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

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

***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.**

Manuscript received 23 August 2021; revised 19 December 2021; accepted 4 April 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Hay. Date of publication 21 April 2022; date of current version 17 October 2022. This work was supported in part by the National Science Foundation of China under Grant 62102392, Grant 62132019, and Grant 61936015; in part by the National Science Foundation of Jiangsu Province under Grant BK20210121; and in part by the Open Research Projects of Zhejiang Laboratory under Grant 2022QA0AB04. Some preliminary results of this paper were published in the Proceedings of IEEE ICNP 2019 [1] [DOI:10.1109/ICNP.2019.8888123]. *(Corresponding author: Gongming Zhao.)*

Hongli Xu, Gongming Zhao, Jianchun Liu, and Liusheng Huang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Suzhou Institute for Advanced Study, Suzhou University of Science and Technology, Suzhou, Jiangsu 215123, China (e-mail: xuhongli@ustc.edu.cn; gmzhao@ustc.edu.cn; lyl617@mail.ustc.edu.cn; lshuang@ustc.edu.cn).

Peng Xi is with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, also with the Suzhou Institute for Advanced Study, Suzhou University of Science and Technology, Suzhou, Jiangsu 215123, China, and also with the School of Educational Science, Anhui Normal University, Wuhu, Anhui 241000, China (e-mail: xipeng@mail.ahnu.edu.cn).

Chen Qian is with the Department of Computer Science and Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064 USA (e-mail: cqian12@ucsc.edu).

Digital Object Identifier 10.1109/TNET.2022.3167169

I. 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, although SRAM is capable for SFC routing, the switches often use TCAM-based forwarding tables to achieve over 20× faster rule lookup or wildcard rule matching speed [6]. However, TCAMs are 400× more expensive and consume 100× more power per Mbit than the RAM-based storage on switches [7]. 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]. 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 [10]. 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 [10], [11].
- *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 [12]) of flows using only a limited size of TCAM-based forwarding entries, especially with SFC requirement. Some SDN switches

(*e.g.*, Noviswitch [13]) adopt RAM-based flow tables for flow forwarding. However, RAM-based flow tables may significantly increase the lookup latency [8]. In addition, a large number of flow entries may increase the delay for switches to modify/match entries [9], [14].

Though the traditional solutions, *e.g.*, default paths [15], 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 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 solve complex SFC routing, several works have designed efficient solutions for MB networks [2], [4], [16], [17]. 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 [18]), there are millions of ingress-egress switch pairs [18], [19]. 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 VI. 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 [12]. 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 [10].

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 II 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 III. Sections IV and V 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 VI. We conclude this paper in Section VII.

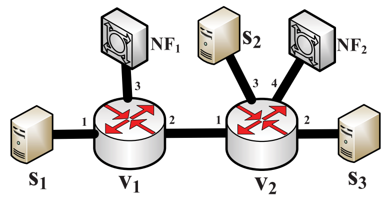


Fig. 1. Traffic from s_1 to s_3 has to traverse NF_1 and traffic from s_2 to s_3 has to traverse NF_1 and NF_2. Different requests with the same egress switch (or destination) will traverse different SFCs. However, the switch-based (or destination-based) default path solution cannot distinguish the traffic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

II. 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 II-A. Then we review the existing SDN-based SFC routing solutions and analyze their limitations in Section II-B.

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 [15], [20]). 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

TABLE I
COMPARISON OF THE ADVANTAGES AND DISADVANTAGES OF EXISTING SOLUTIONS

Schemes	Number of Rules	Control Overhead	Satisfy SFC Policy	Network Performance
Traditional Default Path (*e.g.*, [15] [20] [21] [22])	Few	Low	No	Low
Per-request Routing (*e.g.*, [2] [4] [16] [17])	Many	High	Yes	High
Our Scheme	**Few**	**Low**	**Yes**	**High**

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 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.

B. 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], [23], [24]. 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 today's switch capabilities or result in huge energy consumption and long update delay [8]. 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 VI.

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** [25]–[28] [29], [30]. For example, CoMB [27] 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 [25], MiddleClick [26], and OpenBox [28] also merge similar packet processing elements into one. Slick [29] and SNF [30] 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 IV-E.

