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

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

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

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

I. Introduction

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

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

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

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

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

978-1-7281-2700-2 / 19/$31.00 ©2019 IEEE

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

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

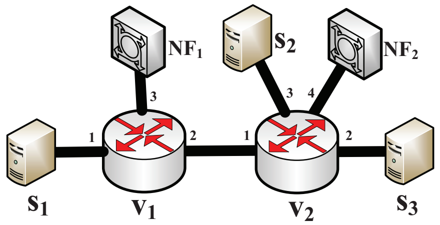

Fig. 1: Traffic from s_1 to s_3 has to traverse NF_1 and traffic from s_2 to s_3 has to traverse NF_1 and NF_2. Different requests with the same egress switch (or destination) will traverse different SFCs. However, the switch-based (or destination-based) default path solution cannot distinguish the traffic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

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

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

II. Background and Motivation

A. Inapplicability of Traditional Default Path Solutions

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

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

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

B. Limitations of Prior Work

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

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

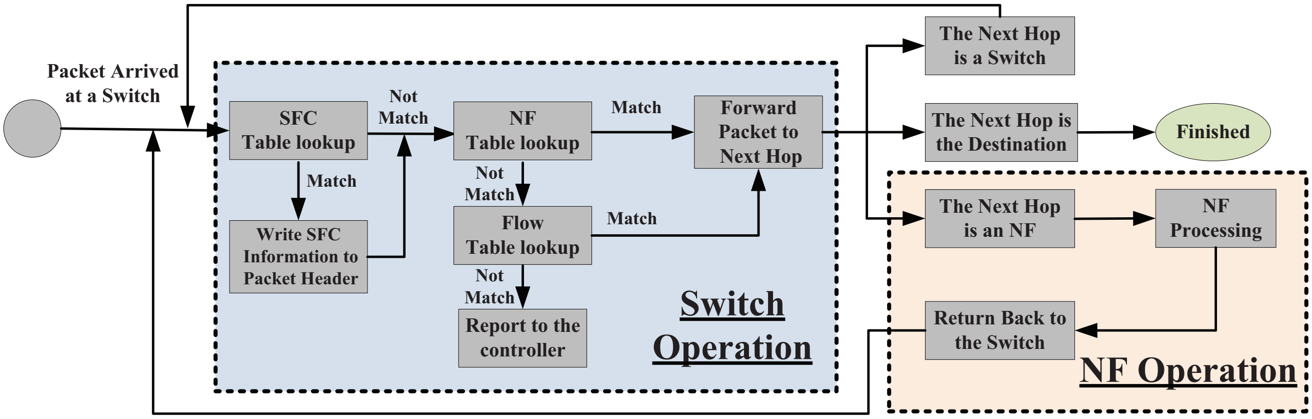

Fig. 2: Illustration of Packet Processing Procedure. When a packet arrives at a switch, the switch matches the header with SFC Table, NF Table and Flow Table in sequence. In this way, the packet will be forwarded to destination while obeying SFC constraints.

