FOCES: Detecting Forwarding Anomalies in Software Defined Networks (Technical Report)

Peng Zhang*†, Shimin Xu*, Zuoru Yang*, Hao Li*†, Qi Li‡, Huanzhao Wang*, and Chengchen Hu*†
* Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China
† MOE Key Lab for Intelligent Networks and Network Security
‡ Graduate School at Shenzhen, Tsinghua University, Shenzhen, China

***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. Such per-flow detection methods have a limited detection scope: they look at one flow each time, thus can only check a limited 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 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. Experiments show FOCES can achieve a detection precision higher than 90% for four network topologies, even when packet loss rates are as high as 10%.**

***Index Terms*—Software defined networks, Forwarding anomaly, Equation system**

I. Introduction

Software defined network (SDN) promises a centralized, flexible, and programmable control of computer networks [1]. However, despite these benefits, SDN is still vulnerable to attacks. First, the operating systems (OSes) of SDN switches can be compromised [2]–[4]. A recent study shows that an attacker can hack the boot loader of switch OSes, so as to gain persistent control over SDN switches [2]. Second, the control channels between the controller and switches also lack security protection. Even OpenFlow recommends the usage of SSL/TLS, most SDN switch vendors just forgo this feature [5].

The above security vulnerabilities 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 policy, like flow isolation and waypoint traversal. For example, the control plane policy may require a specific flow go through a firewall, and forwarding anomaly can cause all packets of this flow bypass the firewall. Note that SDN by itself provides no means to detect forwarding anomaly: the controller only installs rules at switches, but cannot ensure the rules are correctly translated to forwarding behaviors.

Recently, many *data plane debugging* tools are proposed to test whether flow rules at switches are corresponding to the controller's view [6]–[8], or monitor whether the forwarding behaviors of packets are compliant with the control plane policies [9], [10]. However, they assume switches are trustable, thus cannot work when switches are compromised.

To work in adversarial setting, *path verification tools* let switches along the forwarding path embed some cryptographic information (*e.g.*, MAC) so that the controller can verify whether the actual path took by packets are agreeing with what the controller expects [11]–[13]. However, they need extra header space for storing the MACs, therefore introducing high bandwidth overhead. In addition, they need to modify switches to support cryptographic operations, which further increases the deployment cost.

Statistics verification tools try to detect forwarding anomalies by passively collecting and analyzing flow statistics [14]–[16]. These tools rely on the flow conservation principle, which has previously been used to detect compromised routers in traditional networks [17], [18]. 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, they look at flows or switches individually, and cannot detect forwarding anomalies at all switches simultaneously. In addition, they need to install dedicated counter rules at switches for collecting flow statistics, thus placing large overhead on flow table space.

To this end, this paper presents *FlOw Counter Equation System (FOCES)*, a new approach to forwarding anomaly detection for SDN. Generally speaking, FOCES belongs to statistics verification methods. However, different from all the per-flow 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 switch to the whole network, and thus can detect forwarding anomalies at the network-wide scale.

When designing FOCES, we are faced with the following challenges. First, packet losses and out-of-sync counter values may bring noises to FOCES, and cause it to falsely flag the network under forwarding anomaly. We show how to eliminate such false positives by designing a threshold-based detection algorithm, and study how to choose the appropriate threshold values both analytically and using experiments. Second, FOCES needs to solve flow counter equation systems, which requires computing matrix inversions. This can be costly when there are a large number of flows and rules, To make FOCES scalable, we propose a method by slicing the original large FCM into many smaller sub-FCMs, whose matrix inversion can be computed much faster.

In sum, our contribution is four-fold:

- We propose FOCES, a network-wide forwarding anomaly detection method in SDN, which can check whether all flows in a network are forwarded correctly simultaneously, without installing extra rules.
- We theoretically analyze the condition for successful detection using FOCES, and reduce the condition to the problem of finding a loop in a bipartite graph.
- We design a threshold-based detection algorithm to eliminate false positives caused by counter noises, and a slicing-based method to make FOCES scalable for larger networks.
- We use experiments to show FOCES can accurately detect forwarding anomalies in four network topologies, with minimal computation 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 shows how to make FOCES work in realistic setting, by dealing with noises and scalability issues; Section V analyzes the capability of FOCES in detecting forwarding anomalies; 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 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 [1]. Each rule consists of three parts: *matching fields*, *actions*, and *counters*. Switches forward packets by looking up in the flow table. 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 the complete network topology and can request counters of rules from switches [19]. In addition to the above proactive mode of rule installation, rules can also be installed reactively when a new flow comes into the network without any matching rules.

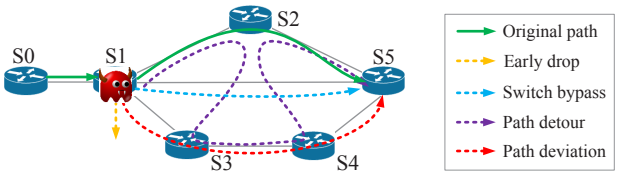


Fig. 1. Illustration of forwarding anomalies considered in this paper.

B. Threat Model

The adversary aims to change the paths that packets should be 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 what are expected 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 is not compromised, as it can drop packets pretending that packets are received by the end hosts.

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.

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.

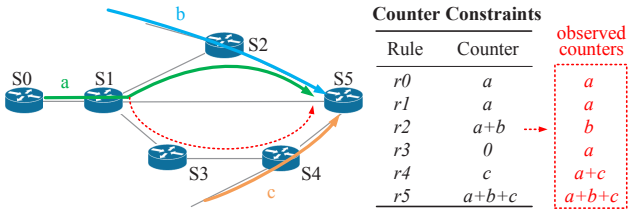


Fig. 2. An example illustrating the basic idea of FOCES.