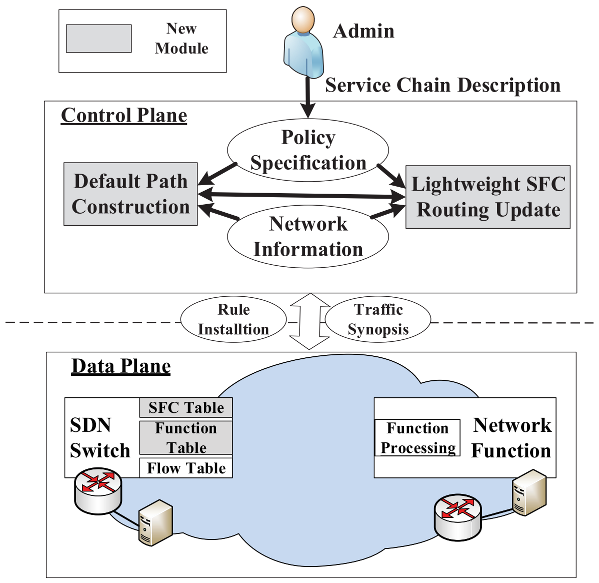


Fig. 2. System overview. In the data plane, each switch maintains three tables: SFC table, NF table and Flow table. The control plane consists of two new modules, default path construction and lightweight SFC routing update, to implement routing strategies.

From Table I, 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.

III. 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 III-A, describe the packet processing procedure in Section III-B, and give an example to illustrate SAFE-ME in Section III-C.

A. System Overview

As shown in Fig. 2, SAFE-ME consists of data plane and control plane designs. The proposed architecture addresses the

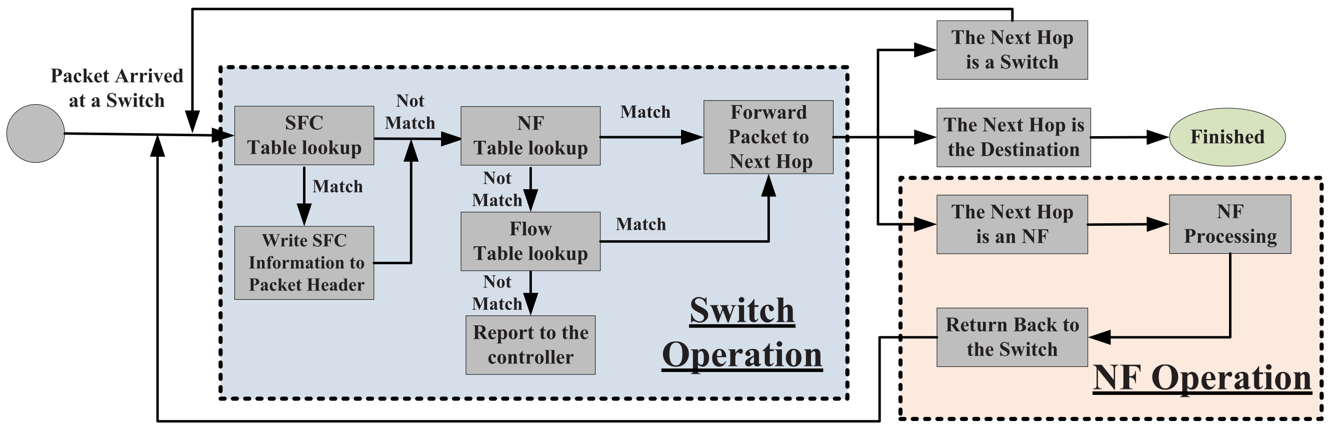


Fig. 3. Illustration of packet processing procedure. When a packet arrives at a switch, the switch matches the header with SFC Table, NF Table and Flow Table in sequence. In this way, the packet will be forwarded to destination while obeying SFC constraints.

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. Each NF unit processes the received packets. As specified in the OpenFlow standard [31], 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 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 V.

B. 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 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 matching the *Flow Table*.

