Network-Wide Forwarding Anomaly Detection and Localization in Software Defined Networks

Peng Zhang, *Member, IEEE*, Fangzheng Zhang, Shimin Xu, Zuoru Yang, Hao Li, *Member, IEEE*, Qi Li, *Senior Member, IEEE*, Huanzhao Wang, Chao Shen, *Senior Member, IEEE*, and Chengchen Hu, *Member, IEEE*

***Abstract*—A crucial requirement for Software Defined Network (SDN) is that data plane forwarding behaviors should always agree with control plane policies. Such requirement cannot be met when there are *forwarding anomalies*, where packets deviate from the paths specified by the controller. Most anomaly detection methods for SDN install dedicated rules to collect statistics of each flow, and check whether the statistics conform to the "flow conservation principle". We find these methods have a limited detection scope: they look at one flow each time, thus can only check a small number of flows simultaneously. In addition, dedicated rules for statistics collection can impose a large overhead on flow tables of SDN switches. To this end, this paper presents FOCES, a network-wide forwarding anomaly detection and localization method in SDN. Different from previous methods, FOCES applies a new kind of flow conservation principle at network wide, and can check forwarding behaviors of *all* flows in the network simultaneously, without installing any dedicated rules. Finally, FOCES applies a voting-based method to localize malicious switches when anomalies are detected. Experiments with four network topologies show that FOCES can achieve a detection precision higher than 90%, when the packet loss rate is no larger than 10%, and a localization accuracy of around 80% when the packet loss rate is no larger than 5%.**

***Index Terms*—Software defined network, forwarding anomaly, detection, localization.**

I. Introduction

SOFTWARE defined networking (SDN) promises a centralized, flexible, and programmable control of computer networks [2]. In a typical SDN, a logically-centralized controller compiles network policies (*e.g.*, routing, access control, waypoint traversal) into forwarding rules, and installs them at switches through a standard control channel (*e.g.*, OpenFlow [2]). Switches enforce these forwarding rules to realize the network policies. A common and cost-effective way to realize SDN is to use white-box switches [3], [4] running third-party switch Operation Systems (OSes) [5]–[7].

Despite its huge benefits, SDN is still vulnerable to attacks. Many real incidents indicate that switches and routers can be compromised [8]–[10], and SDN is even more prone to switch compromise [11]–[13]. A recent study shows that an attacker can hack the boot loader of switch OSes, so as to gain persistent control over SDN switches, even after the switch OSes have been reinstalled [11], [14].

In addition, the control channel between the controller and switches also lacks security protection. For OpenFlow, SSL/TLS becomes optional rather than mandatory after version 1.0 [15], and many SDN switch vendors just forgo this feature. Benton *et al.* reported that only 2 out of 8 switches, and 1 out of 8 controllers have full support of SSL/TLS [16].

The above security vulnerabilities often manifest at the data plane as *forwarding anomalies*, *i.e.*, packets deviate from the paths that are specified by the controller. Forwarding anomalies can cause violation of critical security policies, like flow isolation and waypoint traversal. For example, the control plane policy may require a specific flow go through a firewall, while a forwarding anomaly can cause some or all packets of this flow bypass the firewall. Note that SDN by itself provides no means to detect forwarding anomalies due to its open-loop control mode: the controller only sends rule installation messages to switches, while cannot ensure switches will correctly install these rules, and forward according to these rules.

Recently, many *data plane debugging tools* are proposed to test whether flow rules at switches are corresponding to the controller's view [17]–[19], or monitor whether the forwarding behaviors of packets are compliant with the control plane policies [20], [21]. However, they assume switches are trustable, thus cannot be used to detect forwarding anomalies as switches can be compromised. Some tools have been proposed to detect forwarding anomalies in SDN, and we broadly group them into the following two classes.

Path verification tools let switches along the forwarding path embed a cryptographic tag (*e.g.*, MAC) so that the controller can verify whether the actual paths took by packets are agreeing with what the controller expects [22]–[25]. However, these tools need to modify switches to support cryptographic operations, which makes them not easy to be deployed. In addition, they need extra header space for storing cryptographic information, which can introduce high bandwidth overhead. For

Manuscript received July 13, 2018; revised January 26, 2020 and August 12, 2020; accepted October 12, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Guan. Date of publication November 2, 2020; date of current version February 17, 2021. This work was supported by the National Natural Science Foundation of China under Grant 61772412, Grant 61702407, Grant 61572278, Grant 61822309, Grant 61703301, and Grant U1736205. A preliminary version of this article has been published in the Proceedings of IEEE ICDCS 2018. *(Corresponding author: Hao Li.)*

Peng Zhang, Hao Li, and Chao Shen are with the Ministry of Education (MOE) Key Laboratory for Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: p-zhang@xjtu.edu.cn; hao.li@xjtu.edu.cn; chaoshen@xjtu.edu.cn).

Fangzheng Zhang, Shimin Xu, and Huanzhao Wang are with the School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China.

Zuoru Yang was with the School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China. He is now with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Qi Li is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: qli01@tsinghua.edu.cn).

Chengchen Hu was with the School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China. He is now with Xilinx Labs Asia Pacific, Singapore 486040.

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TNET.2020.3033588

example, the bandwidth overhead of REV is 5.28% when the average packet size is 757 Bytes, and the overhead can be even higher for smaller packets [24]. *Statistics verification tools* try to detect forwarding anomalies by passively collecting flow statistics, and check whether the statistics conform to the flow conservation principle, *i.e.*, the counters along the forwarding path of a flow should be roughly the same [26]–[28]. In contrast to path verification tools, statistics verification tools do not require any extra header space or switch modification, and thus can be easily adopted by production networks. However, rules in SDNs can have wildcard match fields (*e.g.*, a destination IP prefix matching a set of flows), and thus the counter of a flow rule can aggregate multiple flows. As a result, these tools either cannot handle wildcard rules [26], or have to install dedicated *counter rules* for collecting flow statistics [27], [28], therefore placing large overhead on flow table space. In addition, these tools only check the consistency of counters for the set of flows being selected, and may miss some forwarding anomalies happening to unselected flows [26]–[28]. How to detect forwarding anomalies of *all* flows in presence of wildcard rules without installing dedicated counter rules is still an open problem.

To overcome the above limitations, this paper presents *FlOw-Counter Equation System (FOCES)*, a new approach to forwarding anomaly detection and localization for SDN. Generally speaking, FOCES belongs to statistics verification methods. However, different from all the above methods that apply the flow conservation principle for each *individual* flow, FOCES works at a network-scale: it takes *all* flows as a whole, and checks whether their counters are consistent with the controller's view. Specifically, FOCES models the controller's view (*i.e.*, expected forwarding behaviors) with *Flow-Counter Matrix (FCM)*, which captures the relationship between all flows and rules in the network. Then, FOCES periodically collects the counters of all rules in the network, and checks whether they can fit into what we call the *flow-counter equation system* determined by the FCM. In this sense, FOCES generalizes the flow conservation principle from a single flow to a network of flows, and thus can detect and localize forwarding anomalies at the network-wide scale.

When designing FOCES, we are faced with the following challenges. First, noises like packet losses may cause FOCES to falsely flag the network under forwarding anomaly. We show how to eliminate such false positives by designing a threshold-based detection algorithm. Second, since FOCES detects forwarding anomalies by taking all flows as a whole, it is difficult to localize malicious switches that are accountable, especially when there are packet losses. We design a set of metrics for measuring accountability of switches, and use voting to reduce the effect of packet losses. Finally, FOCES needs to solve flow-counter equation systems by computing matrix inversions, which can be costly when there are a large number of flows and rules. To make FOCES scalable, we propose to strategically slice the original large FCM into a set of smaller sub-FCMs, each corresponding to a switch. Then, we only need to solve a set of equation systems, which are much smaller than solving the original one.

In sum, our contribution is four-fold:

- We propose FOCES, a network-wide forwarding anomaly detection method in SDN, and theoretically analyze the condition of successful detection.
- We design a threshold-based detection method to eliminate false positives caused by counter noises, and make FOCES scalable with matrix slicing.
- We design a voting-based localization method to identify malicious switches that cause forwarding anomalies.
- We use experiments to show FOCES can detect and localize forwarding anomalies with high accuracy, with minimal computation and bandwidth overhead.

The rest of this paper proceeds as follows. Section II states the problem of forwarding anomaly in SDN; Section III presents the theoretic framework of FOCES; Section IV introduces the detection algorithms of FOCES, followed by security analysis; Section V introduces the localization algorithms of FOCES; and Section VI evaluates the accuracy, performance, and overhead of FOCES; Section VII discusses related work, and Section VIII concludes.

II. Problem Statement

A. System Model

This paper considers a typical SDN, where a logically-centralized *controller* manages a set of *switches*. Network operators specify high-level policies such as reachability and isolation with the API provided by the controller. The controller breaks down the policies into a set of *rules*, and populates the rules into *flow tables* of switches, through a standard *control channel* like OpenFlow [2]. Each rule consists of three parts: *matching fields*, *actions*, and *counters*. Switches forward packets by looking up in their flow tables. Specifically, when the header of a packet matches the matching fields of a rule, the switch will take the actions specified by the rule (*e.g.*, forwarding to a port), and update the corresponding counters. We assume the controller has access to the complete network topology and can request counters of rules from switches [15].

B. Threat Model

The adversary aims to change the paths that packets are forwarded, thereby causing what we call *forwarding anomaly*. Specifically, we consider the following types of forwarding anomalies, as shown in Fig. 1.

Path Deviation: Packets take a different path than the one specified by the controller. Besides general path deviations, here we highlight two special cases:

- **Switch Bypass.** Packets are received by the destination switch, but one or more switches are skipped.
- **Path Detour.** Packets deviate from one switch S_i to another switch other than the intended next-hop S_{i+1}, and come back to S_i later.