III. FOCES: Theoretical Construction

This section presents the theoretical construction of *FlOw Counter Equation System (FOCES)*, a network-wide forwarding anomaly detection algorithm for SDN. By network-wide, we mean whenever a flow in the network is experiencing forwarding anomaly, FOCES can immediately detect it through analysis. We will first present the basic idea of FOCES, followed by the theoretical construction. Then, we analyze the condition for successful detection.

A. Basic Idea

Before introducing FOCES, we first show how previous statistics verification methods can detect forwarding anomalies by leveraging the "flow conservation" principle [17]. 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.

Based on the above idea, previous works check whether the counters of a flow conform to the flow conservation principle, in both traditional networks [17], [18], and SDN [14]–[16]. However, applying the flow conservation principle for each individual flow has two serious limitations.

- **Limited Detection Scope.** Since they look at one flow each time, mostly they can only check a limited number of flows (*e.g.*, flows passing a specific switch, flows destined to a specific IP address, etc.), instead of checking all flows in the network simultaneously. As a result, they may miss some forwarding anomalies happening to flows that are not checked. For example, if we only check flows passing S_2 in Fig. 2, we may miss the forwarding anomaly 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, but has to install dedicated rules to collect the statistics of that flow. 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. A dedicated rule that solely matches the flow 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 flow table size is generally small for SDN switches.*

The key idea of FOCES is to extend the flow conservation principle from *individual* flows to *a network of* flows, by considering the relationship between flows and rules. For example, in Fig. 2, 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 table. Suppose the green flow of volume a is directed 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 at a network scale. In addition, we directly use counters of aggregate rules, without installing dedicated counter rules.

The Workflow of FOCES. Based on the above idea, FOCES first extracts the constraints that all counters in the network should satisfy, from the network configurations (*e.g.*, flow tables) in the control plane. Then, FOCES collects counters from the data plane, and checks whether these counters satisfy the constraints. If the constraints are violated, then there must be some forwarding anomalies in the network.

In the remaining of this section, we will present the construction of flow counter equation system to realize the above idea, and then clearly define the detection boundary of FOCES, *i.e.*, under what condition can FOCES successfully detect forwarding anomalies.

B. Flow Counter Equation System

We show how to express the constraints for 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 make it work in realistic setting.

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

$$H_{i,j} = \begin{cases} 1 & \text{if flow } f_j \text{ matches rule } r_i \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Then, define the *counter vector* as $Y = (y_1, y_2, ..., y_m)^T$, where y_i is the counter value of rule r_i, and the *volume vector* as $X = (x_1, x_2, ..., x_n)^T$, 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*:

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

Let the volume vector be X_0, then the counter vector should be $Y_0 = HX_0$. Suppose due to attacks the FCM is changed to $H^{'} \neq H$. Then, the observed counter vector will become $Y^{'} = H^{'}X_0 \neq Y_0$, Since the controller has no idea what $H^{'}$ is, when it tries to recover X_0, it needs to solve the flowing flow-counter equation system:

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

Since $m > n$, Eq. (3) is an overdetermined equation system. Also, since $Y^{'} = H^{'}X_0$, 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_0 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 $X_0 = (a, b, c)^T = (3, 4, 5)^T$, 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 establish the relationship between flows and rules, *i.e.*, 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 equivalence classes. A logical flow is defined as a class of packets that experience the same set of rules in the network.

Specifically, we adopt the approach in ATPG [20] to generate the all-reachability table. 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 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 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 matching 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.

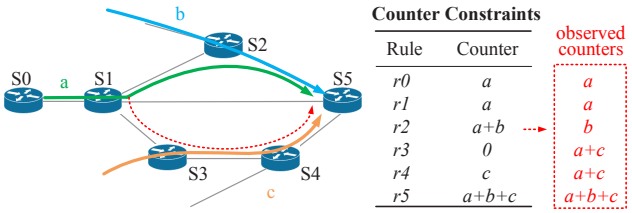


Fig. 3. A counterexample where FOCES misses the forwarding anomaly.

C. The Condition for Successful Detection

In what follows, we will first give a counterexample, where a forwarding anomaly does not violate counter constraints. 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. It may also help us carefully design rules such that FOCES can always detect forwarding anomalies, which is left as one of our future work.

Counterexample for FOCES. Fig. 3 shows an example, which is much the same with that 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 $X_0 = (a, b, c)^T = (3, 4, 5)^T$. 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 for this is that $HX = Y^{'}$ is a consistent equation system. Thus, the condition for successful detection is equivalent to the condition for the equation system

to be inconsistent. In the following, we will analyze such condition from linear algebraic point of view.

Analysis on Detectability. For sake of analysis, we consider a specific network with m forwarding rules $r_1, r_2, \ldots, r_m$, and n flows $f_1, f_2, \ldots, f_n$. Let the FCM of the network be $H_{m\times n}$, which can be represented 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. Let the volume vector be $X_0 = (x_1, x_2, \ldots, x_n)^T$, *i.e.*, flow f_i has volume x_i. In the following, we define what is a forwarding anomaly and when it is detectable.

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 the 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 loops in a bipartite graph, which is much easier and intuitive to apply.

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 rules r in 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. For sake of later proof, we also add a virtual rule r_s acting as the first rule of all flows. Formally, we have:

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

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. 3. The RBG of S_2 is shown in Fig. 4. We can see that there is a loop marked in green dashed lines. The labels besides the edges are the volumes of flows. By applying the operations 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.

