SDN-RDCD: A Real-Time and Reliable Method for Detecting Compromised SDN Devices

Haifeng Zhou, Chunming Wu, Chengyu Yang, Pengfei Wang, Qi Yang, Zhouhao Lu, and Qiumei Cheng

***Abstract*— A software-defined network (SDN) is increasingly deployed in many practical settings, bringing new security risks, e.g., SDN controller and switch hijacking. In this paper, we propose a real-time method to detect compromised SDN devices in a reliable way. The proposed method aims at solving the detection problem of compromised SDN devices when both the controller and the switch are trustless, and it is complementary with existing detection methods. Our primary idea is to employ backup controllers to audit the handling information of network update events collected from the primary controller and its switches, and to detect compromised devices by recognizing inconsistent or unexpected handling behaviors among the primary controller, backup controllers, and switches. Following this idea, we first capture each network update request and its execution result in the primary controller, collect each received network update instruction and the information of any state update in switches, and deliver these four kinds of information to those backup controllers in an auditor role. An auditor controller is designed to create an audit record for each received network update request and to add its execution result of this network update request as well as the received four kinds of matching information to the audit record. In particular, heterogeneous auditor controllers are proposed to avoid the same vulnerability with the primary controller. The audit algorithm and theoretical proof of its effectiveness for security enhancement are then presented. Finally, based on our prototype implementation, our experimental results further validate the proposed method and its low costs.**

***Index Terms*— Software-defined network (SDN), controller hijacking, switch hijacking, SDN forensics, anomaly detection.**

I. Introduction

SOFTWARE-DEFINED network (SDN) [1]–[3] has been widely accepted by academia and industry communities in recent years. As a promising solution of network virtualization, SDN is increasingly deployed in a variety of practical settings. The primary innovation of SDN is the decoupling of the control and forwarding planes and the centralization of network control function, which enables network programmability, brings agile and flexible network management, and promotes network innovation. Nevertheless, this new architecture also brings potential security risks, some of which are never confronted in conventional networks, e.g., the vulnerability in SDN controllers, switches and controller applications. Such vulnerability is inevitable, and is possibly utilized by attackers to hijack controllers and switches. Potentially compromised SDN devices may monitor confidential communication, obstruct normal management and operation, and even damage the infrastructure. Consequently, it is critical to detect compromised SDN devices in time.

In fact, detecting compromised network devices is a long-standing challenge for conventional networks. The relevant research work [4]–[6] focuses on the misbehavior detection for conventional network devices, e.g., routers, switches, and hosts. However, the research work is not directly applicable for detecting compromised SDN devices due to new network architecture and devices. In particular, determining whether an SDN device is compromised or not is more complicated and challenging compared with making the same determination for a conventional network device, since various controller modules and applications are involved into the programming of the data plane and thus the behaviors of controllers or switches are not following only one or several given behavioral norms.

Currently, some research on SDN security has been conducted to mitigate hijacking threats: (1) authentication, access control, and accounting (abbreviated as AAA) for SDN controller applications [7]–[14], (2) Byzantine fault-tolerant SDN control plane [15]–[19], (3) SDN forensics [20]–[22], (4) SDN-based intrusion detection systems (IDSes) [23]–[26], and (5) SDN configuration verification [27]–[34]. Nevertheless, the first kind of research focuses only on protecting a controller from compromised or malicious controller applications, while the vulnerability in controllers and switches is not considered; the second kind of research devotes to improving the reliability of the control plane based on fault-tolerant techniques, whereas the security of the data plane and the evaluation of the potential performance degradation of the control plane are not covered at present; the third kind of research is still purely theoretical; the fourth kind of research is committed to bettering conventional intrusion detection in SDNs, while the detection of compromised SDN devices has not been taken into consideration. In particular, the current research on SDN configuration verification for detecting attacks (see [29], [30], [32], [33]) constructs and maintains the network state by capturing every network update, and

Manuscript received October 25, 2017; revised May 24, 2018; accepted June 24, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor X. Liu. Date of publication September 10, 2018; date of current version October 15, 2018. This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB0800201 and Grant 2016YFB0800102, in part by the Science and Technology Project of State Grid Corporation of China under Grant 52110118001F, in part by the Key Research and Development Program of Zhejiang Province under Grant 2017C01064, Grant 2017C01055, Grant 2018C01088, and Grant 2018C03052, in part by the Ministry of Industry and Information Technology of China for Testing, Solution Verification and Application Promotion of Industrial Information Physics System, and in part by the Fundamental Research Funds for the Central Universities under Grant 2016XZZX001-04. *(Corresponding author: Chunming Wu.)*