Early Drop: Packets are dropped before reaching the destination switch. Note here we implicitly assume the last-hop switch should not drop packets while pretending to have delivered them, which cannot be detected by any flow statistics verification tools.

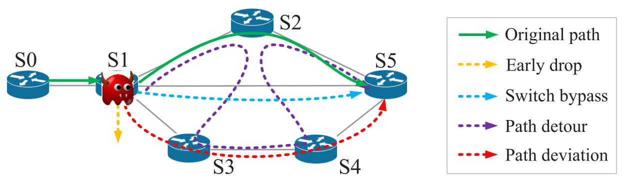


Fig. 1. Illustration of forwarding anomalies considered in this paper.

Finally, we do not consider anomalies which do not change the forwarding paths of packets, such as traffic mirroring or payload modification, since they can be defended using end-to-end encryptions. Note the above definition is inspired by previous works on forwarding anomaly detection [22], [23], [29].

We assume the adversary can compromise switches by exploiting the vulnerabilities of switch OS. Then, the adversary has the following two avenues to cause forwarding anomalies. (1) The adversary can modify output ports of forwarding rules installed at the flow tables of compromised switches. Here, we assume the adversary has full control of compromised switches. Thus, when the controller tries to dump the flow table of a compromised switch, the adversary can just report the original flow table instead of the modified one. Thus, simply dumping flow tables is not effective. (2) The adversary can directly forward any packets to any ports, without matching any rules in the flow table, thereby also dismissing the method of flow table dumping. In addition, we assume the adversary is aware of our detection method, and can modify the counters of rules at compromised switches, so as to pretend to have correctly forwarded packets.

Finally, we assume the controller is always trusted, and the majority of switches in the network are benign, *i.e.*, forwarding packets according to rules installed by the controller. Note such "majority good" assumption is also necessary for other statistics-based anomaly detection methods [26].

III. FOCES: Theoretic Framework

In this section, we will first present a brief introduction to statistics verification tools, highlighting their limitations, and then give an overview of FOCES. After that, we present the theoretic framework of FOCES, which lays the foundation for the detection and localization algorithms to be introduced in Section IV and Section V, respectively.

A. Forwarding Anomaly Detection via Statistics Verification

We first show how to detect forwarding anomalies by leveraging the "flow conservation principle". Taking Fig. 1 as an example, suppose there is only one flow $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_5$ (shown as the green solid line), which matches a rule at each switch of the path. Ideally the counters of rules at S_0, S_1, S_2, S_5 should all be equal to say a, conforming to the flow conservation principle. Now, suppose S_1 is compromised and diverts the flow to path $S_3 \rightarrow S_4 \rightarrow S_5$ (shown as the red dashed line), the counters at S_0 and S_1 will still be a, while the counter at S_2 will be smaller than a, violating the flow conservation principle.

Some statistics verification tools leverage the above idea to detect forwarding anomalies in SDN [26]–[28]. However, these tools apply the flow conservation principle for each

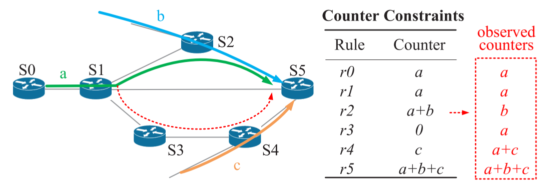


Fig. 2. An example illustrating the basic idea of FOCES. r_i is a rule installed at switch S_i $(i = 0, 1, \ldots, 5)$.

flow individually, and as a result suffer from two serious limitations.

- **Limited Detection Scope.** Since they look at one flow each time, mostly they can only check a limited number of flows [26]–[28], *e.g.*, flows passing a specific switch, flows destined to a specific IP address, etc. As a result, they may miss some forwarding anomalies happening to flows that are not checked. For example, if we only look at flows passing S_2 in Fig. 2, we may miss forwarding anomalies for flows passing S_4. *Thus, applying the anomaly detection algorithms on a per-flow or per-switch basis, without any prior knowledge of where forwarding anomalies happen, can result in a limited detection scope.*
- **High Flow Table Overhead.** In real networks, each forwarding rule may aggregate multiple flows. Thus, in order to check whether a specific flow conforms to the flow conservation principle, one cannot directly use the counters of forwarding rules [26] but has to install dedicated rules to collect the statistics of that flow [27], [28]. For example, in Fig. 2, the rule at S_2 matches two flows. To check the flow in solid blue line, one cannot use the counter of the forwarding rule at S_2 which aggregate two flows. Instead, a dedicated *counter rule* that exactly matches the flow solely for statistics collection should be installed. *If we need to analyze all flows in the network, these methods can impose large overhead for flow tables, considering that the hardware flow table size is generally small for SDN switches (a few thousands of wildcard rules [30]).*

B. FOCES Overview

Idea: The key idea of FOCES is to extend the flow conservation principle from *individual* flows to *a network of* flows, by leveraging the relationship between flows and rules. Take Fig. 2 as an example, which consists of six switches S_0 through S_5, and each switch S_i has only one rule r_i. Assume the three flows (in solid lines) have volume a, b, and c, respectively. Then, the counters of rules should satisfy the constraints listed in the middle of Fig. 2. Suppose the green flow of volume a is diverted to S_3 instead of S_2, as shown in red dashed line. Then, the right dashed box shows the counter values observed by the controller. It is easy to verify that whenever a, b, c are nonzero, the observed counter values cannot satisfy the constraints. Note here we take the counters of all flows in the network as a whole, and thereby can detect forwarding anomaly if any flow is experiencing anomalies. In addition, we directly use counters of aggregate rules, without installing dedicated counter rules.

Architecture: Fig. 3 shows the system architecture of FOCES, which consists of five major components. (1) The

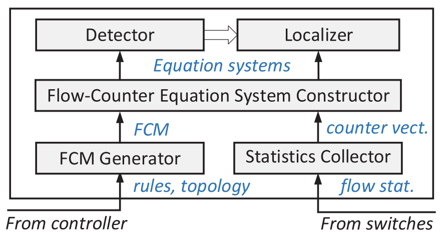


Fig. 3. The system architecture of FOCES.

TABLE I
SUMMARY OF KEY NOTATIONS

S	a switch in the network.
f	a flow in the network.
r	a rule in the network.
H	a Flow-Counter Matrix (FCM).
h_i	the ith column vector of FCM H.
X	a counter vector.
Y	a volume vector.
Δ	an error vector.
$\mathcal{G}_S^H$	the Rule Bipartite Graph (RBG) of a switch S with respect to FCM H.
AI	the anomaly index.
T	the detection threshold.
$EI(r)$	the error index of rule r.
$AW(S)$	the anomaly weight of switch S.
$FT(S)$	the set of rules in switch S's flow table.

FCM Generator retrieves the rules and topologies from the controller, and generates the *Flow-Counter Matrix (FCM)* that captures the relationship among rules and flows in the network. (2) The *Statistics Collector* periodically queries switches for flow statistics, and generates the *counter vector* storing the counters of all rules. (3) The *Flow-Counter Equation System Constructor* constructs an equation system based on the FCM and counter vector. (4) The *Detector* solves the equation system and decides whether there are forwarding anomalies by comparing the solution errors to a given threshold. (5) If the detector flags an anomaly, the *Localizer* tries to identify the switch that is accountable for the anomaly by further analyzing the equation system.

In the remaining of this section, we will present the theoretic framework of FOCES, and analyze the detection boundary of FOCES, *i.e.*, under what condition FOCES can successfully detect forwarding anomalies.

C. Flow-Counter Equation System

We show how to capture the constraints for counters of all rules in the network as a linear equation system. For now, we assume an ideal setting where there are no packet losses, and counters of all rules are perfectly synchronized. Later in the next section, we will show how to extend it to work in realistic settings.

First, assume there are n flows $f_1, f_2, \ldots, f_n$, and m rules $r_1, r_2, \ldots, r_m$ in the network, where $m > n$. Define the *Flow-Counter Matrix (FCM)* $H_{m\times n}$ as:

$$H_{i,j} = \begin{cases} 1 & \textit{if flow } f_j \textit{ matches rule } r_i \\ 0 & \textit{otherwise} \end{cases} \tag{1}$$

Then, define the *counter vector* as $Y = (y_1, y_2, \ldots, y_m)^T$, where y_i is the counter value of rule r_i, and the *volume vector* as $X = (x_1, x_2, \ldots, x_n)^T$, where x_i is the volume of flow f_i. When there is no forwarding anomaly, X and Y should satisfy what we term as the *flow-counter equation system*:

$$HX = Y \tag{2}$$

Suppose due to attacks the FCM is changed to $H' \neq H$. Then, the observed counter vector will become $Y' = H'X \neq Y$, Since the controller has no idea what H' is, when it tries to recover X, it needs to solve the flowing flow-counter equation system:

$$HX' = Y' \tag{3}$$

Here X' is the unknown variable. Since $m > n$, Eq. (3) is an overdetermined equation system. Also, since $Y' = H'X$, with $H' \neq H$, it is very possible that the overdetermined equation system is inconsistent, meaning that there is no exact solution. Then, the least-square estimate for X will be:

$$\hat{X} = (H^T H)^{-1} H^T Y' \tag{4}$$

With this estimate, the resultant counter vector will be $\hat{Y} = H\hat{X}$. Define the *error vector* Δ as the absolute difference between $\hat{Y}$ and Y':

$$\Delta = |Y' - \hat{Y}| \tag{5}$$

Then, if we know that $\Delta \neq 0$, we can definitely conclude that there is a forwarding anomaly in the network.