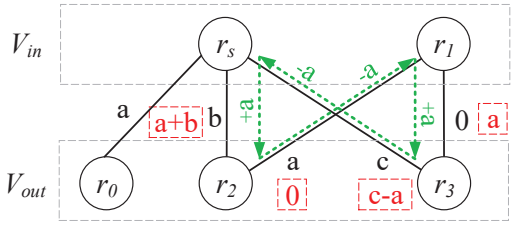


Fig. 4. The flow rule graph for S_2 in Fig. 3.

IV. FOCES: Make it work

The last section shows how FOCES can detect forwarding anomalies in ideal settings. To make FOCES to work in realistic settings, we are still faced with two challenges: (1) **Noises.** In realistic settings, both packet losses and out-of-sync counters can result in $\Delta \neq 0$. We need to eliminate the impact of such noises. (2) **Scalability.** Computing Δ requires calculating the inverse of FCM, which is expensive when there are a large number of rules and flows. In the following, we show how we resolve the above two challenges.

A. Threshold-based Detection Algorithm

In the following, we first define the *anomaly index* to measure the possibility of forwarding anomaly. If the anomaly index is higher than a threshold T, then we say there are forwarding anomalies. Then, we discuss how to set the threshold properly so as to minimize false positives.

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 of flows. 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 $AI = +\infty$. Thus, FOCES judges that there are forwarding anomalies since $AI > T$, where T is a threshold value determined as follows.

Detection threshold. Here, we assume each element $Y^{'}(i)$ is conforming to the normal distribution $N(Y_0(i), \sigma^2)$, where $Y_0(i)$ is the mean and σ is the standard variance. Therefore, 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 [21]. By solving $F(x) = 1/2$, we have $x = \sqrt{2}erf^{-1}(1/2)\sigma \approx 0.675\sigma$. Thus, we

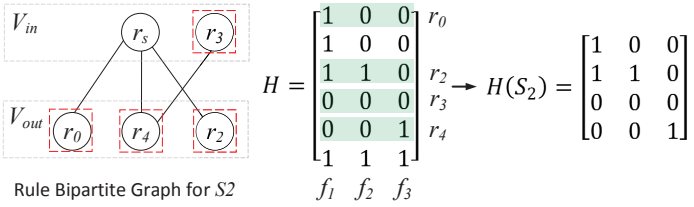


Fig. 5. The construction of sub-FCM for S_2 in Fig 2.

use 0.675σ as an approximate for Err_{med}. According to the three-sigma rule in statistics [22], 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).

Algorithm 1 summarizes the threshold-based detection algorithm of FOCES.

Algorithm 1: `Detect_Anomaly_Baseline`$(Y^{'}, H_{m\times n}, T)$

Input: $Y^{'} = (y_1, y_2, ..., y_m)^T$: the counter vector, $H_{m\times n}$: the Flow-Counter Matrix (FCM), T: the detection threshold.
Output: `True (Anomaly)` or `False (Normal)`.

Compute the volume vector estimate $\hat{X} \leftarrow (H^T H)^{-1} H^T Y^{'}$;
Compute the counter vector estimate $\hat{Y} \leftarrow H\hat{X}$;
Compute the error vector $\Delta \leftarrow |Y^{'} - \hat{Y}|$;
$Err_{max} \leftarrow$ maximum of Δ;
$Err_{med} \leftarrow$ median of Δ;
$AI \leftarrow \frac{Err_{max}}{Err_{med}}$;
if $AI > T$ **then**
  return `True`;
else
  return `False`;
end

B. Making FOCES Scalable with Matrix 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 RGB 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. For switch S_2 in Fig 2, Fig 5 shows its RBG $\mathcal{G}_{S_2}(V_{in}, V_{out}, E)$. 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 detection process of FOCES with slicing.

Algorithm 2: `Detect_Anomaly_Slicing`$(Y^{'}, H_{m\times n}, T, n)$

Input: $Y^{'} = (y_1, y_2, ..., y_m)^T$: 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)`.

foreach $i \leftarrow 1$ **to** n **do**
  Computer the sub-FCM $H(i)$ for switch S_i;
  Extract the sub-vector $Y^{'}(i)$ for switch S_i from $Y^{'}$;
  Compute the volume vector estimate $\hat{X} \leftarrow (H(i)^T H(i))^{-1} H^T Y^{'}(i)$;
  Compute the counter vector estimate $\hat{Y}(i) \leftarrow H(i)\hat{X}$;
  Compute the error vector $\Delta(i) \leftarrow |Y^{'}(i) - \hat{Y}(i)|$;
  $Err_{max} \leftarrow$ maximum of $\Delta(i)$;
  $Err_{med} \leftarrow$ median of $\Delta(i)$;
  $AI \leftarrow \frac{Err_{max}}{Err_{med}}$;
  if $AI > T$ **then**
    return `True`;
  end
end
return `False`;

Analysis on Detection Equivalence. The following theorem says that FOCES with slicing is equivalent to the baseline threshold-based algorithm 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. In the following, we analyze the computation complexity of FOCES with and without slicing, respectively. Since the computation complexity depends on network topology and configuration, here we pick Fattree as a representative to illustrate the analysis process.

In the Fattree ($k = n$) topology, the time complexity of FOCES without slicing approximately equals to that of matrix inversion $\mathcal{O}(N^3)$, where N is the size of the FCM, *i.e.*, the number of flows in the network. Since the Fattree ($k = n$) has $\frac{n^3}{4}$ hosts, the number of flows will be $N = \mathcal{O}(n^6)$. The number of switches is $\mathcal{O}(n^2)$. The average number of flows passing a specific switch is $\mathcal{O}(\frac{n^6}{n^2}) = \mathcal{O}(n^4)$, and the time to apply FOCES on one switch is $\mathcal{O}(n^4)^3 = \mathcal{O}(n^{12})$. Thus, the computation complexity of FOCES with slicing is $\mathcal{O}(n^{12}) \times \mathcal{O}(n^2) = \mathcal{O}(n^{14}) \approx \mathcal{O}(N^{2.3})$, that is, we reduce the

computation complexity from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^{2.3})$, by using slicing. We will use experiments to further demonstrate the reduction of computation time by using slicing in Section VI-F.