The authors are with the Cyber Security Research Institute and the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: zhouhaifeng@zju.edu.cn; wuchunming@zju.edu.cn; yangchengyu@zju.edu.cn; wangpengfei@zju.edu.cn; yangqi@zju.edu.cn; luzhouhao@zju.edu.cn; chengqiumei@zju.edu.cn).

Digital Object Identifier 10.1109/TNET.2018.2859483

incrementally checks for the compliance or correctness of new network updates (e.g., installation or removal of rules and link up or down) according to objective and preassigned security policies. However, these methods are based on a potential assumption that the controller is trustful, and thus the effectiveness of these methods is unable to be guaranteed when the controller hijacking happens. For instance, a compromised controller may conduct a malicious network update that is still consistent with security policies, since limited security policies are hard to or sometimes unable to prevent from all potentially malicious behaviors. As a result, detecting compromised SDN devices still remains a challenging task.

In this paper, we propose a real-time method termed as SDN-RDCD to detect compromised SDN devices in a reliable way. SDN-RDCD devotes to solving the detection problem of compromised SDN devices when both the controller and switch are trustless, and it is compatible and complementary with current detection methods, e.g., the aforementioned configuration verification methods. Our primary idea is to employ backup controllers to audit the online handling information of network update events collected from the primary controller and its switches, and to detect compromised SDN devices by identifying inconsistent or unexpected handling behaviors among the primary controller, backup controllers and switches. In particular, the backup controllers will rehandle each network update event of the primary controller, and the inconsistency of the handling results of the primary and backup controllers will be checked for detecting compromised controllers.

Motivated by this idea, we first propose a new controller role termed as auditor based on the current two controller roles (i.e., master and slave) defined by OpenFlow [1], and term a controller in this role as auditor controller. The auditor controller is selected from slave controllers (i.e., backup controllers), and is assigned to the audit task for its master controller (i.e., primary controller) and switches.

We then collect the handling information of network update events from the master controller and the switch, respectively. In the master controller, we capture every network update request (e.g., adding, modifying, deleting, and reading a flow table entry) from three respects, i.e., switch modules (e.g., `PACKET_IN` message generation modules), controller modules (e.g., routing and load-balancing modules) and applications (e.g., firewall applications). A legitimate network update request will be allocated with a unique audit ID for tracking its subsequent handling in controllers and switches, and it as well as its execution result in the master controller will be mirrored to each auditor controller. An auditor controller is designed to create an audit record for accommodating the received network update request and execution result according to the audit ID, and it will also re-execute the network update request and add its execution result to this audit record. In particular, heterogeneous auditor controllers are proposed to avoid having the same vulnerability with the master controller [35], [36]. For obtaining the same execution result of each network update event in master and auditor controllers, consistent network-related and application-related states are maintained among these controllers, and their differences in handling network update requests are also removed. In the switch, we deliver every network update instruction from the master controller and the information of any state update that is happened in the switch to each of its auditor controller. An auditor controller will add these information to their matching audit records.

By analyzing audit records and recognizing inconsistent or unexpected handling information, each auditor controller is competent to detect compromised SDN devices independently. The SDN-RDCD algorithm for SDN controllers is designed to perform the detection process. Further, we present the theoretical proof of its effectiveness for SDN security enhancement, introduce its system implementation, and analyze its overhead, scalability and advantages.

Finally, based on our prototype implementation on ONOS [37], [38] and OpenDaylight (ODL) [39] controllers, we perform an evaluation of SDN-RDCD by two experimental studies. In the first experimental study, we conduct a controller hijacking attack and a switch hijacking attack, respectively, and we evaluate the detection rate and performance of SDN-RDCD. In particular, these two kinds of attacks are hard to detect by current detection methods in a reliable way. The experimental results indicate that SDN-RDCD succeeds in detecting each hijacking attack and the compromised devices. In the next experimental study, we evaluate the overhead of SDN-RDCD including CPU, memory and bandwidth overhead. The experimental results indicate that SDN-RDCD exerts acceptable impact on the performance of the controllers and switches.