To make the idea more concrete, consider the example shown in Fig. 2, where the original and actual FCM are:

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad H' = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{6}$$

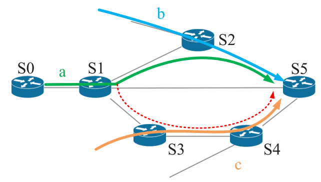
Let the volume vector of these three flows be $X = (a, b, c)^T = (3, 4, 5)^T$, then it is easy to calculate Y', $\hat{X}$, $\hat{Y}$, and Δ as:

$$Y' = \begin{bmatrix} 3 \\ 3 \\ 4 \\ 3 \\ 8 \\ 12 \end{bmatrix}, \quad \hat{X} = \begin{bmatrix} 3 \\ 1 \\ 8 \end{bmatrix}, \quad \hat{Y} = \begin{bmatrix} 3 \\ 3 \\ 4 \\ 0 \\ 8 \\ 12 \end{bmatrix}, \quad \Delta = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

Since $\Delta \neq 0$, we flag the network under forwarding anomaly.

FCM Generation: To realize FOCES, we need a way to efficiently generate the FCM. Since there are a large number of "physical flows", defined by say TCP five-tuple, we cannot directly use them due to scalability issues. Here, we choose to use the notion of "logical flows" or packet equivalence classes. A logical flow is defined as a class of packets that experience the same set of rules in the network.

Specifically, we generate the FCM by adapting the algorithm for generating all-reachability table proposed in ATPG [31]. First, we create a symbolic header with all bits set to wildcards, and inject it to each terminal port of the network. For each terminal port, we match the symbolic header against each rule in the flow table of the switch possessing that port, and create a new symbolic header. The new symbolic header is constrained



Counter Constraints

Rule	Counter	observed counters
r0	*a*	*a*
r1	*a*	*a*
r2	*a+b*	*b*
r3	*0*	*a+c*
r4	*c*	*a+c*
r5	*a+b+c*	*a+b+c*

Fig. 4. A counterexample where FOCES misses the forwarding anomaly.

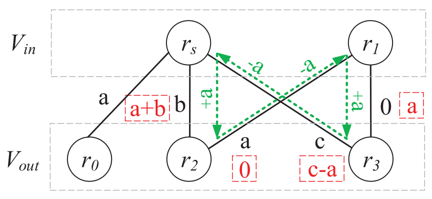


Fig. 5. The flow rule graph for S_2 in Fig. 4.

by the matching fields of the rule, and the actions of the rule are taken on the header. For example, if a rule has a matching field $dst\_ip = 10.0.0.1/24$ and an action "Output to Port 2", then the dst_ip of the header will be set to $10.0.0.1/24$, and the new header will be forwarded to the switch connected to Port 2. As each header h traverses the network, the set of rules matched by the header are recorded in $h.history$. When it reaches a terminal port, a flow is created, and a column is added to the FCM. The column has 1 at each row whose corresponding rule appears in $h.history$, and 0 at all other rows.

D. The Detection Boundary of FOCES

FOCES has a detection boundary, within which anomalies can be detected. In what follows, we will first give a counterexample, where a forwarding anomaly does not cause $\Delta \neq 0$. Then, we theoretically analyze the condition under which FOCES can detect forwarding anomalies. With this condition, we can decide whether a forwarding anomaly in a network can be detected.

Counterexample for FOCES: Fig. 4 shows an example, which is much the same with the one in Fig. 2, except that the flow of volume c now passes switch S_3 before reaching S_4 and S_5. Still, we let the volume vector for these flows be $X = (a, b, c)^T = (3, 4, 5)^T$. Then, the original FCM H, the actual FCM H', and the observed counter vector Y' are:

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad H' = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad Y' = \begin{bmatrix} 3 \\ 3 \\ 4 \\ 8 \\ 8 \\ 12 \end{bmatrix} \tag{8}$$

According to FOCES, the estimate of volume vector is $\hat{X} = (3, 1, 8)^T$, which is an exact solution to $HX' = Y'$, *i.e.*, a solution with $\Delta = 0$. The reason is that $HX' = Y'$ is a consistent equation system. Thus, the condition for successful detection is equivalent to the condition for the flow-counter equation system to be inconsistent. In the following, we will analyze such condition from linear algebraic point of view.

Analysis on Detection Boundary: We still consider a specific network consisting of m rules $r_1, r_2, \ldots, r_m$, and n flows $f_1, f_2, \ldots, f_n$. Represent the FCM $H_{m \times n}$ as a row vector $(h_1, h_2, \ldots, h_n)$. Since each column h_i corresponds to flow f_i, in the following we will also use h_i to refer to flow f_i. In the following, we define what a forwarding anomaly is and under what condition it can be detected.

Definition 1: Consider a network and let H be its FCM. If a flow $h_i \in H$ with nonzero volume is modified to $h'_i \neq h_i$, we say h_i is experiencing a *forwarding anomaly*, denoted as $FA(h_i, h'_i)$.

Note that the above definition captures the two avenues of causing forwarding anomalies (*i.e.*, modifying output ports of rules, and directly forwarding packets without matching rules) that we have specified in Section II-B. The reason is that both avenues will make the forwarding path of a flow h change to a different one, and thus the rules matched by the modified flow h' will be different from those of h.

Definition 2: We say a forwarding anomaly $FA(h_i, h'_i)$ is *detectable* if and only if its flow-counter equation system of Eq. (3) is inconsistent, Otherwise, we say $FA(h_i, h'_i)$ is *undetectable*.

Theorem 1: A forwarding anomaly $FA(h_i, h'_i)$ is *undetectable* if and only if h'_i lies in the linear subspace generated by $h_1, h_2, \ldots, h_n$.

Proof: See Appendix A. ■

Even Theorem 1 gives the necessary and sufficient condition to determine whether a forwarding anomaly is detectable, it is difficult to apply it to real networks. In the following, we will reduce the above condition to the problem of finding a loop in a bipartite graph, which is much easier and intuitive to check. We will first introduce the notion of Rule Bipartite Graph (RBG).

Definition 3: The *Rule Bipartite Graph (RBG)* of a switch S with respect to FCM H, denoted as $\mathcal{G}_S^H(V_{in}, V_{out}, E)$, is constructed as follows. V_{out} consists of all the rules currently installed at switch S. If there is a flow $h \in H$ matching a rule r_i and a rule $r_j \in V_{out}$ in sequence, denoted by $r_i \xrightarrow{h} r_j$, then we add r_i into V_{in} and (r_i, r_j) into E. Similarly, if there is a flow $h \in H$ matching a rule $r_i \in V_{in}$ and a rule r_j in sequence, then we add r_j into V_{out} and (r_i, r_j) into E. To simplify notations, we also add a virtual rule r_s acting as the first rule of all flows. Formally, we have:

$$V_{in} \triangleq \{r_i | \; \exists r_j \in V_{out}, \; \exists h \in H, \; r_i \xrightarrow{h} r_j\} \cup \{r_s\} \tag{9}$$

Theorem 2: A forwarding anomaly $FA(h_i, h'_i)$ is *undetectable* if and only if there is a switch S whose RBG with respect to FCM $\tilde{H} \triangleq H \cup \{h'\}$,, *i.e.*, $\mathcal{G}_S^{\tilde{H}}(V_{in}, V_{out}, E)$, contains a loop.

Proof: See Appendix B. ■

To explain what Theorem 2 means, we return to the example in Fig. 4. The RBG of S_2 is shown in Fig. 5. We can see that there is a loop marked by green dashed lines. The numbers besides the edges are the volumes of flows. By applying the operations $(\pm x)$ marked besides the dashed lines, we can obtain a new flow distribution shown inside the red dashed rectangles. Under this new distribution, the counter values of all rules can be achieved. That is, the distribution can be seen as another "explanation" for the observed counter values. Under this new explanation, we can redirect the flow of volume

a from $r_1 \rightarrow r_2$ to $r_1 \rightarrow r_3$, without breaking the constraints specified by the flow-counter equation system.

Note the detection boundary given by Theorem 2 does not limit the usage of FOCES in real networks. The reason is that the probability that a forwarding anomaly will introduce a loop in some RBG can be very small. Our experiments with realistic datacenter network topologies show that most of the anomalies can be successfully detected (see Fig. 14). As one of our future work, we will study how to strategically generate rules such that it is impossible for any forwarding anomalies to cause loops in RBGs.

IV. Anomaly Detection

The last section shows how FOCES can detect forwarding anomalies in an ideal setting. However, to detect anomalies in realistic settings, we are still faced with two challenges: (1) **Noises.** In real networks, both packet losses and out-of-sync counters can result in $\Delta \neq \mathbf{0}$. We need to make FOCES robust against such noises. (2) **Overhead.** Computing Δ requires calculating the inverse of FCM, which is expensive when there are a large number of rules and flows. We need to speedup FOCES to make it scalable for large networks. In the following, we show how we resolve the above two challenges.

A. Making Detection Robust Against Noises

In the following, we first define the *anomaly index* to measure the possibility that a forwarding anomaly exists. If the anomaly index is higher than a threshold T, then we say there are forwarding anomalies. Then, we discuss how to set the default threshold analytically.

Anomaly Index: The design of anomaly index is based on the "majority good" assumption, *i.e.*, most of the flows are forwarded correctly except a small fraction. Specifically, let Δ be the error vector calculated using Eq. (5). Then, the anomaly index AI is defined as $\frac{Err_{max}}{Err_{med}}$, where Err_{max} and Err_{med} are the maximum and median of all elements in Δ, respectively. Due to the "majority good" assumption, Err_{med} should be always small, while Err_{max} can be large when there are forwarding anomalies. For the example shown in Fig. 2, $Err_{max} = 3$, $Err_{med} = 0$, and $AI = +\infty$. Based on this, FOCES judges that there are forwarding anomalies if $AI > T$, where T is termed as the *detection threshold*. In the following, we show how to set the default detection threshold.