Finally, note that slicing can also help FOCES to pinpoint the malicious switches: if the anomaly index for one switch is high, then it is possible that this switch on its last hop is responsible for the forwarding anomalies. We leave the localization of malicious switches as one of our future work.

V. Security Analysis

This section analyzes the security of FOCES. We discuss how FOCES detects packet early drops and path deviation 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.

VI. Implementation and Evaluation

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

1) Can FOCES *effectively* detect forwarding anomalies?
2) How *accurately* can FOCES detect forwarding anomalies, under different packet loss rates, with different number of rules being modified, for different network topologies?
3) Whether slicing can help FOCES achieve a *faster* detection without loss of accuracy?

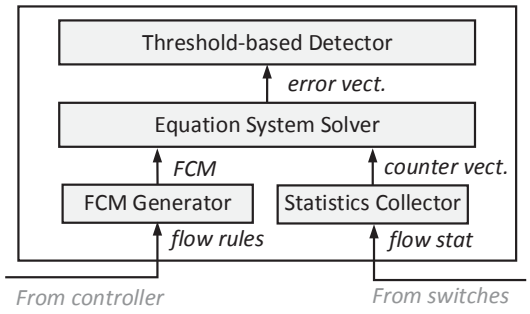


Fig. 6. The system architecture of FOCES.

A. Implementation

Fig. 6 shows the system architecture of FOCES, which mainly contains four parts: (1) the *FCM Generator* retrieves flow rules from the controller, calculates all flows in the network (a flow is a set of packets that match the same set of flow rules), and constructs the FCM according to Eq. (1); (2) the *Statistics Collector* periodically queries switches for flow statistics, and extracts the counters of rules to construct the counter vector; (3) the *Equation System Solver* solves the equation system determined by the FCM and counter vector, to obtain an estimate of the flow volume vector, and calculates the error vector of that estimate; (4) the *Threshold-based Detector* makes the decision on whether there are forwarding anomalies, by computing the forwarding anomaly index from the error vector, and comparing the index with the threshold.

We prototype FOCES with approximately 1500 lines of Python codes. The implementation details are as follows. The *Flow Counter Matrix (FCM) Generator* periodically retrieves flow tables of switches via the Floodlight REST API. Then, it constructs the FCM according to Section III-B, based on ATPG [20]. The FCM is stored as a sparse matrix using the Python `sparse` library. The *Statistics Collector* periodically collects flow statistics by querying the controller, also via the Floodlight REST API. Then, it extracts the counter values to construct the counter vector. The *Equation System Solver* takes the FCM and counter vector as input, and computes a least square solution to Eq. (4). We use the `NumPy` library in Python to compute matrix inversion and transpose.

B. Experiment Setup

We use Floodlight v2.1 [23] as the controller, and use Mininet [24] to generate different network topologies, consisting of Open vSwitches [25]. 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) [26], FatTree(4), BCube(1,4), and DCell(1,4). For Stanford, we attach one host for 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 I. For each network, we generate a flow of the same rate between

TABLE I
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

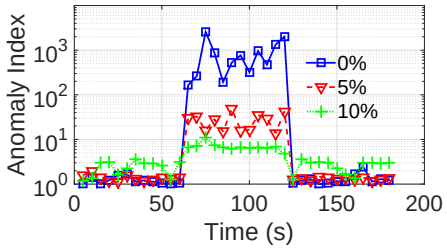


Fig. 7. The anomaly indices with and without forwarding anomalies, respectively. BCube(1,4) is used for the experiment.

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 populates them to switches. To simulate forwarding anomalies, we randomly choose switches from the network, and randomly modify flow rules in the switches' flow tables.

C. Experiment 1: Functional Test

In this experiment, we test whether FOCES can detect forwarding anomalies. 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. 7 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 indices for normal and anomaly cases become less distinguishable.

D. Experiment 2: Detection Accuracy vs. Detection Threshold

In this experiment, we evaluate the detection accuracy of FOCES, and how it is affected by the detection threshold. We use the receiver operating characteristic (ROC) curve, which plots the True Positive (TP) rate against the False Positive (FP) rate. 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. 8 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%, when the threshold is set to 4.5.

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 high detection accuracy for moderate packet loss rates.

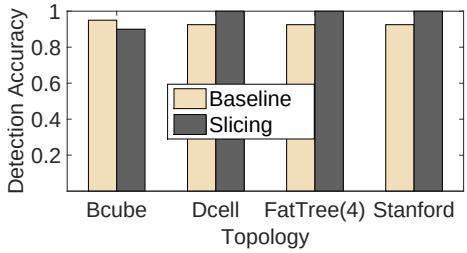


Fig. 10. The detection accuracy of FOCES with and without slicing, when the optimal threshold value is set.