C. 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

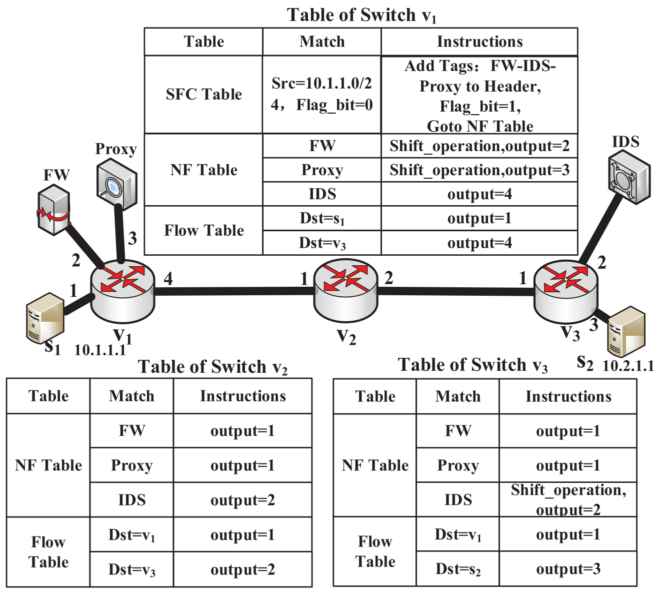


Table of Switch v_1

Table	Match	Instructions
SFC Table	Src=10.1.1.0/24，Flag_bit=0	Add Tags：FW-IDS-Proxy to Header, Flag_bit=1, Goto NF Table
NF Table	FW	Shift_operation,output=2
	Proxy	Shift_operation,output=3
	IDS	output=4
Flow Table	Dst=s_1	output=1
	Dst=v_3	output=4

Table of Switch v_2

Table	Match	Instructions
NF Table	FW	output=1
	Proxy	output=1
	IDS	output=2
Flow Table	Dst=v_1	output=1
	Dst=v_3	output=2

Table of Switch v_3

Table	Match	Instructions
NF Table	FW	output=1
	Proxy	output=1
	IDS	Shift_operation, output=2
Flow Table	Dst=v_1	output=1
	Dst=s_2	output=3

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 $- v_1 - v_2 - v_3 - s_2$."

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 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 *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 II-B, 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.

IV. Date Plane Design

As described in Section III-B, 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 [32] switches. Through these operations, SAFE-ME can forward requests using fewer rules and lower control overhead while satisfying SFC policies.

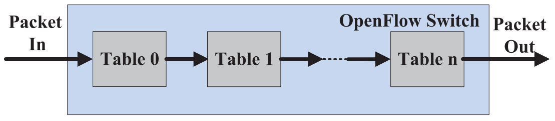


Fig. 5. Pipeline processing in OpenFlow switches. Each switch contains multiple tables, which are sequentially numbered. Packets can match theses tables in sequence.

A. Pipeline Process in OpenFlow Switches

As specified by the OpenFlow standard [31], the pipeline of every OpenFlow switch consists of multiple forwarding tables. Each table contains a certain number of entries. Many commercial switches (*e.g.*, Pica8 3297 switches [33], barefoot switches [34]) 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.

B. Tag Embedding Through SFC Table

In the data plane, there may exist many units of NFs. The controller uses unique identifies (*e.g.*, $1, 2, \ldots, m$) to distinguish these NFs. Recent studies show that the number of NFs is similar to the number of switches [35], [36] 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 design a high-level SAFE-ME header format, 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,

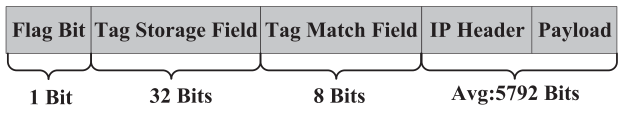


Fig. 6. The high-level design of the SAFE-ME header format. Tag storage/match fields and flag bit in a packet. Flag bit field indicates whether the packet has been embedded a tag or not. Tag Match Field stores the first NF in the SFC and tag storage field embeds the remaining NF(s) in the SFC. The overall bandwidth cost for embedding tags is negligible.

we use 1 flag bit to denote whether the packet has been embedded a tag or not. This header format may be slightly adjusted according to the specific implementation platform. Note that, in data center networks, the average packet size is around 724 bytes (*i.e.*, 5792 bits) [37]. 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 [38] (*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 0×01, 0×02, 0×03 to denote the FW, Proxy, IDS units, respectively. In this way, the service function chain can be encoded into 0×01-0×03-0×02. To embed this tag, *Tag Match Field* records the first NF (*e.g.*, 0×01) and *Tag Storage Field* stores the remaining NFs in the reverse order (*e.g.*, 0×0203). In other words, the SFC entry on switch v_1 can be expressed as "$ip\_src = 10.0.1.0/24, Flag\_bit = 0, actions = \{Tag\_Match\_Field = 0 \times 01, Tag\_Storage\_Field = 0 \times 0203, Flag\_bit = 1, Goto\_Table : NF\_Table\}$". 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 0×01, *Tag Storage Field* to 0×0203 and set the *Flag Bit* to 1. As a result, the SFC policy is successfully embedded in the packet header.

C. 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.*, 0×01). There is a matching NF entry: "$Tag\_Match\_Field = 0 \times 01, actions = shift\_right, output = 2$", 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 0×03 and 0×02, which means the packet still has to traverse two NFs (*i.e.*, 0×03-0×02). When the packet returns to switch v_1 from FW, the switch will match the NF table with the match field "$Tag\_Match\_Field = 0 \times 03$", and forward to IDS through port 4. In the end, the packet will be forwarded to the destination while obeying SFC policy.