Default Detection Threshold: Choosing a good default value for detection threshold analytically requires us to know the probability distribution of Δ. However, this is difficult since we do not have models for the noises introduced by packet losses and out-of-sync counters. As a compromise, we use an over-simplified way to estimate the distribution of Δ, and use this distribution to choose a default threshold for FOCES. Specifically, we approximate both $Y'(i)$ and $\hat{Y}(i)$ with normal distribution $N(Y(i), \sigma^2)$, where $Y(i)$ is the expected counter values in ideal settings. Under this assumption, each element of Δ follows a *folded normal distribution*, whose cumulative distribution function is $F(x) = erf(x/\sqrt{2\sigma^2})$, where $erf()$ is the error function [32]. By solving $F(x) = 1/2$, we have $x = \sqrt{2}erf^{-1}(1/2)\sigma \approx 0.675\sigma$. Thus, we use 0.675σ to approximate Err_{med}. According to the three-sigma rule in

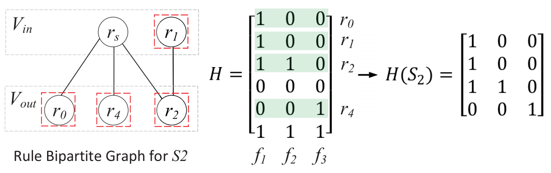

V_{in}
r_s
r_1
V_{out}
r_0
r_4
r_2
Rule Bipartite Graph for $S2$
$H = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
r_0 r_1 r_2 r_4
f_1 f_2 f_3
$H(S_2) = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Fig. 6. The construction of sub-FCM for S_2 in Fig 2.

statistics [33], Err_{max} should be less than 3σ with probability 0.997. Thus, $\frac{Err_{max}}{Err_{med}}$ should be less than $\frac{3\sigma}{0.675\sigma} \approx 4.4$ with high probability. Thus, we choose $T = 4.5$ as the default detection threshold. We will show this threshold achieves a good detection accuracy with experiments (see Section VI-D).

Note that the above analysis only offers a simple way to choose the default detection threshold. We will study how different choices of threshold value affect the false positive rate and true positive rate in Section VI-D. Note it is difficult to determine the optimal threshold value since it depends on a lot of factors including traffic characteristics, packet loss rates, and customized tradeoff between false positives and false negatives. We acknowledge that this challenge is faced by FOCES, as well as other statistics-based methods [26], [28].

Algorithm 1 summarizes the threshold-based detection algorithm.

```
Algorithm 1 DetectAnomaly($Y'$, $H_{m\times n}$, $T$)
Input: $Y' = (y_1, y_2, \ldots, y_m)^T$: the counter vector,
       $H_{m\times n}$: the Flow-Counter Matrix (FCM), $T$: the
       detection threshold.
Output: True (Anomaly) or False (Normal).
1  Compute the volume vector estimate
   $\hat{X} \leftarrow (H^T H)^{-1} H^T Y'$;
2  Compute the counter vector estimate $\hat{Y} \leftarrow H\hat{X}$;
3  Compute the error vector $\Delta \leftarrow |Y' - \hat{Y}|$;
4  $Err_{max} \leftarrow$ maximum of $\Delta$;
5  $Err_{med} \leftarrow$ median of $\Delta$;
6  $AI \leftarrow \frac{Err_{max}}{Err_{med}}$;
7  if $AI > T$ then
8      return True;
9  else
10     return False;
11 end
```

B. Making Detection Scalable With FCM Slicing

In the following, we show how to make FOCES scalable by reducing the computation time. Our method is inspired by the Rule Bipartite Graph (RBG) introduced in Section III. We observe that the RBG for a specific switch only contains rules of that switch and their predecessor rules. We can extract the sub-FCM corresponding to the RBG, one for each switch. Since sub-FCMs are much smaller than the original FCM, we can expect to reduce the computation time by applying the threshold-based algorithm for each sub-FCM individually.

FCM Slicing: For a specific switch S_i, let its RBG be $\mathcal{G}_{S_i}(V_{in}, V_{out}, E)$. First, we extract the rules $R(S_i)$ for S_i as $\{(V_{in} \cup V_{out}) \backslash r_s\}$. Note the virtual flow rule r_s is not included.

Then, we identify the flows $F(S_i)$ for S_i as those that match at least one flow rule in $R(S_i)$. $H(S_i)$ is the sub-matrix of H with only those rows and columns corresponding to $R(S_i)$ and $F(S_i)$, respectively.

Here, we present an example to show how to slice the original FCM into sub-FCMs. Fig. 6 shows the RBG $\mathcal{G}_{S_2}(V_{in}, V_{out}, E)$ for switch S_2 in Fig 2. The rules in $R(S_2)$ are marked with dashed rectangles, and the flows in $F(S_2)$ are just all flows in the network. The sub-FCM $H(S_2)$ is a 4×3 sub-matrix of the original FCM H. Since this small topology contains only 6 rules and 3 flows, the effect of slicing is not that remarkable. In real networks with many rules and flows, the sub-FCMs can be much smaller compared with the original FCM.

Algorithm 2 summarizes the threshold-based detection algorithm with slicing.

```
Algorithm 2 DetectAnomalySlicing($Y', H_{m\times n}, T, n$)
Input: $Y' = (y_1, y_2, \ldots, y_m)^T$: the counter vector,
       $H_{m\times n}$: the Flow-Counter Matrix (FCM), $T$: the
       detection threshold, $n$: the number of switches.
Output: True (Anomaly) or False (Normal).
1  foreach $i \leftarrow 1$ to $n$ do
2      Compute the sub-FCM $H(i)$ for switch $S_i$;
3      Extract the sub-vector $Y'(i)$ for switch $S_i$ from $Y'$;
4      Compute the volume vector estimate
       $\hat{X} \leftarrow (H(i)^T H(i))^{-1} H^T Y'(i)$;
5      Compute the counter vector estimate $\hat{Y}(i) \leftarrow H(i)\hat{X}$;
6      Compute the error vector $\Delta(i) \leftarrow |Y'(i) - \hat{Y}(i)|$;
7      $Err_{max} \leftarrow$ maximum of $\Delta(i)$;
8      $Err_{med} \leftarrow$ median of $\Delta(i)$;
9      $AI \leftarrow \frac{Err_{max}}{Err_{med}}$;
10     if $AI > T$ then
11         return True;
12     end
13 end
14 return False;
```

Analysis on Detection Equivalence: The following theorem says that the detection algorithm with slicing is equivalent to the one without slicing in detecting forwarding anomalies.

Theorem 3: If a forwarding anomaly $FA(h_i, h'_i)$ is detectable (without slicing), then it is still detectable when using slicing.

Proof: See Appendix C. ■

We will use experiments to further validate such equivalence in Section VI-F.

Analysis on Computation Complexity Reduction: We use analysis to show that by using slicing, the computation complexity of FOCES reduces from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^{2.3})$, where N is the number of flows in the network (See Appendix D).

We will use experiments to further demonstrate the reduction of computation time by using slicing in Section VI-F.

C. Security Analysis

In the following, we show how FOCES can detect packet early drops and path deviation that are defined in Section II. Suppose a packet p should be forwarded along a path $S_1 \rightarrow S_2, \ldots, \rightarrow S_n$, and let r_i be the rule matched by p at switch S_i. Consider the following anomalies.

Path Deviation: Here, we only consider the following two special cases, and the cases of general path deviation are largely the same.

- **Switch Bypass.** Suppose S_i is compromised and forwards p directly to switch S_{i+2}, bypassing the intended next hop S_{i+1}. Although the counters of r_i and r_{i+2} can be made consistent, the counters of r_{i+1} will be less than expected, resulting in inconsistency. Such inconsistency can be detected by FOCES when the condition given in Theorem 2 is met.
- **Path Detour.** Suppose S_i is compromised and forwards p along a path $S_i \rightarrow D_1 \rightarrow D_2, \ldots, \rightarrow D_m \rightarrow S_i \rightarrow S_{i+1}$. Although the counters of r_i and r_{i+1} can be made consistent (recall that the adversary has full control over S_i), the counters of D_1 through D_m will be higher than its expected value, resulting in inconsistency. Such inconsistency can be detected by FOCES when the condition given in Theorem 2 is met.

Early Drop: Suppose S_i is compromised and drops p instead of sending it to its next hop S_{i+1}. As a result, the counter of r_{i+1} should be less than its expected value, thereby breaking the consistency between the counters of r_i and r_{i+1}. Such inconsistency can be detected by FOCES when the condition given in Theorem 2 is met.

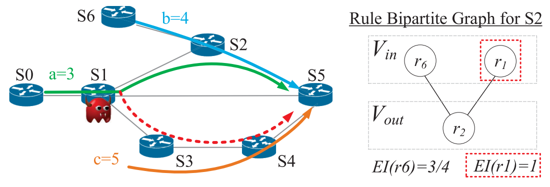
V. Anomaly Localization

In the last section, we have shown how FOCES can detect forwarding anomalies by analyzing counters of flow rules. It will be even better if we know which switches are accountable for the anomalies, which can help us isolate these switches. In this section, we will present two algorithms for localizing malicious switches. For simplicity of presentation, we first assume there is only one malicious switch, and then discuss how to handle multiple ones.

A. Observation and Basic Idea

Recall in the slicing-based approach, we calculate the anomaly index AI for each switch. One may think the switch with the largest anomaly index AI is the malicious switch. However, we observe that *when a malicious switch modifies one of the rules in its flow table, the counters of those rules at its downstream switches will actually be affected, resulting in a larger anomaly indices AI for these switches.* Taking Fig. 7 as an example, there are seven switches S_0 through S_6, and each switch S_i has only one rule r_i. Suppose S_1 is the malicious switch which modifies the action of r_1 from "forwarding to S_2" to "forwarding to S_3". As a result, r_2's counter becomes 4 instead of 7, and the error vector for S_2, denoted as $\Delta(S_2)$, is calculated as $(1, 1, 1)^T$, Thus, the AI for S_2 is 1, while the AI's for other switches (including S_1) are 0.