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

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

III. System Architecture

A. System Overview

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

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

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

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

B. Packet Processing Procedure

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

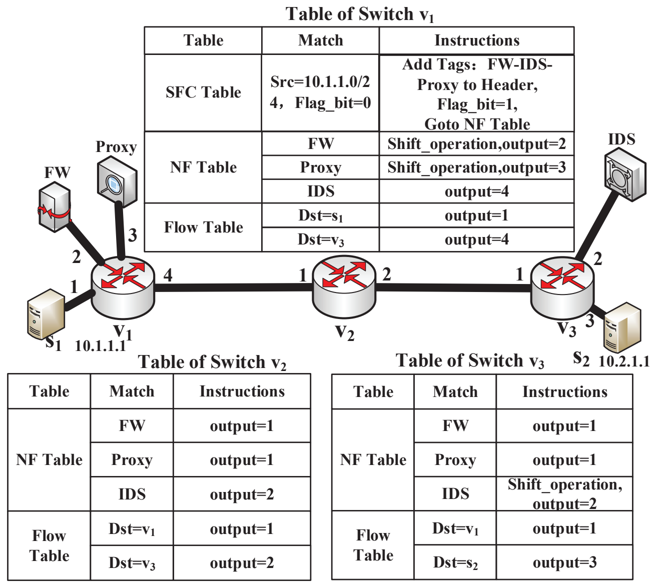

Table of Switch v_1

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

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

C. Illustration of SAFE-ME Design

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

When a request from subnet 10.1.1.0/24 arrives at the ingress switch, v_1 will match this packet header with the *SFC table*, and write the tag (*i.e.*, the SFC information: “FW-IDS-Proxy”) into the packet header. The packet will then be matched with the *NF Table* (*i.e.*, “match=FW”). Since v_1 is connected with a firewall, this switch executes shift operation (which will be introduced in Section IV) to delete the “FW-” information in the tag and then forwards this packet to the firewall through outport 2. After the packet is processed by the firewall and returns to switch v_1, this switch will match the NF entry “match=IDS and output=4”, and forward this packet to switch v_2, which will then forward it to v_3 by the *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 “$s_1 - v_1 - \text{FW} - v_1 - v_2 - v_3 - \text{IDS} - v_3 - v_2 - v_1 - \text{Proxy} - v_1 - v_2 - v_3 - s_2$”. That means, we only need one special SFC entry on the ingress switch for this request, and the other entries are shared by different requests. Consequently, SAFE-ME will greatly reduce the use of rules and the control overhead.

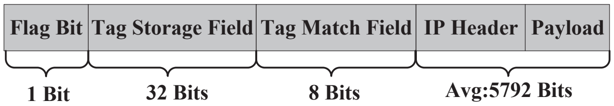

Fig. 4: Tag Storage/Match Fields and Flag Bit in a packet. Flag Bit Field indicates whether the packet has been embedded a tag or not. Tag Match Field stores the first NF in the SFC and Tag Storage Field embeds the rest NF(s) in the SFC. The overall bandwidth cost for embedding tags is negligible.

IV. Date Plane Design

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

A. Tag Embedding through SFC Table

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

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

When a packet from subnet 10.0.1.0/24 arrives at switch v_1, it will match the entry in the SFC Table. The actions will modify the *Tag Match Field* to 0x01, *Tag Storage Field* to 0x0203 and set the *Flag Bit* to 1. As a result, the SFC information is successfully embedded in the packet header.

B. Tag Shifting through NF Table

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

C. Discussions

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

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

V. Control Plane Design

A. Default Path Construction (DPC) for SFC Routing

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

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

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

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

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

B. Lightweight SFC Routing Update

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Now, we would assume that

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

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

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

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

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

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

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

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

VI. Performance Evaluation

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

A. Performance Metrics and Benchmarks

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

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

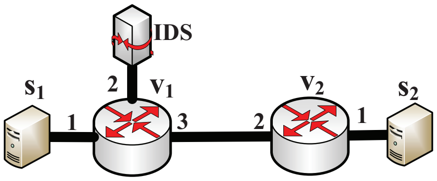

Fig. 5: Topology of the pica8 testbed. The testbed consists of two pica8 3297 switches, one VNF (IDS) and two servers (s_1, s_2). We generate traffic, from s_1 to s_2, which has to traverse IDS.

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

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

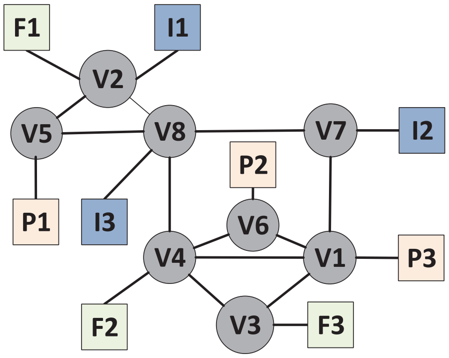

Fig. 6: Telstra Topology for small-scale testing. Circles represent switches and squares represent NFs. It contains 8 switches (from V_1 to V_8), 3 firewalls (from F_1 to F_3), 3 IDSs (I_1 to I_3) and 3 proxies (from P_1 to P_3).

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

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

B. System Implementation with Pica8 and Evaluations

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

As shown in Fig. 5, the servers (s_1 and s_2) and VNF are connected to the Pica8 3297 switches [32] through 1Gbps links. The VNF is an open source IDS, called Snort [33], running on a server with a core i5-3470 processor and 8GB of RAM. To test the efficiency of tag operations on NFs, we route traffic from s_1 to s_2 using three different solutions. (1) PDA forwards traffic through path $s_1 - v_1 - \text{IDS} - v_1 - v_2 - s_2$ with the help of inport information. (2) SIMPLE leverages simple tag operations to forward requests through path $s_1 - v_1\ (add\ tags) - \text{IDS} - v_1 - v_2\ (delete\ tags) - s_2$. (3) Our proposed SAFE-ME method implements *shift* operation on VNF to forward requests through path $s_1 - v_1\ (add\ tags) - \text{IDS}\ (shift\ operation) - v_1 - v_2\ (delete\ tags) - s_2$. We can see that the tag operations of SAFE-ME are the most complex, while those of PDA (without tag operations) are the simplest, among three algorithms.