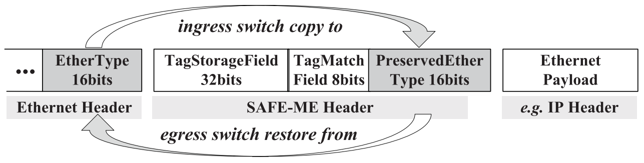


Fig. 7. The SAFE-ME header format in P4 switches.

D. SAFE-ME Design With P4 Switches

Compared to using existing protocol headers, adding a customized new SAFE-ME header can make the SAFE-ME implementation more applicable and flexible. Thus, we realize our design with a highly programmable data plane consisting of the *Programming Protocol-independent Packet Processors* (P4) [32] switches as a case study. Leveraging the flexibility of the P4$_{16}$ programming language [39], SAFE-ME can easily be implemented with P4 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 "0×0123" 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 "0×0123" 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.

E. Discussions

Applicability for the Consolidated Platform. NF consolidation (*e.g.*, consolidating several NFs onto a server with containers [40]) reduces the TCAM cost and control overhead through simplifying the SFC processing in middlebox networks [25], [27], [28]. 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 [41], P4-based barefoot switches [34]), 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], [23]. Meanwhile, the *shift* operation is a basic and high-speed function [42]. Thus, the tag operations in SAFE-ME can achieve line rate if the switch supports *shift* operation (*e.g.*, Open vSwitches [41], barefoot switches [34]). 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 [23] 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 [24], [43] or consolidated platform [28], [44] 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 I. 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 [45]. 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 [46] 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 [47]. Both SRv6 and SAFE-ME are based on source routing and tag-based forwarding mechanisms, but SAFE-ME is explicitly designed for scalable and flexible middlebox routing. Specifically, SAFE-ME takes advantage of the programmable data plane for scalable and flexible SFC routing using specialized header formats and high-speed label shifting operations. Meanwhile, by installing flow rules with different priorities on switches, SAFE-ME can realize dynamic and flexible rerouting without changing the content of the packets, which meets complex middlebox routing requirements.

V. Control Plane Design

A. 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 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) 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.

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.

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.

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 VI-B.

B. 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.

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 [48] 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 V-B.3), we can select one optimal SFC path from the feasible paths 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 feasible path p, under the traditional wisdom, the controller will deploy one forwarding rule for each switch that appears on path p. Thus, totally ω forwarding rules are deployed, 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 $NF_1, NF_2, \ldots, NF_q$, 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 $NF_2, \ldots, NF_q$ 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 $I^f(f,p,v)$.

3) Problem Definition for SFC Routing Update: We denote the set of switches as $V = \{v_1, \ldots, v_{|V|}\}$, the set of servers as $U = \{u_1, \ldots, u_{|U|}\}$, and the set of NFs as $N = \{n_1, \ldots, n_{|N|}\}$. 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 [49] or other statistics collection mechanisms [50], [51]. 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 V-B.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 V-B.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:

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

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 [52]. Thus, the LSRU-MB problem is NP-Hard too. □