Based on the above observation, our idea is to first find the switch, say S_i, that has the largest anomaly index, and then suspect one of S_i's *upstream* switches is malicious. In the following, we will introduce a baseline localization algorithm, discuss its limitation, and finally present an improved algorithm.



numerical results for S2

$$X = \begin{bmatrix}3\\4\end{bmatrix}\ H = \begin{bmatrix}1 & 0\\1 & 1\\0 & 1\end{bmatrix}\ H' = \begin{bmatrix}1 & 0\\0 & 1\\0 & 1\end{bmatrix}\ Y' = \begin{bmatrix}3\\4\\4\end{bmatrix}\ \hat{X} = \begin{bmatrix}2\\3\end{bmatrix}\ \hat{Y} = \begin{bmatrix}2\\5\\3\end{bmatrix}\ \Delta = \begin{bmatrix}1\\-1\\1\end{bmatrix}$$

Fig. 7. An example of localizing malicious switches.

B. The Baseline Localization Method

Before introducing the baseline localization method, we first formally define what a upstream switch is. Let $\mathcal{G}_{S_i}(V_{in}(S_i), V_{out}(S_i), E(S_i))$ be the Rule Bipartite Graph (RBG) for switch S_i, then we say S_j is an upstream switch of S_i if S_j has a rule $r \in V_{in}(S_i)$. The set of upstream switches of S_i is then:

$$UP(S_i) \triangleq \{S_j |\ \exists r \in (FT(S_j) \cap V_{in}(S_i))\} \tag{10}$$

where $FT(S_j)$ is the set of rules in S_j's flow table.

Since $UP(S_i)$ can have more than one switch, we still need to identify the malicious switch in $UP(S_i)$. To achieve that, we evaluate the possibility for each rule $r \in V_{in}(S_i)$ to be the modified rule, and then choose the switch that has the rule with the largest possibility as the malicious one. Specifically, let Δ be the error vector for S_i, and $Sum(\Delta)$ be the sum of absolute values for all elements in Δ. For each rule $r \in V_{in}(S_i)$, we calculate its *error index* EI as:

$$EI(r) = \frac{Min(Sum(\Delta), r.counter)}{Max(Sum(\Delta), r.counter)} \tag{11}$$

Here, $r.counter$ is the counter value of r. Note $EI(r)$ reflects the similarity between $Sum(\Delta)$ and $r.counter$. The rationale for this definition of EI is explained as follows.

Suppose S_i has n upstream switches, *i.e.*, $UP(S_i) = \{S_1, S_2, \ldots, S_n\}$, and each upstream switch S_j has a rule r_j which forwards flows to S_i. Assume these flows from upstream switches match m rules at S_i. Then, the sub-FCM of S_i, denoted by $H(S_i)$, has $n+m$ rules, *i.e.*, n rules at upstream switches and m rules at S_i. Let $H'S_i$ be the sub-FCM due to forwarding anomalies, and let the expected and observed counter value of rule r_i be c_i and c'. Suppose $S_x \in UP(S_i)$ is malicious and modifies its rule r_x to send flow f_i to another switch other than S_i. Then, we have $c'_i = c_i - c_x \neq c_i$, with a deviation of c_x. We observe that the least square estimate will result in the deviation c_x will be evenly spread into each rule' counter. Specifically, suppose the least square estimate for X is $\hat{X}$, then, the resultant counter vector $\hat{Y} = H'(S_i)\hat{X}$ will be:

$$\hat{Y} = (c'_1 \pm \frac{c_x}{n+m}, c'_2 \pm \frac{c_x}{n+m}, \ldots, c'_{m+n} \pm \frac{c_x}{n+m})^T \tag{12}$$

Recall in Eq. (5), $Sum(\Delta)$ is the sum of the absolute deviation of each rule's counter, *i.e.*, $Sum(\Delta) = \sum_{i=1}^{m+n} |c'_i - (c'_i \pm \frac{c_x}{n+m})| = c_x$.

Take Fig. 7 for example, where we find S_2 has the maximum AI. For switch S_2, we have $n = 2$, $m = 1$, and $Sum(\Delta) = 3$. $\hat{Y}$ is calculated as:

$$\hat{Y} = (3 - \frac{c_1}{2+1}, 4 + \frac{c_1}{2+1}, 4 - \frac{c_1}{2+1})^T \tag{13}$$

Here, c'_1 is the counter of r_1, *i.e.*, $c'_1 = 3$. Since, $V_{in}(S_2) = \{r_1, r_6\}$, we compute EI for r_1 and r_6 and get $EI(r_1) = 3/3 = 1$ and $EI(r_6) = 3/4$.

Given the above discussion, we choose to use the similarity between a rule's counter and $Sum(\Delta)$ as the error index EI of the rule. That is, the larger a rule's EI is, the more possible the rule is modified. Our baseline localization method tries to find the rule r with the largest EI, and if r is installed at S, then it identifies S as the malicious switch. In the above example, r_1 has the largest EI, and we can correctly identify S_1 as the malicious switch.

Even the baseline localization can help localize the malicious switch, it has several limitations. (1) Noises (*e.g.*, packet losses, out-of-sync counters) can make AI fluctuate dramatically. As a result, multiple switches may have similar AI's, and we cannot determine which switch is at the downstream of the malicious switch. If we make a hard decision here, we may run into false localization results. (2) Even we can correctly determine the downstream switch, rules in the V_{in} of its RBG may have similar EI's, which can further result in false localization results. Consider the example in Fig. 7, and suppose the volume of flow in blue becomes 3. Then, the counter of r_6 is 3, and $EI(r_6)$ would be 1, the same as $EI(r_1)$. In such a case, we cannot identify S_1 correctly.

C. The Voting-Based Localization Method

To overcome the the above limitations, we propose an improved localization method. In this method, we introduce an *anomaly weight* AW for each switch, indicating the possibility for a switch to be malicious. The anomaly weight $AW(S)$ of S is calculated as follows:

$$AW(S) = \frac{\sum_{r \in FT(S)} EI(r)}{|FT(S)|} \tag{14}$$

Then, we identify the switch with the maximum AW as malicious.

Algorithm 3 summaries the localization process of voting-base method. First, FOCES generates its RBG for each switch S_i, and constructs its sub-FCM H_i from the RBG (Line 1-3). Then, FOCES computes the error vector δ_i for S_i (Line 4), and computes the error index EI for each rule in V_{in} of the RBG (Line 5-7). Finally, FOCES calculates the anomaly weight AW for each switch by first summing up the EIs of all rules installed in it, and normalizes the sum with the number of rules (Line 9-15). The switch with the maximum AW is identified as malicious (Line 16-17).

First, note this voting-base method improves FOCES's robustness against packet losses. Since packet losses will not just affect one rule's EI, but all other rules' EI's. Thus, if we compute the sum of all EI's for each switch and compare the sums, the effect of packet losses will even out. Secondly, a malicious switch may probably modify more than one rule. If so, by summing up the EI's for each switch, the difference between the malicious switch and normal switches will be even remarkable, thereby improving the localization accuracy. We

Algorithm 3 LocalizeMaliciousSwitch($\mathcal{S}$)

```
Input: $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$: the set of all switches
Output: $S_x \in \mathcal{S}$: the malicious switch
1  foreach $i \leftarrow 1$ to $n$ do
2      Compute the RBG of $\mathcal{G}_{S_i}(V_{in}, V_{out}, E)$ for $S_i$ ;
3      Construct the sub-FCM $H_i$ from the RBG of $S_i$;
4      Compute the error vector $\Delta_i$ for $S_i$ ;
5      foreach $r \in V_{in}$ do
6          $EI(r) \leftarrow \frac{Min(Sum(\Delta_i), r.counter)}{Max(Sum(\Delta_i), r.counter)}$ ;
7      end
8  end
9  foreach $i \leftarrow 1$ to $n$ do
10     $AW(i) \leftarrow 0$ ;
11     foreach $r \in FT(S_i)$ do
12         $AW(i) \leftarrow AW(i) + EI(r)$ ;
13     end
14     $AW(i) \leftarrow AW(i)/|FT(S_i)|$ ;
15 end
16 $x \leftarrow \text{argmax}_i SW(i)$;
17 return $S_x$ ;
```

will use experiments to show the voting-based method indeed improves the accuracy compared with the baseline method (see Section VI-E).

Discussion: When there are multiple malicious switches, we can simply adapt this voting-based method by introducing a threshold, *i.e.*, *localization threshold*. If a switch has a value of AW larger than the localization threshold, we identify this switch as one of the malicious switches. How to choose the value of localization threshold is similar to that for detection threshold, and is out of scope of this paper.

VI. Implementation and Evaluation

This section presents the implementation of FOCES, and evaluates it with experiments. We are interested in answering the following questions:

1) Can FOCES *effectively* detect forwarding anomalies when there are packet losses?
2) Will FOCES achieve a high detection *accuracy* and *precision*?
3) How *accurately* can FOCES localize the malicious switches accountable for the detected anomalies?
4) Whether slicing can help FOCES achieve a *faster* detection without loss of accuracy?
5) Will FOCES incur large *overhead* on the channel from switches to the controller?

A. Implementation

We prototype FOCES with approximately 1800 lines of Python codes. The architecture is shown in Fig. 3, and some implementation details are explained as follows. The *FCM Generator* retrieves the rules and topologies via the `Static Flow Pusher`, a REST API offered by Floodlight, and generates the FCM using an algorithm adapted from ATPG [31], as introduced in Section III-C We store the FCM as a sparse matrix using the Python `sparse` library. The *Statistics Collector* collects flow statistics also via the `Static Flow Pusher` API, and the *Detector* uses the `NumPy` library in Python to compute matrix inversion and transpose.

TABLE II

The Parameters of Four Network Topologies Used in Our Experiments

	# switches	# hosts	# flows	# rules
Stanford	26	26	650	1300
FatTree(4)	20	16	240	556
BCube(1,4)	24	16	240	597
DCell(1,4)	25	20	380	859

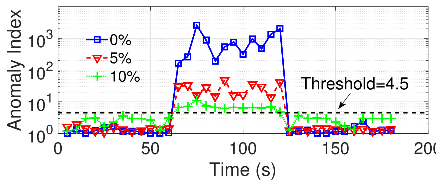


Fig. 8. The anomaly indices with and without forwarding anomalies, respectively. BCube(1,4) is used for the experiment.