The rest of this paper is presented as follows. In Section II, we introduce the background about the detection problem of compromised SDN devices. The proposed detection method is then introduced in Section III. This is followed by an evaluation of the proposed method in Section IV. In Section VI, we conclude the paper.

A. Related Work

Detecting compromised network devices is a longstanding challenge in conventional networks. Haeberlen *et al*. [4] proposed an idea to create a per-node security log for recording messages that the network node has received and sent, and to authorize each peer node to request the log of another node for independently determining whether the node's behavior is deviated from behavioral norms. Based on the designed temper-evident log, a correct node is capable of defending itself against malicious accusations, and a compromised or faulty node will be detected by periodically performing the peer review process. Zhou *et al*. [5] proposed a mechanism termed as secure network provenance (SDP) to detect misbehaving or faulty nodes, and to assess the damage brought by these nodes. By utilizing the temper-evident log [4], operators are able to find out the truth when compromised nodes lie or frame innocent nodes, and SDP ensures that an observable symptom of a fault or an attack is able to be traced to a specific event and at least one compromised or faulty node. However, the above research work focuses on conventional network devices, e.g., routers, switches, and hosts, and is not directly applicable for detecting compromised SDN devices.

Kreutz *et al*. [15] first identified potential vulnerability in SDN controllers, switches and applications that might be utilized by attackers to launch controller and switch hijacking. To defend controllers against the threat from compromised or malicious controller applications, some research work on application access control, authentication and accounting has been conducted, e.g., Fortnox [7], OperationCheckpoint [10], LegoSDN [11], C-BAS [12], SE-Floodlight [13], and AuthFlow [14]. This kind of research work focuses on

the design of behavioral control and accounting methods for applications, aiming at preventing misbehaving applications from hijacking controllers and disturbing normal network operations. Whereas, the detection of compromised controllers or switches is not covered.

Kreutz *et al*. [15] investigated a solution of secure and dependable SDN control planes, and recognized the importance of controller diversity in the control plane for preventing Byzantine faults (a.k.a, arbitrary faults) since diverse controllers have less common-mode faults or intersecting vulnerability [35], [36], e.g., software bugs. Li *et al*. [16], [17] proposed to resist Byzantine faults in the control plane by utilizing Byzantine fault tolerant techniques [40], [41]. According to this research, a network update request needs to be co-conducted by multiple controllers in the control plane, and a switch needs to receive the same network update instruction from at least a given number of controllers before it carries out the instruction. Botelho *et al*. [18] proposed the design of a fault-tolerant SDN control plane, and proposed to store network-related and application-related states in a shared data store for smooth transition. Qi *et al*. [19] proposed a control plane architecture to accommodate multiple heterogeneous controllers for preventing controller hijacking, and conducted the theoretical analysis of its effectiveness while the system implementation is not presented. According to this research work, each network update request is handled in multiple heterogeneous controllers, and the controllers will vote and determine its final handling result. Nevertheless, the above methods only focus on the control plane rather than the whole SDN system (including the control and data planes), which makes these methods ineffective to prevent switch-related attacks and to detect the root cause of an anomaly involving both the control and data planes. Moreover, the performance degradation of the control and data planes brought by these Byzantine fault tolerant techniques is doubtless while it is not evaluated in the research work. In contrast, we consider the detection of compromised controllers and switches in a systematic way, and evaluate the performance impact brought by our method.

Al-Shaer and Al-Haj [27] proposed a method to check for conflicting rules among multiple SDN switches. Son *et al*. [28] proposed to model flow tables with the Yices SMT solver for detecting non-bypass property violations. Khurshid *et al*. [29] proposed a layer termed as VeriFlow between an SDN controller and its network devices to check for network-wide invariant violations in a real-time and dynamical way. Kazemian *et al*. [30] proposed a real-time policy checking tool called as NetPlumber based on Header Space Analysis (HSA) [31] to incrementally check for the compliance of state changes by leveraging a set of conceptual tools that maintain a dependency graph between rules. Dhawan *et al*. [33] proposed a method termed as SPHINX to detect the attack that violates the security policy of network topology and data plane forwarding. Further, SPHINX leverages the metadata gleaned from four kinds of OpenFlow messages, i.e., `FEATURES_REPLY`, `PACKET_IN`, `FLOW_MOD`, and `STATS_REPLY`, to incrementally build and update flow graphs for each traffic flow. It then performs topological state constraint verification according to learnt and administrator-specified constrains, and achieves forwarding state constraint verification based on flow consistency checks. However, the above research work is based on a potential assumption that the controller is trustful, and it is hard to be valid when the controller is compromised. In comparison to the work we aim at solving the detection problem of compromised SDN devices when both the controller and switch are trustless.