E. Experiment 3: Detection Precision vs. Number of Anomalies

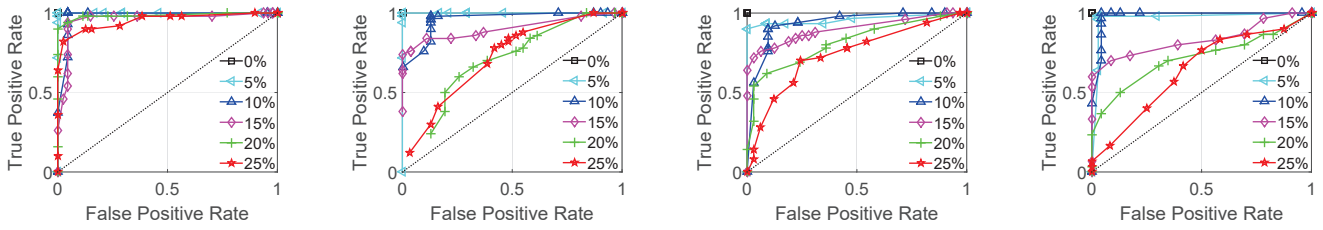
In this experiment, we evaluate how precise is the detection result of FOCES, and how the precision is affected when there are different number of forwarding anomalies. To quantify precision, we use $\frac{TP}{TP+FP}$, which indicates what percentage of samples that are marked as forwarding anomalies are actually such. In the 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. 9 reports the relationship between the precision of FOCES and the 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.

F. Experiment 4: The Effectiveness of Slicing

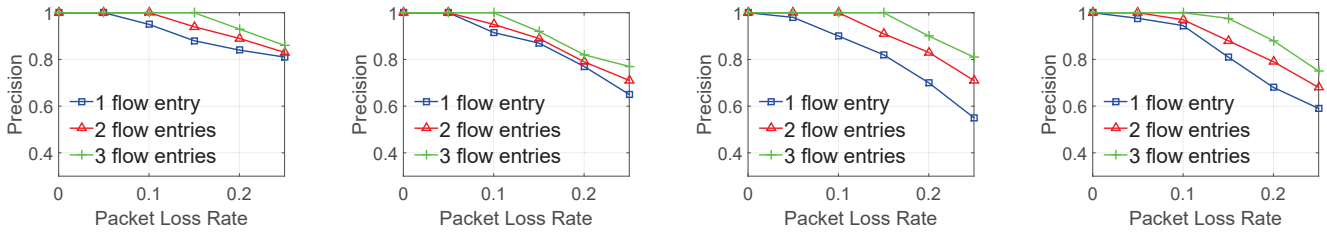
In this experiment, we study whether slicing can help FOCES to reduce the computation time while maintaining high detection accuracy.

Detection Accuracy. Fig. 10 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 with slicing, FOCES



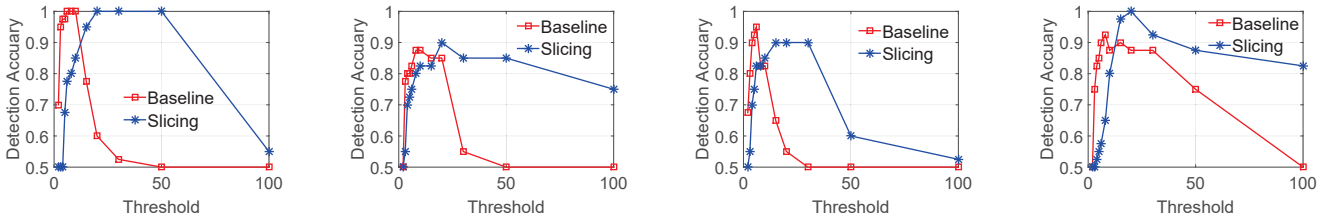
(a) Stanford (b) FatTree(4) (c) BCube(1,4) (d) DCell(1,4)

Fig. 8. The ROC curves for four different topologies.



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

Fig. 9. The detection precision for four different topologies.



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

Fig. 11. The relationship between detection accuracy and threshold value for FOCES with and without slicing.

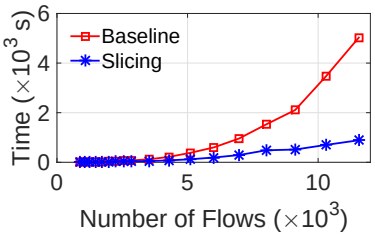


Fig. 12. The detection time for different number of flows.

can achieve an even better detection accuracy, except for the BCube(1,4) topology. This is probably because dealing with each switch can prevent noises of benign forwarding behaviors 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. 11. We can see that FOCES with slicing prefers a larger threshold compared with that of not using slicing. This is perhaps due to the fact that slicing can help FOCES eliminate noises.

Computation Time. We continue to evaluate how slicing helps FOCES to 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 is roughly the same when there are less than 5K flows. However, the computation time grows much slower than the baseline FOCES. Specifically, when there are around 12K flows, the computation time of FOCES with slicing is less than 20% of that of the baseline FOCES. This means that we can effectively reduce the computation overhead of FOCES by slicing.

VII. Related Work

Many tools have been proposed to detect the forwarding anomalies and localize the malicious switches 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 [27]–[29]. The basic idea is to let each router along the forwarding path embeds a Message Authentication Code (MAC) for each packet, such that destination switch can check whether a packet has followed the path claimed by the sender. **SDNsec** [11], **REV** [12], and **WedgeTail** [13] apply path verification in the new context of SDN.

In SDNsec [11], 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. Then, by letting egress switches report the packets, the controller can check whether the real forwarding path of these packets is consistent with the expected one. 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 [12] 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 [13] 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. If the actual trajectories differ from the real trajectories, then it identifies the first switch where these trajectories diverge as the malicious switch.

One common drawback of the above path verification tools is that extra packet header space should be reversed to carry forwarding paths or cryptographic information, and switches should be modified to embed tags into packets, 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 principles, *i.e.*, if all packets of a flow are forwarded along the expected path, the counters for this flow should be roughly the same [17], [18].