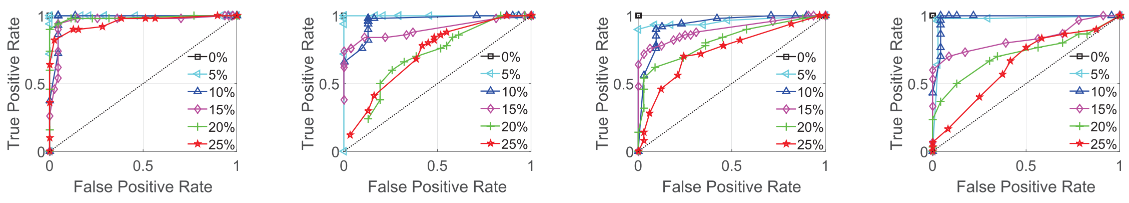
B. Experiment Setup

We use Floodlight v2.1 [34] as the controller, and use Mininet [35] to generate different network topologies, consisting of Open vSwitches [36]. Floodlight and Mininet are run on the same Linux desktop with 3.5GHz Intel Core i3 CPU and 16GB memory. We use four different network topologies including: the Stanford backbone network (Stanford) [37], FatTree(4), BCube(1,4), and DCell(1,4). For Stanford, we attach one host to each switch, while for the other three topologies, we attach one host for each edge switch. The parameters of these topologies are summarized in Table II. For each network, we generate a flow of the same rate between each pair of host by using `iperf`. Since we fix the total flow rate of each network to $800Mbps$, the flow rate is $2Mbps$ for Stanford, $4Mbps$ for FatTree and BCube, and $2.5Mbps$ for DCell. For each flow, the Floodlight controller computes flow rules using shortest-path routing, and installs the rules at switches' flow tables.

C. Experiment 1: Functional Test

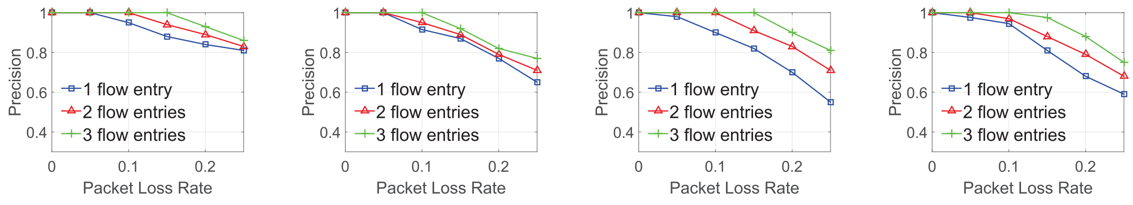
In this experiment, we test whether FOCES can detect forwarding anomalies when there are packet losses. We use BCube(1,4) under packet loss rates 0%, 5%, and 10%. After 60 seconds, we randomly modify one rule in the network to create forwarding anomalies, and repair the modified rule after another 60 seconds. The experiment runs for 180 seconds in total. FOCES runs a detection process every 5 seconds, with the detection threshold set to 4.5.

Fig. 8 shows the anomaly index of each detection. We can see the index quickly goes beyond the threshold, when the forwarding anomalies happen, and returns to low values when the forwarding anomalies end. We also see that when the packet loss rate increases, the anomaly indices for forwarding anomalies and normal cases become less distinguishable.



(a) Stanford (b) FatTree(4) (c) BCube(1,4) (d) DCell(1,4)

Fig. 9. The ROC curves for four different topologies.



(a) Stanford (b) FatTree(4) (c) BCube(1,4) (d) DCell(1,4)

Fig. 10. The detection precision for four different topologies.

D. Experiment 2: Detection Accuracy and Precision

Detection Accuracy: We first evaluate the detection accuracy of FOCES under different packet loss rates. We use the receiver operating characteristic (ROC) curve, which plots the True Positive (TP) rate against the False Positive (FP) rate, and the area under the ROC curve represents the detection accuracy. In the experiment, for each network topology, we randomly modify one flow rule, and vary the detection threshold from 1 to 100. The detection threshold is still set to 4.5.

Fig. 9 reports ROC curves for different packet loss rates from 0% to 25%. The dotted lines from the left-bottom to right-top are reference curves representing "random guess", and the larger area under the ROC curve, the more accurate the detection method is. We can see that the accuracy of FOCES is little affected when the packet loss rate is below 10%, for all four networks. Especially for DCell with packet loss rate 10%, FOCES achieves a TP rate nearly 100% and a FP rate around 4.3%. The accuracy of FOCES drops when packet loss rate is larger than 10%. The reason is that packet losses can violate the constraints of counters, leading to more false positives. Despite the degradation, FOCES is still a useful indicator of forwarding anomaly (compared with "random guess"), even for packet loss rate as high as 25%. The above results show that FOCES has a high detection accuracy under moderate packet losses.

Detection Precision: We continue to evaluate the detection precision of FOCES, and how it is affected by the number of forwarding anomalies. To quantify precision, we use $\frac{TP}{TP+FP}$, which indicates what percentage of samples that are marked as forwarding anomalies are actually such. In this experiment, we fix the detection threshold to $T = 3.5$, and vary the packet loss rates. For each packet loss rate, we randomly modify 1, 2, and 3 rules.

Fig. 10 reports the relationship between detection precision and packet loss rate, for different number of modified flow rules. Here each data point is an average of 50 experiment runs. We can see that for a fixed packet loss rate, the precision improves as more rules are modified (thereby causing more forwarding anomalies). Thus, FOCES can identify anomalies more precisely, when more flows are deviating from their expected paths.

E. Experiment 3: Localization Accuracy

In this experiment, we evaluate the accuracy of the proposed localization methods, including the baseline method and the voting-based method. To simulate forwarding anomalies, in each experiment run, we randomly pick one switch as the malicious switch, and randomly modify one rule in its flow table. Then we try to detect the forwarding anomaly using FOCES, and if an anomaly is detected, we check whether FOCES can localize the malicious switch.

We run the experiment using the four topologies and for each topology, we repeat the experiment for 50 times. Let N_d and N_l be the number of times that FOCES successfully detect the forwarding anomalies and localize the malicious switch, respectively. Note here $N_l \leq N_d$ since we perform localization only after FOCES successfully detects the anomaly. Then, we calculate the localization accuracy as N_l/N_d.

In addition to reporting the above "exact" accuracy, where only the rule/switch with the largest EI/AW is chosen to be malicious, we also report a "coarse" accuracy, where two rules/switches with the top-2 largest EI/AW are chosen. Specifically, we report N_l'/N_d, where N_l' is the number of times for successful localization, given that the malicious switch has a rule with the top-2 highest EI (for the baseline method), or has the top-2 highest AW (for the voting-based method).

As can be seen in Fig. 11, the improved voting-based method can achieve a higher localization accuracy than the baseline method. Specifically, when the packet loss rate is 5%, the baseline method can only achieve an accuracy of around 50%, while the voting-based method can achieve an accuracy of around 80%. If we are allowed to choose the top-2 suspected rules/switches, the accuracy for both methods increases, and the voting-base method can achieve an accuracy as high as 90% for all four topologies when the packet loss rate is no larger than 10%.

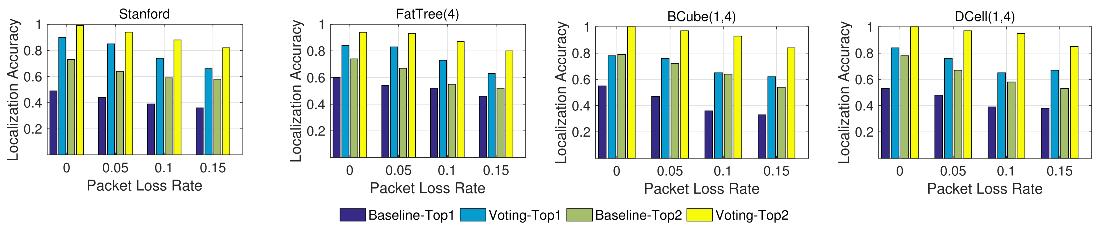


Fig. 11. The localization accuracy of baseline and voting-based method for different topologies. "Top1" refers to only selecting the rule/switch with the largest EI/AW, and "Top2" refers to selecting two rules/switches with the top-2 largest EI/AW.

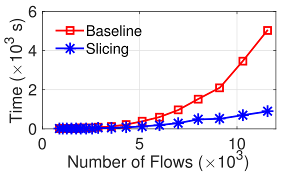


Fig. 12. The detection time with and without slicing.

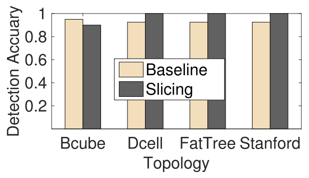


Fig. 13. The detection accuracy with and without slicing.

F. Experiment 4: The Effectiveness of Slicing

In this experiment, we study whether slicing can help FOCES reduce the computation time while maintaining high detection accuracy.

First, we show slicing can indeed help FOCES significantly reduce the computation time. We use the FatTree(8) topology, and set up different number of flows. As can be seen from Fig. 12, the computation time of the baseline FOCES (without slicing) is around 40 seconds when there are no more than 5K flows, but it increases rapidly to more than 5K seconds when there are around 12K flows. In contrast, the computation time of FOCES with slicing grows much slower than the baseline. Specifically, when there are around 12K flows, the computation time of FOCES with slicing is less than 20% that of FOCES without slicing.