Matsumoto *et al*. [34] proposed a controller design to protect network availability from malicious administrators. Nevertheless, this research work only focuses on the fast recover of network availability after the malicious configuration happens. Chi *et al*. [32] proposed a mechanism to periodically perform sampling detection for compromised SDN switches by sending carefully designed test packets and checking for whether the under-detecting switches are correctly forwarding these packets. However, the under-detecting switches and their under-checking flow rules are randomly sampled, which makes this mechanism incapable of detecting potentially compromised switches in a reliable and timely way. In contrast, we consider the detection of both the compromised controller and switch in a reliable and timely way.

Bates *et al*. [6] proposed an SDN-based forensic system to investigate various faults in data center networks including previously unobservable attacks, e.g., data exfiltration, and collusion between compromised nodes. Nevertheless, this method is designed to deal with conventional network devices. Khan *et al*. [20] proposed an SDN forensic management architecture for monitoring and investigating security threats. This architecture consists of two modules, i.e., C-Watch, and S-Watch, for monitoring the controller and switch, respectively. However, it remains theoretical. Khan *et al*. [21] identified the significance of a forensic mechanism for SDN, and presented the potential locations for collecting possible evidence against attackers. Nevertheless, the research work only introduces the concept. We aim at proposing a practical method for detecting compromised SDN devices.

In addition, SDN-based intrusion detection and prevention systems, and monitor systems have been explored in [23]–[26], [42], and [43], whereas all the research work focuses on the detection or monitor for the data flow in the data plane without involving the control flow between controllers and switches.

B. Contributions

To conclude, the main contributions of this paper are as follows.

(1) We explore a detection method of compromised SDN devices when both the controller and switch are trustless, and consider the detection of compromised controllers and switches in a systematic way.

(2) We propose to employ backup controllers to audit the handling information of network update events collected from the primary controller and its switches, and achieve the effective detection of SDN compromised devices with low overhead.

(3) We describe the proposed method based on current SDN techniques, present the corresponding algorithm for SDN controllers, and introduce the prototype implementation in detail.

(4) We present both theoretical proofs and experimental results to validate the proposed method.

II. Detection of Compromised SDN Devices

In this section, we introduce the SDN paradigm, the potential security threat brought by compromised SDN controllers

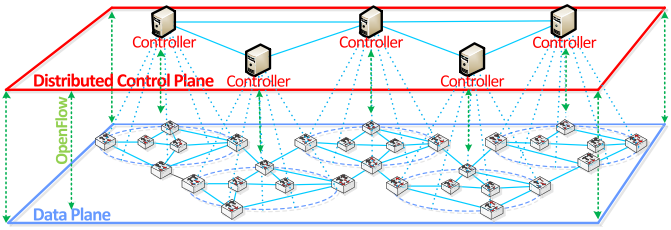


Fig. 1. An SDN scenario with multiple distributed controllers.

and switches, and the detection problem of these compromised devices.

A. SDN Paradigm

In SDNs, a controller manages its switches, and programs the forwarding behavior of them by configuring their flow table entries via a southbound protocol, e.g., OpenFlow [1]. Based on the northbound interface of the controller, controller applications, e.g., firewalls, monitors, and IDSes, are developed to achieve various network functions.

In particular, when a packet transmits from a host to its adjacent SDN switch, the switch will first search for a matching entry in the flow table according to the routing-related information contained in the packet, e.g., IP source and destination addresses, MAC source and destination addresses, transport source and destination ports, and VLAN ID [1]. If the switch fails to find a matching flow table entry, it will encapsulate the packet as a `PACKET_IN` message defined by OpenFlow, and will deliver the message to the controller. The controller will then extract the packet from the `PACKET_IN` message and acquire the routing-related information. The relevant controller modules (e.g., routing and load-balancing modules) and applications will deal with this event and determine whether to and how to deploy the transmission route for the packet and its following ones. Finally, the controller will instruct relevant switches to deploy required flow table entries for the data flow (i.e., the packet and its subsequent ones) via the southbound protocol.