Chao *et al.* [14] try to detect malicious switches by checking consistency of flow statistics. 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** [15] generates and installs dedicated flow rules at switches to collect flow counters, and check 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 [16] collects and analyzes port statistics to detect packet droppers and packet swappers in SDN, also leveraging the flow conversation principle. Different from [14] and [15], 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 on a network-wide scale, 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, which can detect forwarding anomalies in Software Defined Networks (SDNs). FOCES captures the correct forwarding behaviors as a linear equation system, *i.e.*, flow counter equation system, and tries to detect forwarding anomalies by analyzing whether the counters of rules in the network can fit into this equation system. 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 accuracy with small false positive rates. To make FOCES scale to large networks, we proposed a slicing-based method to reduce the computation cost. We showed the slicing-based method can reduce the computation overhead by 80% when there are 12K flows.

Our future work includes: (1) designing algorithms to localize the compromised switches that cause forwarding anomalies; (2) studying how to install rules which meet the detection conditions of FOCES, such that all possible forwarding anomalies can be detected.

IX. Acknowledgement

The authors would like to thank all the anonymous reviewers for their valuable comments. This work is supported by the National Key Research and Development Program of China (No. 2017YFB0801703, 2016YFB0800100), the National Natural Science Foundation of China (No. 61772412, U1736205, 61702407, 61672425, 61572278, U1736209), and the Fundamental Research Funds for the Central Universities. The first author is supported by the K. C. Wong Education Foundation.

References

[1] 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 CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[2] G. Pickett, "Staying persistent in software defined networks," in *Black Hat Briefings*, 2015.

[3] M. Antikainen, T. Aura, and M. Särelä, "Spook in your network: Attacking an SDN with a compromised OpenFlow switch," in *Secure IT Systems*, 2014.

[4] P.-W. Chi, C.-T. Kuo, J.-W. Guo, and C.-L. Lei, "How to detect a compromised SDN switch," in *IEEE NetSoft*, 2015.

[5] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *ACM HotSDN*, 2013.

[6] P. Peresini, M. Kuzniar, and D. Kostic, "Monocle: Dynamic, fine-grained data plane monitoring," in *ACM CoNEXT*, 2015.

[7] 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 *IEEE INFOCOM*, 2016.

[8] P. Zhang, C. Zhang, and C. Hu, "Fast testing network data plane with RuleChecker," in *IEEE ICNP*, 2017.

[9] P. Zhang, H. Li, C. Hu, L. Hu, and L. Xiong, "Stick to the script: Monitoring the policy compliance of SDN data plane," in *ACM/IEEE ANCS*, 2016.

[10] P. Zhang, H. Li, C. Hu, L. Hu, L. Xiong, R. Wang, and Y. Zhang, "Mind the gap: Monitoring the control-data plane consistency in software defined networks," in *ACM CoNEXT*, 2016.

[11] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, "SDNsec: Forwarding accountability for the SDN data plane," in *IEEE ICCCN*, 2016.

[12] P. Zhang, "Towards rule enforcement verification for software defined networks," in *IEEE INFOCOM*, 2017.

[13] A. Shaghaghi, M. A. Kaafar, and S. Jha, "WedgeTail: An intrusion prevention system for the data plane of software defined networks," in *Proceedings of ACM AsiaCCS*, 2017.

[14] T. W. Chao, Y. M. Ke, B. H. Chen, and J. L. Chen, "Securing data planes in software-defined networks," in *IEEE Netsoft*, 2016.

[15] C. Pang, Y. Jiang, and Q. Li, "FADE: Detecting forwarding anomaly in software-defined networks," in *IEEE ICC*, 2016.

[16] A. Kamisinski and C. Fung, "FlowMon:detecting malicious switches in software-defined networks," in *ACM CCS Workshop on Automated Decision Making for Active Cyber Defense*, 2015, pp. 39–45.

[17] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson, "Detecting disruptive routers: A distributed network monitoring approach," *IEEE Symposium on Security and Privacy*, 1998.

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

[19] Open Networking Foundation, "OpenFlow switch specification (version 1.5.0)," 2014.

[20] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "Automatic test packet generation," in *ACM CoNEXT*, 2012.

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

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

[23] "Floodlight OpenFlow Controller," http://floodlight.openflowhub.org/.

[24] "Mininet," http://mininet.org/.

[25] "Open vSwitch," http://openvswitch.org/.

[26] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *USENIX NSDI*, 2012.

[27] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: Secure and adoptable source authentication," in *NSDI*, 2008.

[28] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller, and A. Seehra, "Verifying and enforcing network paths with ICING," in *ACM CoNEXT*, 2011.

[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 *ACM SIGCOMM*, 2014.

Appendix

A. Proof of Theorem 1

According to Eq. (2), we have:

$$x_1h_1 + x_2h_2 + ... + x_ih_i^{'} + x_nh_n = Y^{'} \tag{10}$$

Suppose the anomaly is undetectable, then we know that the equation system $HX = Y^{'}$ should be consistent, and let $\hat{X}$ be its solution. Then we have:

$$\hat{x}_1h_1 + \hat{x}_2h_2 + ... + \hat{x}_ih_i^{'} + \hat{x}_nh_n = Y^{'} \tag{11}$$

Combining the above two equations, we have:

$$\sum_{j\neq i}(x_j - \hat{x_j})h_j + x_ih_i - \hat{x_i}h_i^{'} = 0 \tag{12}$$

That is, $h_i^{'}$ lies in the linear subspace generated by $h_1, h_2, \ldots, h_n$.

B. Proof of Theorem 2

Minimum linearly dependent set. Suppose there is a forwarding anomaly $FA(h_i, h_i^{'})$, and $h_i^{'}$ lies in the linear subspace generated by $h_1, h_2, ..., h_n$, then we know $h_1, h_2, \ldots, h_n, h_i^{'}$ are linearly dependent. We define a *minimum linearly dependent set* with respect to $FA(h_i, h_i^{'})$ as the minimum subset of $\{h_1, h_2, \ldots, h_n, h_i^{'}\}$, such that the vectors in this subset are linearly dependent. Note that there can be more than one minimum linearly dependent sets. Without loss of generality, let $\{h_1, h_2, \ldots, h_k\},\ k \leq n+1$ be one such set. Then, we have:

$$c_1h_1 + c_2h_2 + \cdots + c_kh_k = 0,\ c_1, c_2, \ldots, c_k \neq 0, \tag{13}$$

where c_j is termed as the *coefficient* of flow h_j.

