circles represent switches and squares represent NFs. It contains 8 switches (from V_1 to V_8), 3 firewalls (from F_1 to F_3), 3 IDSs (I_1 to I_3) and 3 proxies (from P_1 to P_3).

PDA forwards traffic through path $s_1 - v_1 - \text{IDS} - v_1 - v_2 - s_2$ with the help of inport information. (2) SIMPLE leverages simple tag operations to forward requests through path $s_1 - v_1$ $(add\ tags)$ $-$ IDS $- v_1 - v_2$ $(delete\ tags) - s_2$. (3) Our proposed SAFE-ME method implements *shift* operation on VNF to forward requests through path $s_1 - v_1$ $(add\ tags)$ $-$ IDS $(shift\ operation) - v_1 - v_2$ $(delete\ tags) - s_2$. We can see that the tag operations of SAFE-ME are the most complex, while those of PDA (without tag operations) are the simplest, among three algorithms.

We use Ping to test the icmp delay, leverage Iperf3 tool [34] to test the maximum end-to-end throughput, and adopt Qperf tool [29] to test the UDP/TCP delay. The testing results are listed in Table II. Due to space limit, we omit the detailed description here. Overall, although SAFE-ME contains some tag operations (*e.g.*, *shift* operation), it still achieves similar end-to-end delay/throughput performance (less than $< 5\%$) compared with both PDA and SIMPLE. For example, the results show that SAFE-ME (809Mbps) only decreases end-to-end throughput by about 0.4% and 0.6% compared with SIMPLE (812Mbps) and PDA (814Mbps), respectively. Thus, we can conclude that tag operations in SAFE-ME are lightweight, which is also testified in the next section (*i.e.*, Figs. 7-8).

C. Small-scale Experiments with Open vSwitches

Experimental Settings: In this section, we implement SAFE-ME with the popular Open vSwitch (OVS, version 2.8.5) [23] on a small-scale topology Telstra from the Rocketfuel dataset [35], as depicted in Fig. 6. Since the topology does not provide NF information, we utilize VNF mechanism [20] to deploy three types of NFs (*i.e.*, Firewall, IDS, and Proxy) and place 3 units for each type of NF for simplicity. In other words, we deploy total 3×3=9 NFs on the Telstra topology. Each OVS and its connected NF(s) are running on a single server with a core i5-3470 processor and 16GB of RAM. Besides, we use RYU [36] as the controller software running on another server with a core i7-8700k and 32GB of RAM.

We use Packet Generator (PktGen) [30] to generate network traffic, which is a powerful tool also used by [37] [38]. By using PktGen, we can generate requests with various sizes and patterns, and collect FCT, load information through PktGen APIs. In the experiments, we generate DCTCP (datacenter TCP) pattern requests [30]. All requests have to traverse either

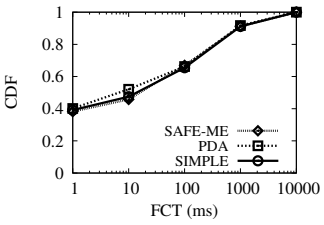

Fig. 7: CDF vs. FCT on Telstra

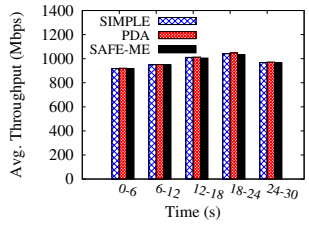

Fig. 8: Avg. Throughput vs. Time on Telstra

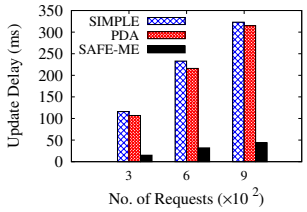

Fig. 9: Update Delay vs. No. of Requests on Telstra

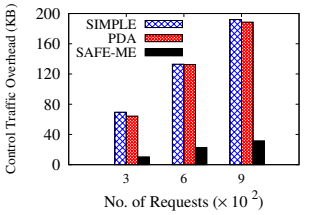

Fig. 10: Control Traffic Overhead vs. No. of Request on Telstra

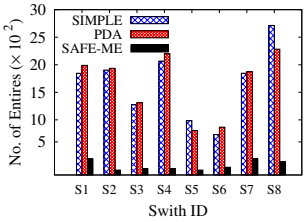

Fig. 11: No. of Entries on Each Switch on Telstra

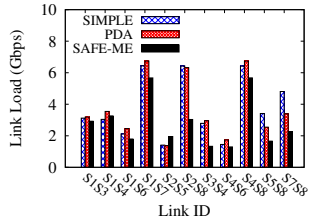

Fig. 12: Link Load of Each Link on Telstra

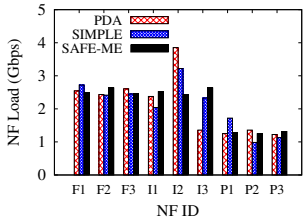

Fig. 13: NF Load of Each NF on Telstra

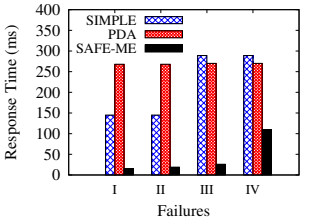

Fig. 14: Response Time vs. Failures on Telstra [2]

Firewall-IDS-Proxy or *Firewall-IDS*.