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 [53]) 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 [54], 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 LSRU-MB Problem**
2: Construct a linear program by replacing the integral constraints with $y_f^p \ge 0$
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 $f$
10:    end if
11:  end for
12: end for
```

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 V-B.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, $x_{f_1,e}, x_{f_2,e}, \ldots$ are mutually independent. The expected traffic load on link e is:

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

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

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

Thus, Chernoff bound [15] can be applied. Assume that ρ is an arbitrary positive value. It follows

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

Now, we would assume that

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

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

$$\rho \ge \frac{\log n + \sqrt{\log^2 n + 8\alpha\log n}}{2\alpha}, \quad n \ge 2$$
$$\Rightarrow \rho \ge \frac{\log n + \sqrt{(2\log n + 2\alpha)^2}}{2\alpha} = \frac{3\log n}{2\alpha} + 1, \; n \ge 2 \quad (7)$$

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.*, $\alpha \ge 3\log n$, the proposed RBSU algorithm can achieve the approximation factor of 2 for link/NF capacity constraints.

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

$$\rho \ge \frac{\log n + \sqrt{(\log n - 2\alpha)^2 - 4\alpha^2 + 12\alpha\log n}}{2\alpha}, \; n \ge 2$$
$$\Rightarrow \rho \ge \frac{\log n + \sqrt{(\log n - 2\alpha)^2}}{2\alpha} \Rightarrow \rho \ge 1 \quad (8)$$

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 V-B.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.*, $\alpha \ge 3\log n$, 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.*, $\alpha \ge 3\log n$), 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.

VI. Performance Evaluation

In this section, we evaluate the scalability and efficiency of SAFE-ME. All code is publicly available on Github.[1] We first give the metrics and benchmarks for performance comparison (*Section VI-A*). We then perform a small-scale test with P4 switches and give the performance analysis (*Section VI-B*). Finally, we evaluate the performance of several middlebox systems with large-scale simulations (*Section VI-C*). 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.

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 [55] 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 (Pkt-Gen) tool [56] to measure its flow completion time (FCT). Besides, we adopt vnStat tool [57] 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

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

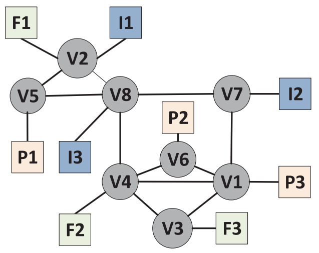


Fig. 8. Telstra topology. Circles represent switches and squares represent NFs.

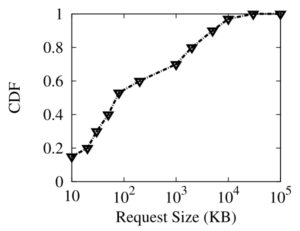


Fig. 9. CDF vs. request size generated by packet generator on Telstra.

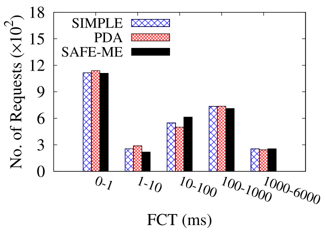


Fig. 10. No. of requests vs. FCT on Telstra.

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 (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 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.

B. 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 consisting of switches based on P4. The P4 software switches named with *simple_switch_grpc* base on the *behavioral model version 2* (bmv2) [58] are employed to construct our small-scale topology called Telstra from the Rocketfuel dataset [59], as depicted in Fig. 8.

Experimental Settings: Since the topology Telstra does not provide NF information, we utilize VNF mechanism [36] 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) [60], 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 [61] 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 [56] to generate network traffic, which is a software based traffic generator also used by [62], [63]. 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 [56] 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 [57] 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. They 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. This lower update delay is enabled by the reduced number of required

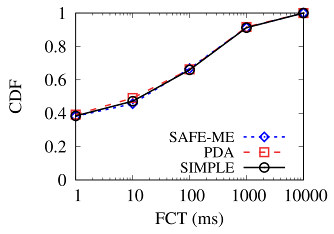


Fig. 11. CDF of FCT on Telstra.

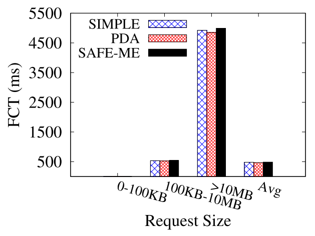


Fig. 12. FCT vs. request size on Telstra.

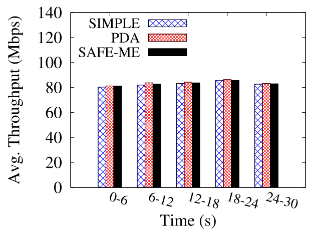


Fig. 13. Avg. throughput vs. time on Telstra.