Let $H_{min} = (h_1, h_2, \ldots, h_k)$ be the corresponding FCM, and R_{min} be the set of rules r satisfying that there is at least one flow h in H_{min} matching rule r. Then, we can construct what we term as the min-RGB $\mathcal{G}_S^{H_{min}} \subset \mathcal{G}_S^{\tilde{H}}$ from H_{min} and R_{min}, according to Definition 3. In the following we assign values to the edges and vertices of $\mathcal{G}_S^{H_{min}}$.

Let the sub-RGB of S be $\mathcal{G}_S^{H_{min}}(V_{in}, V_{out}, E)$. For each edge $e = (r_i, r_j) \in E$, let $Flows(e)$ be the set of flows $h \in H_{min}$ such that $r_i \xrightarrow{h} r_j$. Define $Value(e)$ as the sum of coefficients of flows in $Flows(e)$ *i.e.*, $Value(e) = \sum_{h_j \in Flows(e)} c_j$. For each $r \in V_{in} \cup V_{out}$, let $Edges(r) \in E$ be set of edges of r. Define $Value(r)$ as the sum of values of edges in $Edges(r)$, *i.e.*, $Value(r) = \sum_{e \in Edges(r)} Value(e)$.

Lemma 1. Suppose there is a forwarding anomaly $FA(h_i, h_i^{'})$, and $h_i^{'}$ lies in the linear subspace generated by

$h_1, h_2, \ldots, h_n$. Then, there exists a switch S, such that for each $r \neq r_s$ in its min-RGB $\mathcal{G}_S^{H_{min}}$, we have $Value(r) = 0$.

Proof: Let $r \neq r_s$ be any rule in the min-RBG $\mathcal{G}_S^{H_{min}}$. According to our definition, $Value(r) = \sum_{h_i \in Flows(e), e \in Edges(r)} c_i$. That is, $Value(r)$ is the sum of the coefficients c_i of flows $h_i \in H_{min}$ that match r. Since h_i has value 1 in the row corresponding to r if h_i matches r, and 0 otherwise, $Value(r)$ equals its corresponding row of vector $\sum_i c_i h_i$. Since $\sum_i c_i h_i$ is an all-zero vector, we have $Value(r) = 0$ for all r in the min-RBG $\mathcal{G}_S^{H_{min}}$. ■

To proceed, we define several types of rules. If $\exists r_i, r_j, r_k$ $(i \neq j \neq k)$, $r_i \xrightarrow{h_a} r_j$ for some h_a, and $r_i \xrightarrow{h_b} r_k$ for some h_b, then we say r_i is a *separation rule* for h_a and h_b. If $\exists r_i, r_j, r_k (i \neq j \neq k)$, $r_i \xrightarrow{h_a} r_k$ for some h_a, and $r_j \xrightarrow{h_b} r_k$ for some h_b, then we say r_k is an *aggregation rule* for h_a and h_b. The definition of aggregation rule is transitive: if r_k is an aggregation rule for h_a and h_b, and $r_k \xrightarrow{h_a} r_l$ and $r_k \xrightarrow{h_b} r_l$, then r_l is also an aggregation rule for h_a and h_b. If r_k is both an aggregation rule and separation rule for h_a and h_b, then we say r_k is an *pivot rule* for h_a and h_b.

Lemma 2. Let $\mathcal{G}_S^{H_{min}}$ be a min-RGB of a switch S, with $Value(r) = 0$ for all $r \neq r_s$. Suppose there are no pivot rules, then there must exist at least one edge $e \in E$ with $Value(e) \neq 0$,

Proof: We prove it by contradiction. Suppose there is no such an edge, *i.e.*, all edges have value 0. Then, we show there must be a pivot rule.

Let h_m be the flow with the longest path in H_{min}, and let $head(h_m) = r_m$, where $head(h)$ is the first rule (except the virtual rule r_s) matched by a flow h. According to Lemma 1, there must exist another flow $h_x \in H_{min}$ also matching r_m (otherwise, we cannot make $Value(r_m) = 0$).

First, we will show $head(h_x) = r_m$ by contradiction. Suppose $head(h_x) = r_x \neq r_m$, then r_m is an aggregation rule for h_m and h_x. There must exist a separation rule r_k for h_m and h_x after r_m. Otherwise, h_m and h_x will match the same set of rules after r_m, and h_x has a longer path than h_m. This contradicts our assumption that h_m has the longest path. Since h_k is also an aggregation rule for h_m and h_x, following r_m is an aggregation rule for h_m and h_x, we know h_k is a pivot rule for r_m and h_x, contradicting the assumption that there are no pivot rules. Thus, we have $head(h_x) = r_m$.

Second, we show if $head(h_x) = r_m$, then there must be a pivot rule. Since $head(h_x) = r_m$, there should be a separation rule for h_m and h_x; otherwise, h_m and h_x will be the same flow. Let r_k be the last separation rule for h_m, and $r_k \xrightarrow{h_m} r_{k+1}$. Then there must exist a edge $e = (r_k, r_{k+1})$ in the $\mathcal{G}_S^{H_{min}}$. Since $Value(e) = 0$, there are are least two flows match r_k and r_{k+1} in sequence. Let h_n be another flow such that $r_k \xrightarrow{h_n} r_{k+1}$. Since r_k is the last separation rule, then h_m and h_n will match the same set of rules after r_k. In addition, there should exist an aggregation rule for h_m and h_n; otherwise, h_m and h_n will match the same set of rules before r_k, making $h_m = h_n$. Thus, r_k is a pivot rule for h_m and h_n, contradicting the assumption that there are no pivot rules.