FCT and Throughput Performance: In the first set of experiments, we generate TCP requests with a duration of 30s and measure the FCT and end-to-end throughput. Note that, PktGen provides the FCT information of all requests. The end-to-end throughput can be derived by the vnStat tool [31] through measuring the average throughput of one server port per 6-second interval. The results in Figs. 7-8 show that our proposed SAFE-ME system achieves similar FCT and throughput performance compared with other two algorithms. That means the tag operations of SAFE-ME are lightweight.

Update Performance: In the second set of experiments, we conduct the traffic dynamics, which require to dynamically adjust routing paths and update forwarding entries for load balancing. In Fig. 7, FCT of nearly 50% requests is less than 10ms. Thus, if we update routing paths at a low speed, the network performance will be greatly reduced. Figs. 9-10 show that SAFE-ME can reduce update delay and control traffic overhead by about 85% and 83%, respectively, compared with other two solutions. Lower update delay and control traffic overhead can make the network more reliable and robust during update procedure. SAFE-ME can achieve lower update delay because it greatly reduces the number of required entries by about 85% for updating compared with other algorithms, as shown in Fig. 11. The rule installation speed is about 0.4ms/rule on OVS, which causes the huge update delay gap for these methods. Note that, the rule installation speed of a physical switch (*e.g.*, 50.25ms/rule on HP 5130 switches [17]) is substantially much slower than that of OVS, which means the gap of update delay is even larger among these solutions on the physical platform. Figs. 12-13 show link/NF load conditions for these three systems. Using Alg. 1, SAFE-ME achieves better link load balancing and similar NF load balancing compared with SIMPLE/PDA. For example, by Fig. 12, SAFE-ME reduces the maximum link load by about 14% and 18% compared with SIMPLE and PDA, respectively. Note that, we can also tweak RBSU to work for the other two schemes and obtain a similar link/NF load balancing performance. However, without the support of the SAFE-ME's data plane, both SIMPLE and PDA would still require a higher number of flow entries, causing larger control overhead and longer update delays, similar to what is shown in Figs. 9-11, even after adopting RBSU.

Dealing with Failures: The network may encounter switch/NF/link failures in practice. We consider four failure scenarios on the Telstra topology: (I) single-NF failure, (II) single-link/switch failure, (III) multi-NF failures, and (IV) multi-link/switch failures. Under all four scenarios, the controller should re-route requests and we focus on the failure response delay to reconfigure the network. When network failure occurs, the controller needs to be aware of failures, compute new rules and install them on switches. For single NF/link/switch failure, PDA costs much time to compute and install new rules. SIMPLE pre-computes pruned sets for the single NF failure scenario. Thus, the time cost is mainly for installing rules on switches in SIMPLE. SAFE-ME only adjusts fewer affected SFC entries to embed requests with other available NFs (*e.g.*, nearest available NFs). The number of updated entries for route update of SAFE-ME is less than that of other two benchmarks. As a result, the failure response time of SAFE-ME is much short. The results in Fig. 14 show that SAFE-ME can reduce the failure response time by about 93% and 87% compared with PDA and SIMPLE, respectively, for single NF/link/switch failure. For multi-NF failures, SAFE-ME only needs to modify affected SFC entries to redirect requests to other available NFs. However, both PDA and SIMPLE will cost much time to re-compute and install rules. As a result, SAFE-ME reduces failure response time by about 90% compared with other two benchmarks for multi-NF failures. For multi-link/switch failures, SAFE-ME re-computes default paths and installs them. Since the number of affected entries of SAFE-ME is far less than that of other two solutions, SAFE-ME reduces response time by about 59% and 61% compared with PDA and SIMPLE, respectively, for multiple links/switches failures.