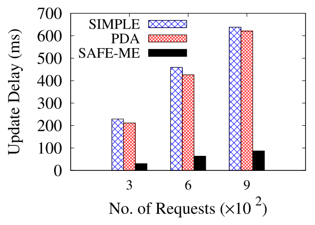


Fig. 14. Update delay vs. no. of requests on Telstra.

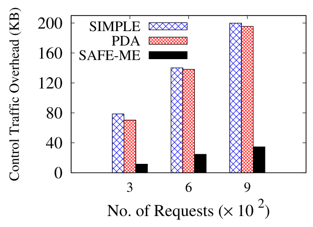


Fig. 15. Control traffic overhead vs. no. of request on Telstra.

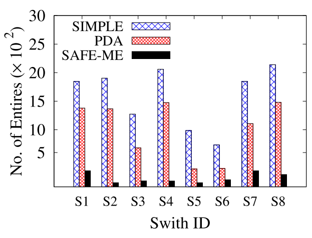


Fig. 16. No. of entries on each switch on Telstra.

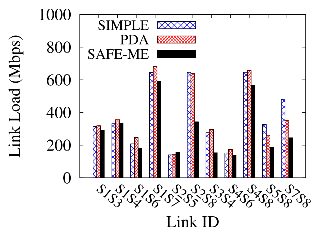


Fig. 17. Link load of each link on Telstra.

entries (by about 89%) for updating compared to 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.

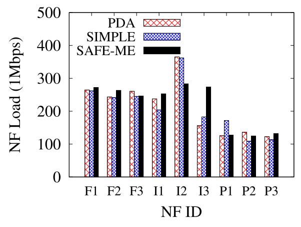


Fig. 18. NF load of each NF on Telstra.

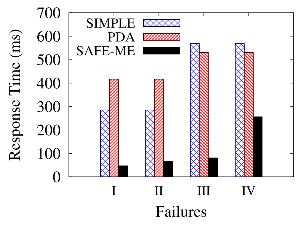


Fig. 19. Response time vs. failures on Telstra[2].

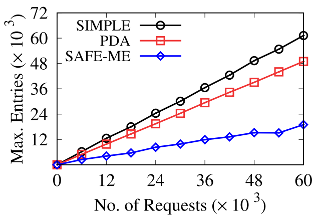


Fig. 20. Max. no. of entries vs. no. of requests on Ebone.

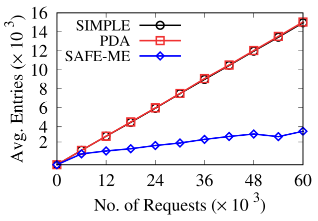


Fig. 21. Avg. no. of entries vs. no. of requests on Ebone.

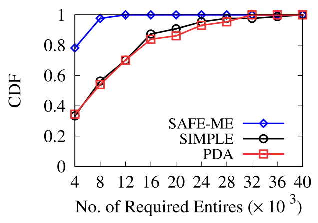


Fig. 22. CDF vs. no. of required entries on Ebone.

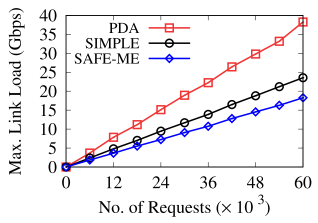


Fig. 23. Max. link load vs. no. of requests on Ebone.

C. 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 Github.[3] We simulate the traces across the Rocketfuel project [59], 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 [36] 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 [59]. 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

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

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

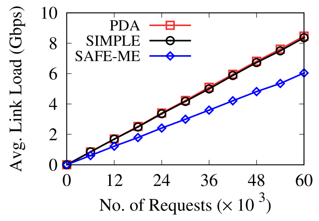


Fig. 24. Avg. link load vs. no. of requests on Ebone.

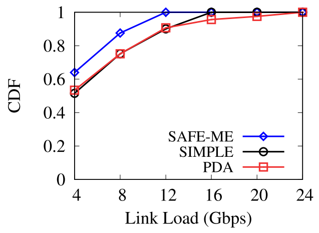


Fig. 25. CDF vs. link load on Ebone.

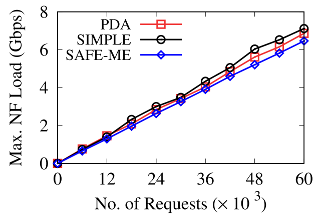


Fig. 26. Max. NF load vs. no. of requests on Ebone.

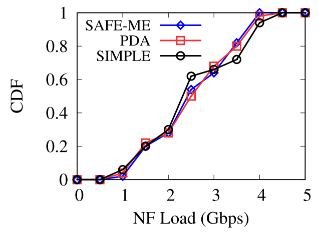


Fig. 27. CDF vs. NF load on Ebone.

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. 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.

VII. 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 *Proc. 27th IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2019, pp. 1–11.

[2] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, Aug. 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 *Proc. 24th IEEE Int. Conf. Netw. Protocols (ICNP)*, Nov. 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. Service Manage.*, vol. 18, no. 2, pp. 1415–1429, Jun. 2021.

[6] K. Kannan and S. Banerjee, "Flowmaster: Early eviction of dead flow on SDN switches," in *Proc. Distrib. Comput. Netw. 15th Int. Conf. (ICDCN)*. Berlin, Germany, Springer, 2014, pp. 484–498.

[7] N. P. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2014, pp. 175–180.

[8] 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 Distrib. Syst.*, vol. 32, no. 3, pp. 728–742, Mar. 2021.

[9] H. Song, S. Guo, P. Li, and G. Liu, "FCNR: Fast and consistent network reconfiguration with low latency for SDN," *Comput. Netw.*, vol. 193, Jul. 2021, Art. no. 108113.

[10] 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, Feb. 2018.

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

[12] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements analysis," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*, 2009, pp. 202–208.

[13] Noviflow. (2020). *Noviswitch 21100 White Paper*. Accessed: Apr. 10, 2022. [Online]. Available: https://noviflow.com/wp-content/uploads/2019/11/NoviSwitch-21100-Datasheet-400_V5.pdf

[14] J. Liu *et al.*, "Incremental server deployment for software-defined NFV-enabled networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 1, pp. 248–261, Feb. 2021.

[15] 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, Aug. 2018.

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

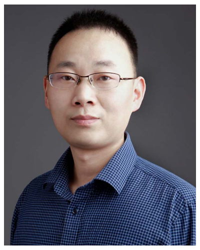
[17] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest path and maximum flow problems under service function chaining constraints," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 2132–2140.

[18] NTT. (2021). *NTT Hong Kong Financial Data Center*. Accessed: Apr. 10, 2022. [Online]. Available: https://datacenter.hello.global.ntt/sites/default/files/apac_hongkong_brochure_jun2020_.pdf

[19] L. Fang *et al.*, "Hierarchical SDN for the hyper-scale, hyper-elastic data center and cloud," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res. (SOSR)*, 2015, pp. 7:1–7:13.

[20] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Architectures, Protocols Comput. Commun.*, 2011, pp. 254–265.

[21] J. Zhou *et al.*, "WCMP: Weighted cost multipathing for improved fairness in data centers," in *Proc. 9th Eurosys Conf.*, 2014, pp. 5:1–5:14.