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

C. Small-scale Experiments with Open vSwitches

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

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

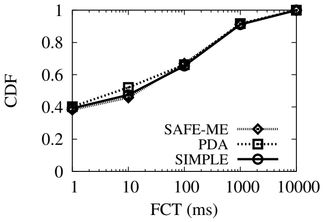

Fig. 7: CDF vs. FCT on Telstra

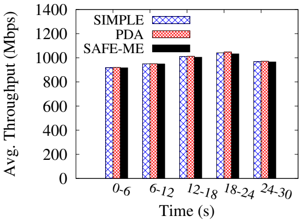

Fig. 8: Avg. Throughput vs. Time on Telstra

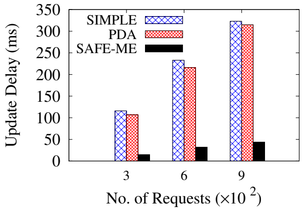

Fig. 9: Update Delay vs. No. of Requests on Telstra

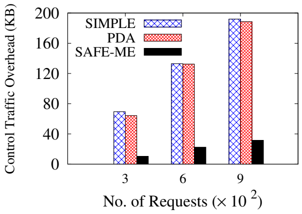

Fig. 10: Control Traffic Overhead vs. No. of Request on Telstra

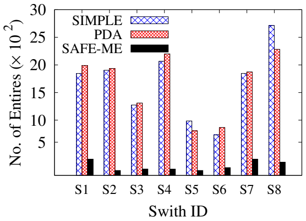

Fig. 11: No. of Entries on Each Switch on Telstra

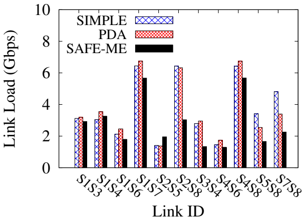

Fig. 12: Link Load of Each Link on Telstra

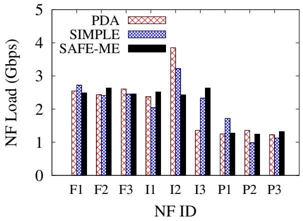

Fig. 13: NF Load of Each NF on Telstra

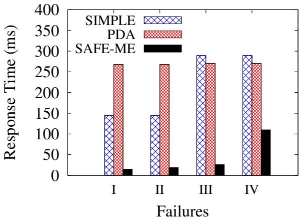

Fig. 14: Response Time vs. Failures on Telstra [2]

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

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

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

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

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

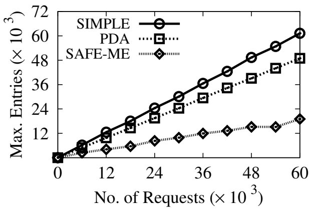

Fig. 15: Max. No. of Entries vs. No. of Requests on Ebone

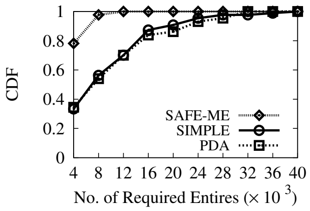

Fig. 16: CDF vs. No. of Required Entries on Ebone

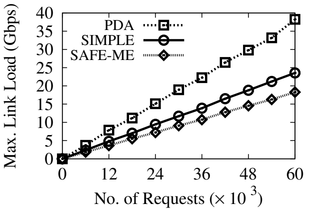

Fig. 17: Max. Link Load vs. No. of Requests on Ebone

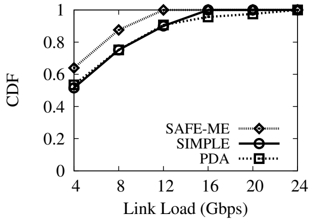

Fig. 18: CDF vs. Link Load on Ebone

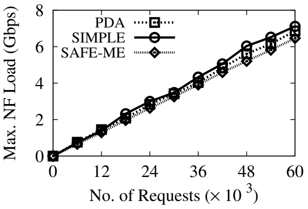

Fig. 19: Max. NF Load vs. No. of Requests on Ebone

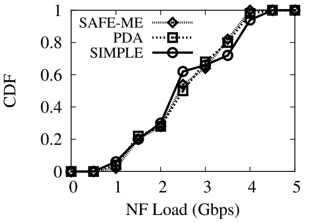

Fig. 20: CDF vs. NF Load on Ebone

D. Large-scale Simulations

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

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

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

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

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

VII. Conclusion

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

VIII. Acknowledgement

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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