[2]I: single-NF failure, II: single-link/switch failure, III: multi-NF failures, IV: multi-link/switch failures

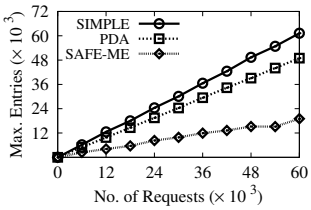

Fig. 15: Max. No. of Entries vs. No. of Requests on Ebone

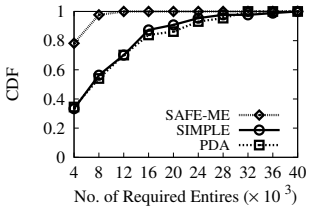

Fig. 16: CDF vs. No. of Required Entries on Ebone

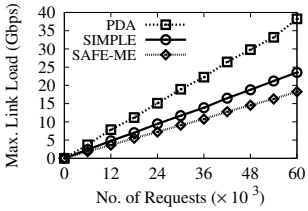

Fig. 17: Max. Link Load vs. No. of Requests on Ebone

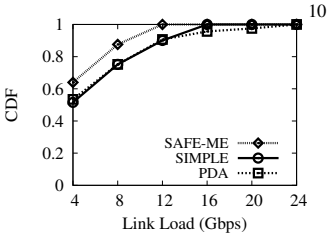

Fig. 18: CDF vs. Link Load on Ebone

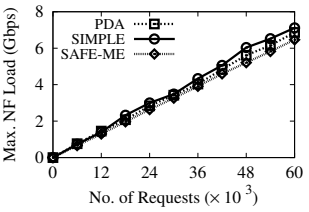

Fig. 19: Max. NF Load vs. No. of Requests on Ebone

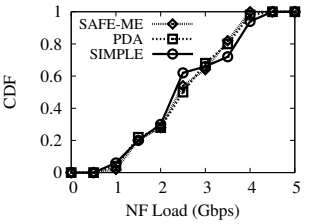

Fig. 20: CDF vs. NF Load on Ebone

D. Large-scale Simulations

Simulation Settings: In the large-scale simulations, we use packet traces of our campus network, which is shared at dropbox[3]. We simulate the traces across the Rocketfuel project [35], called Ebone, which contains 87 switches and 348 servers. Since this topology does not provide any NF information, similar to small-scale experiments, we adopt VNF placement scheme [20] to deploy 5 types of NFs (*i.e.*, Firewall, IDS, IPSec, Proxy and WAN-opt) and the number of each type of NF is set as 10 by default. In other words, we deploy totally 5×10=50 NFs on the Ebone topology. There exist four SFCs (*i.e.*, FW-IDS-IPSec, FW-Proxy, FW-IDS-IPSec-WAN-opt and IDS-Proxy), and each request will be assigned with one of SFC requirements. Note that, since the campus network is different from Ebone, we use a gravity model to map requests to ingress/egress switches [35]. We execute each simulation 50 times and average the numerical results.

Flow Entry Resource: We first compare the required entry resources of these three systems. In Fig. 15, with the increasing number of requests, the maximum number of required entries increases for all systems. In comparison, the proposed SAFE-ME system uses much fewer entries than other two solutions. For example, when there are 36×10^3 requests, SAFE-ME uses a maximum number of 11,900 entries among all switches, while SIMPLE and PDA use 36,500 and 29,500 entries, respectively; SAFE-ME needs 2,600 entries on average, while both SIMPLES and PDA need about 9,000 entries (not shown due to space limit). In other words, SAFE-ME can reduce the maximum number of required entries by about 68% and 60% compared with SIMPLE and PDA, respectively. Meanwhile, SAFE-ME reduces the average number of required entries by about 71% compared with the other two solutions. Fig. 16 shows the CDF of the number of entries under a fixed number (*e.g.*, 36×10^3) of requests. We observe that about 2.2% of switches need more than 8,000 entries by SAFE-ME, while over 45% of switches need more than 8,000 entries by SIMPLE and PDA.

[3]dropbox.com/s/f6wl5zyyfmqq4ry/flow_trace.pcap?dl=0.

Bandwidth Resource: Figs. 17-18 give the comparisons of bandwidth resource consumption for these algorithms. We claim that SAFE-ME can save bandwidth resources through well-designed routing strategy. For example, when there are 36×10^3 requests, our proposed algorithm can reduce the maximum/average link load by about 23%/28% and 51%/30% compared with SIMPLE and PDA, respectively (not shown average performance due to space limit). Fig. 18 shows the CDF of link load ratio under a fixed number (*e.g.*, 36×10^3) of requests. We observe that over 64% of links undertake load less than 4Gbps while only 53% (or 51%) of links undertake load less than 4Gbps by SIMPLE (or PDA).

NF processing Resource: Figs. 19-20 show the comparisons of NF loads for different algorithms. From these two figures, we observe that SAFE-ME can achieve similar NF load performance compared with both SIMPLE and PDA. Note that, since we assume all requests can be served by needed NFs, the average NF loads of these solutions are the same.