[22] 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, Feb. 2018.
[23] 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 *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2014, pp. 543–546.
[24] 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 *Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 731–741.
[25] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. Maguire, "Metron: NFV service chains at the true speed of the underlying hardware," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 171–186.
[26] T. Barbette, C. Soldani, R. Gaillard, and L. Mathy, "Building a chain of high-speed VNFs in no time," in *Proc. IEEE 19th Int. Conf. High Perform. Switching Routing (HPSR)*, Jun. 2018, pp. 1–8.
[27] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. 9th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2012, pp. 323–336.
[28] A. Bremler-Barr, Y. Harchol, and D. Hay, "OpenBox: A software-defined framework for developing, deploying, and managing network functions," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 511–524.
[29] B. Anwer, T. Benson, N. Feamster, and D. Levin, "Programming slick network functions," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, Jun. 2015, p. 14.
[30] G. P. Katsikas, M. Enguehard, M. Kuźniar, G. Q. Maguire, and D. Kostić, "SNF: Synthesizing high performance NFV service chains," *PeerJ Comput. Sci.*, vol. 2, p. e98, Nov. 2016.
[31] O. N. Foundation. (2014). *OpenFlow Switch Specification Version 1.3.5*. Accessed: Apr. 10, 2022. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf
[32] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.
[33] Pica8. (2014). *Pica8 P3297 Switches*. Accessed: Apr. 10, 2022. [Online]. Available: https://www.pica8.com/wp-content/uploads/pica8-datasheet-48x1gbe-p3297.pdf
[34] *Barefoot-Networks*. (2014). *Barefoot Tofino Switches*. Accessed: Apr. 10, 2022. [Online]. Available: https://www.barefootnetworks.com
[35] 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 *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2015, pp. 98–106.
[36] T. Lukovszki, M. Rost, and S. Schmid, "It's a match!: Near-optimal and incremental middlebox deployment," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, pp. 30–36, Jan. 2016.
[37] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling network function parallelism in NFV," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 43–56.
[38] M. C. Luizelli, D. Raz, and Y. Sa'ar, "Optimizing NFV chain deployment through minimizing the cost of virtual switching," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 2150–2158.
[39] P4.org. (2021). *P4$_{16}$-Language-Specification*. Accessed: Apr. 10, 2022. [Online]. Available: https://p4lang.github.io/p4-spec/docs/P4-16-v1.2.2.html
[40] Z. Meng, J. Bi, H. Wang, C. Sun, and H. Hu, "CoCo: Compact and optimized consolidation of modularized service function chains in NFV," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–7.
[41] OVS. (2018). *Open vswitch: Open Virtual Switch*. Accessed: Apr. 10, 2022. [Online]. Available: http://openvswitch.org/
[42] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC implementation of the AES sboxes," in *Proc. Cryptographers' Track RSA Conf.* Berlin, Germany, Springer, 2002, pp. 67–78.
[43] L. Guo, J. Z. T. Pang, and A. Walid, "Joint placement and routing of network function chains in data centers," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Honolulu, HI, USA, Apr. 2018, pp. 612–620.
[44] 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, Jun. 2018.
[45] D. E. Sarmiento, A. Lebre, L. Nussbaum, and A. Chari, "Decentralized SDN control plane for a distributed cloud-edge infrastructure: A survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 1, pp. 256–281, 2021.
[46] Z. N. Abdullah, I. Ahmad, and I. Hussain, "Segment routing in software defined networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 464–486, 1st Quart., 2019.
[47] IETF. (2020). *IPv6 Segment Routing Header (SRH)*. Accessed: Apr. 10, 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8754.txt
[48] E. Q. V. Martins and M. M. B. Pascoal, "A new implementation of Yen's ranking loopless paths algorithm," *Quart. J. Belgian, French Italian Oper. Res. Soc.*, vol. 1, no. 2, pp. 121–133, 2003.
[49] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.
[50] H. Xu, Z. Yu, C. Qian, X.-Y. Li, 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, Dec. 2017.
[51] 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, Dec. 2018.