Secondly, we show slicing will not decrease the detection accuracy of FOCES. Fig. 13 reports the detection accuracy of FOCES with and without slicing, when the optimal threshold value is set. The accuracy is calculated as $\frac{TP+TN}{N+P}$, where N and P are the total number of negatives and positives, respectively. Surprisingly, we find that FOCES with slicing can achieve an even higher detection accuracy than that without slicing, except for the BCube(1,4) topology. This is probably because slicing can prevent noises from smoothing out the high anomaly index caused by forwarding anomalies. We further study the optimal threshold for FOCES using and without using slicing. Specifically, we vary the threshold from 0 to 100, and report the detection accuracy in Fig. 14. We can see that FOCES with slicing prefers a larger threshold compared with FOCES without slicing. This may further validate our guess that slicing can reduce the effect of noises.

In sum, the above experiments show that slicing can indeed reduce the computation overhead of FOCES, without sacrificing detection accuracy.

G. Experiment 5: Bandwidth Overhead

FOCES needs to periodically query counters from switches, thus we are interested in whether this will congest the switch-to-controller channel. In this experiment, we let FOCES request flow table of each switch from the controller, and sum up the size of each flow table. Table III shows that the total flow table size is relatively small for these topologies. As the detection period is on a scale of several seconds, it is expected that FOCES will not congest the channel from switches to the controller.

H. Experiment 6: Results for Larger Topologies

In the previous experiments, we only consider topologies with 20-26 switches, as shown in Table II. To demonstrate how FOCES scales to large networks, we redo the experiments 2, 3, and 4 using the FatTree(8) topology, which consists of 80 switches. Fig. 15 reports the ROC curves, detection precision, and the relationship between detection accuracy and threshold value. Fig. 16 reports the localization accuracy of baseline and voting-based method. We can see that the results are quite similar to those in Fig. 9, Fig. 10, Fig. 14, and Fig. 11.

I. Experiment 7: Speeding Up Detection With Multiple Cores

As already noted in Algorithm 2, our detection algorithm can slice the large FCM to reduce the computation complexity. More importantly, the algorithm can naturally be parallelized in order to scale to large networks. In this experiment, we evaluate how our parallelized detection algorithm can scale for large network topologies using multiple CPU cores.

We use the same setting as Experiment 4, *i.e.*, FatTree(8) as the topology. Different from Experiment 4, we fix the number of flows to around 12K, and use two Intel Xeon Gold 5118 CPUs @2.30GHz with 12 cores. Since our detection algorithm is implemented with Python, where global interpreter lock (GIL) prevents multiple threads running simultaneously, we use multiple processes to run simultaneously on multiple CPU cores. In addition, we use the Intel Math Kernel Library [38] to further speed up the matrix computation.

As shown in Fig. 17, initially the detection time is around 950 seconds, which is similar to that in Fig. 12. However, as we gradually increase the number of cores to 24, the time decreases to around 330 seconds. It is expected that our

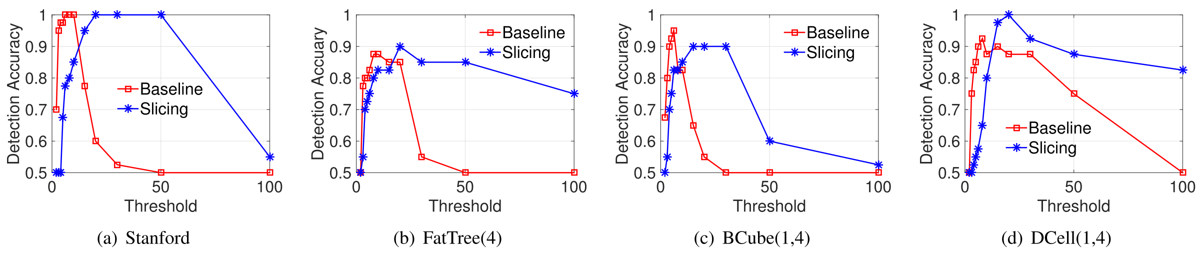


Fig. 14. The relationship between detection accuracy and threshold value for FOCES with and without slicing.

TABLE III
THE TOTAL SIZE OF FLOW TABLES REQUESTED BY FOCES IN DIFFERENT NETWORKS

	FatTree(4)	FatTree(6)	FatTree(8)	BCube(1,4)	BCube(1,6)	DCell(1,4)	DCell(1,6)
Number of Rules	1790	7396	30168	1513	7943	2969	13454
Total Flow Table Size (MB)	0.45	1.81	7.38	0.39	1.98	0.57	2.25

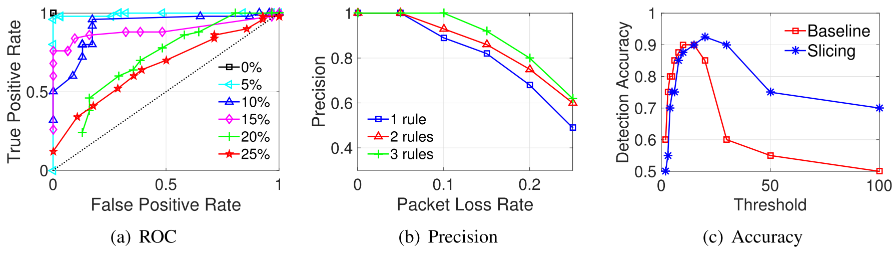


Fig. 15. The ROC curves, detection precision, and the relationship between detection accuracy and threshold value for FatTree(k = 8) topology.

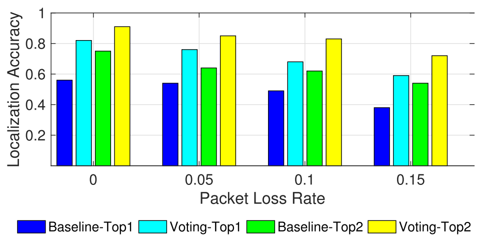


Fig. 16. The localization accuracy of baseline and voting-based method for FatTree(k = 8) topology. "Top1" refers to only selecting the rule/switch with the largest EI/AW, and "Top2" refers to selecting two rules/switches with the top-2 largest EI/AW.

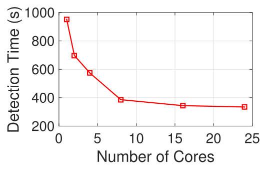


Fig. 17. The relationship between detection time of FOCES and the number of CPU cores. The FatTree(8) topology with around 12K flows is used for experiments.

algorithm can further speed up when implementing as a multi-thread program in C++.

VII. RELATED WORK

Many tools have been proposed to detect the forwarding anomalies in SDN. We broadly classify them into two categories: *path verification tools* and *statistics verification tools*.

Path verification tools try to detect forwarding anomalies by verifying whether the forwarding path took by packets are corresponding to the paths calculated by the controller. Path verification for the Internet has been extensively studied [29], [39], [40]. The basic idea is to let each router along the forwarding path embeds a cryptographic tag, *e.g.*, Message Authentication Code (MAC), into each packet, such that destination switch can check whether a packet has followed the path claimed by the sender. **SDNsec** [22], **REV** [23], and **WedgeTail** [25] apply path verification techniques in the new context of SDN.

In SDNsec [22], the controller pre-computes the path for each flow to be examined, and generates a forwarding entry for each switch along the path. Then, the controller instructs the ingress switch to embed these entries into packets, and each switches along the path forward packets according to the embedded forwarding entries. In addition, each switch also embeds a Message Authentication Code (MAC) into each packet to proof that it forwards the packet. One serious problem with SDNsec is that the overhead: to ensure every packet has followed the correct path, the egress switch needs to report all packets of the flow to the controller, which will incur high overhead on the control channel.

REV [23] reduces the overhead by proposing the *compressive MAC*, which allows switches to compress MACs before reporting to the controller. Using compressive MAC, the egress switch of a flow only needs to report a single *flow packet* to the controller. The authors prove that once the flow packet passes the verification, it holds with high probability that each packet of the flow has traversed the intended path.

WedgeTail [25] uses a similar approach, where the controller compares the expected and actual trajectories of packets to detect forwarding anomalies. Different from SDNsec and REV which checks a specific flow, WedgeTail aims to detect malicious forwarding node. To do so, it tries to identify the most frequently traversed nodes in the network, and analyzes the trajectories of packets that originating from ports of these devices.

One common drawback of the above path verification tools is that switches should be modified to compute tags and extra header space should be reversed to carry these tags, which can seriously limit their deployment in real networks.

Statistics verification tools try to detect forwarding anomalies by collecting and analyzing flow statistics. Compared with path verification tools, they do not need to add extra packet headers, or modify switch processing logics. The key idea is to leverage the "flow conservation principle", *i.e.*, if all packets of a flow are forwarded correctly, the counters of this flow at all switches along the forwarding path should be roughly the same [41], [42].

SPHINX [26] calculates a metric named *similarity index*, for each flow, based on the flow statistics collected from switches along the forwarding path of the flow. The index approximately measures the data volume for the flow, and honest switches tend to report a similar index. Then, SPHINX detects a malicious switch if its index differs much from the average index for the flow. SPHINX did not discuss how to deal with wildcard rules that can match multiple flows.

Chao *et al.* [27] propose to detect malicious switches by checking whether the counters of adjacent switches are consistent. Specifically, to monitor a rule r at a switch S, the controller installs dedicated counter rules (rules used solely for counting, without affecting forwarding behaviors) at each neighboring switch of S. These rules have the same matching fields with r, and count the number of packets flow in or out of switch S. Rule r passes the verification if the difference of these two numbers is below a threshold. Since monitoring each rule requires one or two counter rule for each neighboring switch, it may exhaust TCAM memory of switches if we need to check all the rules in the network. In addition, it requires cascaded flow tables, which are still not well supported by many SDN switches.

Based on a similar idea, **FADE** [28] installs dedicated flow rules at switches to collect flow counters, and checks the consistency among counters of the same flow. To minimize the cost of dedicated flow rules, FADE introduces the concept of *rule paths*, and designs algorithms to select the minimum number of flows to cover all rule paths. Similarly, FADE has large overhead as it needs more than two dedicated rules per flow for collecting statistics.