From these simulation results, we can draw some conclusions. First, from Figs. 15-16, SAFE-ME reduces the number of required entries by about 70% on average compared with other two solutions for serving all requests in the network. Second, from Figs. 17-18, SAFE-ME reduces the link load by about 30% on average compared with SIMPLE and PDA. Finally, from Figs. 19-20, we believe SAFE-ME can achieve similar NF load compared with SIMPLE and PDA, which consume more entry and bandwidth resources than SAFE-ME.

VII. Conclusion

Scalability is a critical challenge in middlebox networks due to the routing complexity and traffic dynamics. We proactively deploy multi-level default paths so that requests can be forwarded to destination while obeying SFC policy with less resource (*e.g.*, TCAM) consumption. We further study the joint optimization of default path and per-request routing to update the SFC routing paths. With the help of default paths, we only need modify fewer rules when encountering traffic dynamics or link/switch/NF failures.

VIII. Acknowledgement

We thank our shepherd, Prof. Geoffrey Xie, and the anonymous reviewers for their suggestions. This research of Zhao, Xu, Liu, Ge and Huang is partially supported by the National Science Foundation of China (NSFC) under Grants 61822210, U1709217, and 61936015; by Anhui Initiative in Quantum Information Technologies under No. AHY150300. The research of Qian is partially supported by National Science Foundation (NSF) Grant 1750704.

REFERENCES

[1] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM computer communication review*, vol. 43, no. 4. ACM, 2013, pp. 27–38.

[2] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[3] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 539–550.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[5] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 175–180.

[6] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on sdn network utilization," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 1734–1742.

[7] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, and Y. Sun, "Minimizing controller response time through flow redirecting in sdns," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 1, pp. 562–575, 2018.

[8] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. ACM, 2009, pp. 202–208.

[9] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[10] L. Guo, J. Pang, and A. Walid, "Dynamic service function chaining in sdn-enabled networks with middleboxes," in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*. IEEE, 2016, pp. 1–10.

[11] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 24–24.

[12] G. Zhao, H. Xu, S. Chen, L. Huang, and P. Wang, "Joint optimization of flow table and group table for default paths in sdns," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1837–1850, 2018.

[13] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags." in *NSDI*, vol. 14, 2014, pp. 543–546.

[14] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 486–494.

[15] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 731–741.

[16] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: a software-defined framework for developing, deploying, and managing network functions," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 511–524.

[17] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. Maguire Jr, "Metron: Nfv service chains at the true speed of the underlying hardware," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. USENIX Association, 2018.

[18] O. N. Foundation *et al.*, "Openflow version 1.3.4," 2014.

[19] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 98–106.

[20] T. Lukovszki, M. Rost, and S. Schmid, "It's a match!: Near-optimal and incremental middlebox deployment," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 1, pp. 30–36, 2016.

[21] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "Nfp: Enabling network function parallelism in nfv," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 43–56.

[22] M. C. Luizelli, D. Raz, and Y. Sa'ar, "Optimizing nfv chain deployment through minimizing the cost of virtual switching," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2150–2158.

[23] OVS. (2018) Open vswitch: open virtual switch. [Online]. Available: http://openvswitch.org/

[24] B. Switches. (2014). [Online]. Available: https://www.barefootnetworks.com

[25] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An asic implementation of the aes sboxes," in *Cryptographers Track at the RSA Conference*. Springer, 2002, pp. 67–78.

[26] H. Xu, Z. Yu, C. Qian, X.-Y. Li, Z. Liu, and L. Huang, "Minimizing flow statistics collection cost using wildcard-based requests in sdns," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 6, pp. 3587–3601, 2017.

[27] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*. IEEE, 1975, pp. 184–193.

[28] P. Raghavan and C. D. Tompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[29] Qperf. (2018). [Online]. Available: https://github.com/linux-rdma/qperf

[30] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ecn in multi-service multi-queue data centers." in *NSDI*, 2016, pp. 537–549.

[31] vnStat. (2018). [Online]. Available: https://humdi.net/vnstat/

[32] Pica8. (2014) Pica8 p3297 switches. [Online]. Available: https://www.pica8.com/wp-content/uploads/pica8-datasheet-48x1gbe-p3297.pdf

[33] Snort. (2019). [Online]. Available: http://www.snort.org

[34] iperf3. (2016). [Online]. Available: https://iperf.fr/

[35] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[36] Ryu. (2017). [Online]. Available: https://osrg.github.io/ryu/

[37] G. Chen, Y. Lu, Y. Meng, B. Li, K. Tan, D. Pei, P. Cheng, L. Luo, Y. Xiong, X. Wang *et al.*, "Fast and cautious: Leveraging multi-path diversity for transport loss recovery in data centers." in *USENIX Annual Technical Conference*, 2016, pp. 29–42.

[38] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "Clicknp: Highly flexible and high performance network processing with reconfigurable hardware," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 1–14.