To guarantee the scalability and reliability of SDN, distributed controllers are proposed [44]–[47]. Fig. 1 illustrates a scenario of distributed SDN controllers. In Fig. 1, the distributed controllers work as a cluster, and maintain a synchronized network view. Each of these controllers manages its assigned switches. A switch is managed by its primary controller (i.e., master controller defined by OpenFlow), and it also has several backup controllers (i.e., slave controllers). When the primary controller encounters a failure, one of the backup controllers will be selected as the new primary controller.

B. Security Threats From Compromised SDN Devices

The centralized control plane of SDN brings new security risks. Owing to the unavoidable software vulnerability, controllers have a potential risk of being compromised [15], and the whole network might be took over by attackers even without being perceived by operators. The potential security threats brought by controller hijacking are mainly concluded as follows.

- **Maliciously Damaging**. The attacker may command a compromised controller to disable its switches by deleting all their flow table entries, or to utilize its switches to launch DDoS attacks against servers, links, other controllers, switches, etc.
- **Resource Stealing**. The attacker may arbitrarily steal and utilize the network resources that belong to legitimate users by the compromised controller.
- **Monitoring and Manipulating**. The attacker may sneakily change the data flow between a pair of virtual machines (VMs) through a certain switch for monitoring and manipulating.
- **Further Hijacking**. The attacker may continue to hijack other controllers for controlling more switches until achieves the attack objective.
- **Network View Tamper**. The attacker may tamper the network view (e.g., topology) and flow table statistics (e.g., link utility), which will impact normal network functions.
- **Log Tamper**. The attacker may tamper the log of a compromised controller to prevent operators from detecting the hijacking attack, or to accuse innocent SDN devices for misguiding operators.

Meanwhile, the switch (especially software switch) also has a risk of being compromised [32]. Since the switch is the actual executant of controller instructions, compromised switches could bring similar six kinds of security threats. In particular, switch hijacking is competent to enhance the impact of controller hijacking attacks. For instance, with the assistance of a compromised switch, the aforementioned 3rd kind of security threat is capable of being conducted, and the misbehavior is hard to detect by tampering the logs of the controller and switch in a collaborative way.

C. Problem Definition and Motivation

Based on the above introduction, we now introduce the problem of detecting compromised SDN devices. Determining whether an SDN device is compromised or not depends on whether its behaviors are satisfied with the expected ones. In conventional networks, the determination is more simple. For instance, in many situations, determining whether a router is compromised or not only needs to determine whether its forwarding behaviors are deviated from its routing protocol, e.g., RIP, and OSPF. However, in SDNs, since varieties of controller modules and applications are involved into the programming of the data plane, the behaviors of controllers or switches are not following only one or several given behavioral norms, which makes the determination more complex and challenging.

To determine a controller is compromised, an effective and feasible method is to validate some of its network events are handled in an unexpected way. Such network events mainly includes (but not limited to): (1) network update events (e.g., establishing a transmission route for a data flow between two VMs), (2) network view establishment, and (3) flow table statistics acquisition. Network update events are mainly from three respects, i.e., the data plane (e.g., `PACKET_IN` message generation modules), the control plane (e.g., routing and load-balancing modules), and controller applications (e.g., firewall, monitor and IDS applications). A network update event generally involves all the necessary network update operations including adding a new flow table entry, modifying an existing flow table entry, deleting a specific flow table entry, reading flow table statistics, sending a packet from a specified switch port, turning down (or turning off) a specified switch port, etc. In fact, any network event, including network view establishment and flow table statistics acquisition, is eventually decomposed into a series of network update events.

A network update event is started with a network update request. A legitimate network update request will first be executed by the controller. The controller will then deliver the corresponding network update instructions to relevant switches for practical deployment according to the execution result. Consequently, by comparing the execution result of each network update request with the expected result, we are capable of determining whether the controller is compromised or not. The challenge is how to acquire the expected result for comparing, and we will introduce how to overcome it in the next section. Meanwhile, we are competent to determine a switch is compromised by detecting that some of its state updates are inconsistent with the corresponding network update instructions from the controller.