FlowMon [43] detects packet droppers and packet swappers in SDN, also based on the flow conversation principle. Different from [27] and [28], FlowMon does not require dedicated flows. However, it has a smaller detection scope as it only checks the per-port statistics, rather than per-flow statistics. This means that FlowMon may miss some carefully-crafted forwarding anomalies that preserve the conservation of per-port statistics.

Different from all the above statistics verification tools that detect forwarding anomalies on a per-flow or per-switch basis, FOCES can detect anomalies at network wide, since it analyzes the flow-counter equation system, which characterizes the network-wide forwarding behaviors. In addition, FOCES does not require dedicated counter rules, thus has no overhead on switch flow table space.

VIII. Conclusion

This paper presented FOCES, a new forwarding anomaly detection and localization method for Software Defined Networks (SDNs). Different from existing statistics verification tools that look at individual flows, FOCES can detect forwarding anomalies at a network wide, and does not need to install any dedicated rules. We theoretically analyzed the condition for FOCES to successfully detect forwarding anomalies, and used experiments to show FOCES can achieve a high detection and localization accuracy with small false positive rates. To make FOCES scale to large networks, we proposed a slicing-based method which can significantly reduce the computation cost. Our future work includes (1) designing an algorithm to adaptively determine the optimal threshold value, and (2) studying how the condition that an adversary can game the statistics-based methods including FOCES, in order to mislead them to make false judgments.

References

[1] P. Zhang *et al.*, "FOCES: Detecting forwarding anomalies in software defined networks," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 830–840.

[2] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[3] (2014). *Dell Advances Open Networking Initiative*. [Online]. Available: https://bit.ly/2HpE5Se

[4] (2020). *Pica8 White Box Switches*. [Online]. Available: https://bit.ly/35lfYwu

[5] (2020). *Switch Light OS*. [Online]. Available: https://bit.ly/37sqOTR

[6] (2020). *Cumulus Linux*. [Online]. Available: https://bit.ly/31yQryO

[7] (2019). *PicOS*. [Online]. Available: https://bit.ly/3jlfVFV

[8] (2005). *Cisco Security Hole a Whopper*. [Online]. Available: https://bit.ly/3dKS80Z

[9] (2014). *Snowden: The NSA Planted Backdoors in Cisco Products*. [Online]. Available: https://bit.ly/1PKtbQW

[10] (2015). *Cisco Routers Compromised by Malicious Code Injection*. [Online]. Available: https://bit.ly/1KtUoTs

[11] G. Pickett, "Staying persistent in software defined networks," in *Proc. Black Hat Briefings*, 2015, pp. 1–9.

[12] M. Antikainen, T. Aura, and M. Särelä, "Spook in your network: Attacking an SDN with a compromised OpenFlow switch," in *Proc. Nordic Conf. Secure IT Syst.*, 2014, pp. 229–244.

[13] P.-W. Chi, C.-T. Kuo, J.-W. Guo, and C.-L. Lei, "How to detect a compromised SDN switch," in *Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, Apr. 2015, pp. 1–6.

[14] (2015). *SDN Switches Aren't Hard to Compromise, Researcher Says*. [Online]. Available: https://bit.ly/3ohD4fS

[15] Open Networking Foundation. (2015). *OpenFlow Switch Specification (Version 1.5.1)*. [Online]. Available: https://bit.ly/31rNrUI

[16] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2013, pp. 151–152.

[17] P. Peresini, M. Kuzniar, and D. Kostic, "Monocle: Dynamic, fine-grained data plane monitoring," in *Proc. ACM CoNEXT*, 2015, pp. 1–13.

[18] K. Bu, X. Wen, B. Yang, Y. Chen, L. E. Li, and X. Chen, "Is every flow on the right track?: Inspect SDN forwarding with RuleScope," in *Proc. IEEE 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[19] P. Zhang, C. Zhang, and C. Hu, "Fast testing network data plane with RuleChecker," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–10.

[20] P. Zhang, H. Li, C. Hu, L. Hu, and L. Xiong, "Stick to the script: Monitoring the policy compliance of SDN data plane," in *Proc. Symp. Architectures Netw. Commun. Syst. (ANCS)*, 2016, pp. 81–86.

[21] P. Zhang *et al.*, "Mind the gap: Monitoring the control-data plane consistency in software defined networks," in *Proc. 12th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2016, pp. 19–33.

[22] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, "SDNsec: Forwarding accountability for the SDN data plane," in *Proc. 25th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2016, pp. 1–10.

[23] P. Zhang, "Towards rule enforcement verification for software defined networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2017, pp. 1–9.

[24] P. Zhang, H. Wu, D. Zhang, and Q. Li, "Verifying rule enforcement in software defined networks with REV," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 917–929, Apr. 2020.

[25] A. Shaghaghi, M. A. Kaafar, and S. Jha, "WedgeTail: An intrusion prevention system for the data plane of software defined networks," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 849–861.

[26] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting security attacks in software-defined networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 8–11.

[27] T.-W. Chao *et al.*, "Securing data planes in software-defined networks," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 465–470.

[28] C. Pang, Y. Jiang, and Q. Li, "FADE: Detecting forwarding anomaly in software-defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[29] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *Proc. ACM Conf. SIGCOMM (SIGCOMM)*, 2014, pp. 271–282.

[30] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow: Dependency-aware rule-caching for software-defined networks," in *Proc. Symp. SDN Res. (SOSR)*, 2016, pp. 1–12.

[31] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "Automatic test packet generation," in *Proc. 8th Int. Conf. Emerg. Netw. Experiments Technol. (CoNEXT)*, 2012, pp. 241–252.

[32] A. Strecok, "On the calculation of the inverse of the error function," *Math. Comput.*, vol. 22, no. 101, pp. 144–158, 1968.

[33] F. Pukelsheim, "The three sigma rule," *Amer. Statistician*, vol. 48, no. 2, pp. 88–91, May 1994.

[34] (2020). *Floodlight OpenFlow Controller*. [Online]. Available: https://github.com/floodlight/floodlight

[35] (2020). *Mininet*. [Online]. Available: http://mininet.org/

[36] (2020). *Open vSwitch*. [Online]. Available: http://openvswitch.org/

[37] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proc. USENIX NSDI*, 2012, pp. 113–126.

[38] (2020). *Intel Math Kernel Library*. [Online]. Available: https://software.intel.com/en-us/mkl

[39] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: Secure and adoptable source authentication," in *Proc. USENIX NSDI*, 2008, pp. 1–14.

[40] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra, "Verifying and enforcing network paths with ICING," in *Proc. 7th Conf. Emerg. Netw. EXperiments Technol. (CoNEXT)*, 2011, pp. 1–12.

[41] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson, "Detecting disruptive routers: A distributed network monitoring approach," in *Proc. IEEE Symp. Secur. Privacy*, Sep. 1998, pp. 50–60.

[42] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage, "Detecting and isolating malicious routers," *IEEE Trans. Dependable Secure Comput.*, vol. 3, no. 3, pp. 230–244, Jul. 2006.

[43] A. Kamisinski and C. Fung, "FlowMon: Detecting malicious switches in software-defined networks," in *Proc. Workshop Automated Decis. Making Act. Cyber Defense (SafeConfig)*, 2015, pp. 39–45.

Peng Zhang (Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University in 2013. He was a Visiting Researcher at The Chinese University of Hong Kong and Yale University. He is currently an Associate Professor with the School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China. He is also with the Ministry of Education (MOE) Key Laboratory for Intelligent Networks and Network Security, Xi'an Jiaotong University. His research interests include verification, measurement, and security in computer networks.

Fangzheng Zhang is currently pursuing the master's degree with the School of Computer Science and Technology, Xi'an Jiaotong University, China. His research interest includes network security.

Shimin Xu is currently pursuing the master's degree with the School of Computer Science and Technology, Xi'an Jiaotong University, China. His research interest includes software-defined networking.

Zuoru Yang received the B.Eng. degree in computer science from Xi'an Jiaotong University in 2018. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include data deduplication and security.

Hao Li (Member, IEEE) received the Ph.D. degree in computer science from Xi'an Jiaotong University in 2016. He is currently an Associate Professor with the School of Computer Science and Technology, Xi'an Jiaotong University. His main research interests include SDN/NFV and network measurement.

Qi Li (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University. He has ever worked with ETH Zurich and The University of Texas at San Antonio. He is currently an Associate Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network and system security, particularly in Internet and cloud security, mobile security, and big data security. He is currently an Editorial Board Member of IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING (TDSC) and ACM *DTRAP*.

Huanzhao Wang is an Associate Professor with the School of Computer Science and Technology, Xi'an Jiaotong University. Her research interests include software-defined networking and network security.

Chao Shen (Senior Member, IEEE) was a Research Scholar with Carnegie Mellon University from 2011 to 2013. He is currently a Professor with the School of Electronics and Information Engineering, Xi'an Jiaotong University, China, where he also serves as the Associate Dean of the School of Cyber Security. He is also with the Ministry of Education (MOE) Key Laboratory for Intelligent Networks and Network Security. His research interests include network security, human–computer interaction, insider detection, and behavioral biometrics.

Chengchen Hu (Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University. He is a Principal Engineer with Xilinx Inc., and is the Founding Director of Xilinx Labs Asia Pacific based in Singapore, currently leading research on networked processing systems. Prior to joining Xilinx in August 2017, he was a Professor and the Head of the Department of Computer Science and Technology, Xi'an Jiaotong University, China. His main research interests include monitor, diagnose, and manage the Internet, the cloud data center networks, and the distributed systems through hardware optimized and software-defined approaches. He has served on the organization committee and technical program committee of many conferences, including INFOCOM, IWQoS, ICC, GLOBECOM, ANCS, Networking, and APCC. He was a recipient of the New Century Excellent Talents in University Award from the Ministry of Education, China, a Fellowship from the European Research Consortium for Informatics and Mathematics (ERCIM), and a Fellowship of Microsoft "Star-Track" Young Faculty.