[52] S. A. Amiri, K.-T. Foerster, R. Jacob, and S. Schmid, "Charting the algorithmic complexity of waypoint routing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 1, pp. 42–48, Jan. 2018.
[53] *V12. 1: User's Manual for Cplex*, vol. 46, no. 53, I. I. CPLEX, Int. Bus. Mach. Corp., Armonk, New York, NY, USA, 2009, p. 157.
[54] 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.
[55] J. George. (2018). *Qperf*. Accessed: Apr. 10, 2022. [Online]. Available: https://github.com/linux-rdma/qperf
[56] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in *Proc. SPIE13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2016, pp. 537–549.
[57] T. Toivola. (2018). *VnStat*. Accessed: Apr. 10, 2022. [Online]. Available: https://humdi.net/vnstat/
[58] (2015). P4.org. *Behavioral-Model Version 2*. Accessed: Apr. 10, 2022. [Online]. Available: https://github.com/p4lang/behavioral-model
[59] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proc. Conf. Appl., Technol., architectures, protocols for Comput. Commun. - SIGCOMM*, 2002, pp. 133–145.
[60] (2021). P4·org. *P4-Compiler*. Accessed: Apr. 10, 2022. [Online]. Available: https://github.com/p4lang/p4c
[61] Google. (2021). *GRPC*. Accessed: Apr. 10, 2022. [Online]. Available: https://www.grpc.io/
[62] G. Chen *et al.*, "Fast and cautious: Leveraging multi-path diversity for transport loss recovery in data centers," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2016, pp. 29–42.
[63] B. Li *et al.*, "ClickNP: Highly flexible and high performance network processing with reconfigurable hardware," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 1–14.



Hongli Xu (Member, IEEE) received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China (USTC), China, in 2002 and 2007, respectively. He is currently a Professor with the School of Computer Science and Technology, USTC. He has published more than 100 articles in famous journals and conferences, including IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the International Conference on Computer Communications (INFOCOM), and the International Conference on Network Protocols (ICNP). He has held more than 30 patents. His research interests include software-defined networks, edge computing, and the Internet of Things. He was awarded the Outstanding Youth Science Foundation of NSFC in 2018. He has won the best paper award or the best paper candidate in several famous conferences.

Peng Xi received the B.S. degree in computer science and the M.S. degree in computer software and theory from the University of Science and Technology of China in 2004 and 2010, respectively, where he is currently pursuing the Ph.D. degree with the School of Computer Science. He is currently a Lecturer with the School of Educational Science, Anhui Normal University (AHNU). His current research interests include software-defined networks and wireless networks.

Gongming Zhao (Member, IEEE) received the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2020. He is currently an Associate Professor with the University of Science and Technology of China. His current research interests include software-defined networks and cloud computing.

Jianchun Liu (Member, IEEE) received the B.S. degree from North China Electric Power University in 2017. He is currently pursuing the Ph.D. degree with the School of Data Science, University of Science and Technology of China (USTC). His main research interests are software-defined networks, networks function virtualization, edge computing, and federated learning.

Chen Qian (Senior Member, IEEE) received the B.Sc. degree in computer science from Nanjing University in 2006, the M.Phil. degree in computer science from The Hong Kong University of Science and Technology in 2008, and the Ph.D. degree in computer science from The University of Texas at Austin in 2013. He is currently an Assistant Professor with the Department of Computer Science and Engineering, University of California at Santa Cruz. He has published more than 60 research papers in highly competitive conferences and journals. His research interests include computer networking, networks security, and the Internet of Things. He is a member of ACM.

Liusheng Huang (Senior Member, IEEE) received the M.S. degree in computer science from the University of Science and Technology of China in 1988. He is currently a Senior Professor and a Ph.D. Supervisor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or coauthored six books and over 300 journal/conference papers. His research interests are in the areas of the Internet of Things, vehicular ad-hoc networks, information security, and distributed computing.