Given the above discussion, we know there exists at least one edge $e \in E$ that $Value(e) \neq 0$. ■

Lemma 3. Let $\mathcal{G}_S^{H_{min}}$ be a min-RBG of a switch S, with $Value(r) = 0$ for all $r \neq r_s$, and an edge $e \in E$ with $Value(e) \neq 0$, Then, we there is a sub-RBG $\mathcal{G} \subset \mathcal{G}_S^{H_{min}}$, where each rule has a degree equal or larger than 2.

Proof: For $r \neq r_s$, since $Value(r) = 0$ and there is an edge e with $Value(e) \neq 0$, we can construct the sub-RBG by only selecting those non-zero edges. Each rule r will have at least two edges; otherwise, we cannot make $Value(r) = 0$. ■

Lemma 4. For an undirected graph $G(V, E)$, if the degree of each vertex is equal or larger than 2, then there must be at least one loop in $G(V, E)$.

Proof: We prove it by contradiction. Suppose there is no loop in $G(V, E)$, then we can always find a longest path P that traverses each vertex at most once. Let (v_{n-1}, v_n) be last edge in P. Since the degree of v_n is equal or larger than 2, then we have another edge (v_n, v_{n+1}). If v_{n+1} have not appeared on path P, then we can find a loop by adding (v_n, v_{n+1}) into P. Otherwise, we can extend the path P by one hop after (v_n, v_{n+1}) into P, resulting in a longer path. This contradicts the assumption that P is the longest path. ■

Lemma 5. Suppose there is a forwarding anomaly $FA(h_i, h_i^{'})$, and there are no pivot rules. Then, $h_i^{'}$ lies in the linear subspace generated by $h_1, h_2, \ldots, h_n$ if and only if there is a switch S whose RBG with respect to $\tilde{H} \triangleq H \cup \{h^{'}\}$, , *i.e.*, $\mathcal{G}_S^{\tilde{H}}(V_{in}, V_{out}, E)$, contains a loop, and the loop consists of two rules r_a and r_b, such that $r_a \xrightarrow{h_i^{'}} r_b$.

Proof: (1) "⇐". Suppose the RBG $\mathcal{G}_S^{\tilde{H}}(V_{in}, V_{out}, E)$ of a switch S contains a loop. We choose a minimum loop, in the sense that each rule has a degree of 2. Then, we construct a new RBG $\mathcal{G}(V_{in}, V_{out}, E)$ by choosing the rules and edges in this minimum loop. Let r be any rule in V_{in}, then there must be two rules r_a and r_b in V_{out} such that $r \xrightarrow{h_i} r_a$ and $r \xrightarrow{h_j} r_b$, for some h_i and h_j. In addition, h_i and h_j should match the same sequence of rules in prior to r. Otherwise, r will become a pivot rule, contradicting our assumption. By assigning flows h_i and h_j with opposite values, *e.g.*, $c_i = k$, $c_j = -k$, we can ensure $Value(r) = 0$. We perform the above process to ensure that $Value(r) = 0$ for each $r \in V_{in}$. Similarly, we can ensure $Value(r) = 0$ for each $r \in V_{out}$. Let $h_1, h_2, \ldots, h_k$ be the set of flows that are assigned with values (*e.g.*, $\pm k$) in the above process. Since for each r in the network, $Value(r)$ equals r's corresponding row of vector $\sum_i c_i h_i$, we have

$$c_1 h_1 + c_2 h_2 + \cdots + c_k h_k = 0 \ (c_1, c_2, \ldots, c_k \neq 0) \quad (14)$$

Thus, flows $h_1, h_2, \ldots, h_k$ are linearly dependent. Since we can always select $h_i^{'}$ as one of the flows, $h_i^{'}$ lies in the linear subspace generated by vectors in $\{h_1, h_2, \ldots, h_k\} \backslash \{h_i^{'}\}$, which is a subspace generated by $h_1, h_2, \ldots, h_n$.

(2) "⇒". By applying Lemma 1, Lemma 2, and Lemma 3, we can construct a sub-RBG $\mathcal{G} \subset \mathcal{G}_S^{H_{min}}$, where each rule has

a degree equal or larger than 2. Then, we can find a loop in $\mathcal{G}$ according to Lemma 4. Since the coefficient $c_i^{'}$ for $h_i^{'}$ is nonzero, then $h_i^{'} \in H_{min}$. Suppose the loop does contain two rules r_a and r_b, such that $r_a \xrightarrow{h_i^{'}} r_b$. Then according to (1), we can find a linearly dependent set which consists of flows $H_{min} \backslash \{h_i^{'}\}$. This contradicts the fact that H_{min} is a minimum linearly dependent set. ■

Theorem 2 is proven by combining Theorem 1 and Lemma 5.

C. Proof of Theorem 3

We prove the contrapositive of this theorem. In the baseline method without slicing, let $\mathcal{G}_S^{\tilde{H}}$ be the RBG of switch S with respect to $\tilde{H} \triangleq H \cup \{h^{'}\}$. According to our slicing-based method, the RBG of switch S using slicing is still $\mathcal{G}_S^{\tilde{H}}$. Thus, if the forwarding anomaly $FA(h_i, h_i^{'})$ is undetectable using slicing, we can find a loop in $\mathcal{G}_S^{\tilde{H}}$, or there is a pivot rule, according to Theorem 2. This means that the anomaly is undetectable without slicing, also according to Theorem 2.