III. The Proposed Method: SDN-RDCD

In this section, we first introduce SDN-RDCD's threat model, principles and algorithm. We then present the theoretical proof of its effectiveness for security enhancement. This is followed by the introduction of its system implementation. Finally, we analyze its overhead, scalability, advantages and complementary relationship with existing detection methods, respectively.

A. Threat Model

We in this paper do not assume the controller or switch is trustful. As a result, both the controller and switch have a risk of being compromised. To extend the applicable range of our research work as wide as possible, we assume that an attacker may have compromised an unknown set of controllers and switches and he has complete control over these compromised SDN devices. Non-malicious problems (e.g., simple misconfigurations and hardware faults) are thus covered as a special case. Further, we conservatively consider: the attacker is able to instruct the compromised devices and the normal switches of the compromised controllers to arbitrarily change their states, and the attacker is even competent to disable the primary system and any security method (including our proposed method) by reading, forging, tampering with or destroying any information in the compromised devices. However, to keep our proposed method effective, we need to assume that at least one of the switches involved in a hijacking attack and at least one of the relevant auditor controllers are honest. Based on this assumption, at least an anomaly will be detected by more than one auditor controller even when compromised controllers and switches deceive operators collaboratively.

B. Principles

According to our analysis, to detect potentially compromised SDN controllers or switches in a reliable way, it is essential to collect the information on each network update event and the corresponding handling information of the master controller and switches, and to audit the collected information for recognizing inconsistent or unexpected handling information, e.g., inconsistent information between a network update instruction from the master controller and the corresponding state update in a switch. Specifically, the information that needs to be collected includes: (1) every network update request, (2) the execution result of each network update request in the master controller, (3) every network update instruction received by switches, and (4) each state update in switches. To collect the four kinds of information, a unique ID (termed as audit ID) is stamped on a legitimate network update request for tracking its following handling in controllers and switches. The audit ID is in favor of classifying the collected information and speeding up the audit process. Moreover, it is connected with the initiator of this network update request for enabling the root cause analysis of anomalies. The detailed introduction of the audit ID is presented in Section III.E.

Nevertheless, since both the controller and switch are not trustful, it is still unable to determine whether the network update request is correctly handled by the master controller and switches based on the above four kinds of information. According to our introduction in Section II.C, since the behavioral norm of the master controller and switches is not unique in SDNs, it is challenging to make such a determination.

Aiming at overcoming this challenge, we select and employ one or several slave controllers (i.e., backup controllers) to perform the audit task of the collected information for its master controller and switches. More importantly, by maintaining consistent network-related and application-related states among the master and slave controllers, each of these slave controllers is able to have the same execution result of network update requests with the master controller's in normal condition. As a result, based on our assumption in the threat model, we are able to determine whether the master controller is compromised or not by comparing its execution result of each network update request with the slave controllers'. Moreover, we are also competent to detect compromised switches when the network update instruction from the master controller is inconsistent with its practical implementation in these switches.

Further, to achieve a flexible selection and transition of slave controllers for the audit task, we propose a new controller role termed as auditor based on the current two controller roles (i.e., master and slave), and term a controller in this role as auditor controller. In particular, to avoid the same vulnerability of the master controller, heterogeneous controllers are more considerable to be auditor controllers.

To collect the handling information of network update events in the master controller, the master controller is designed to mirror every received legitimate network update request and the corresponding execution result to each of its auditor controllers. The auditor controller will create an audit record for each received legitimate network update request according to the audit ID, and will re-execute each network update request. Moreover, an auditor controller will add the received execution result of a certain network update request and its own execution result of this request to a matching audit record according to the audit ID. If the two kinds of execution results are available in an audit record, the auditor controller is capable of determining whether the execution result of the master controller is consistent with its own.

Meanwhile, to collect the handling information of network update events in the switch, the switch is designed to mirror received network update instructions (from the master controller) to each of its auditor controllers. Moreover, when a state update happens in a switch, the switch will capture this state update and inform each of its auditor controllers along with the state update information including the audit ID (if have) and state update content. In particular, the switch is also designed to deliver `PACKET_IN` messages to its auditor controllers as well as its master controller. This design is competent to fasten the audit process, since the auditor controller

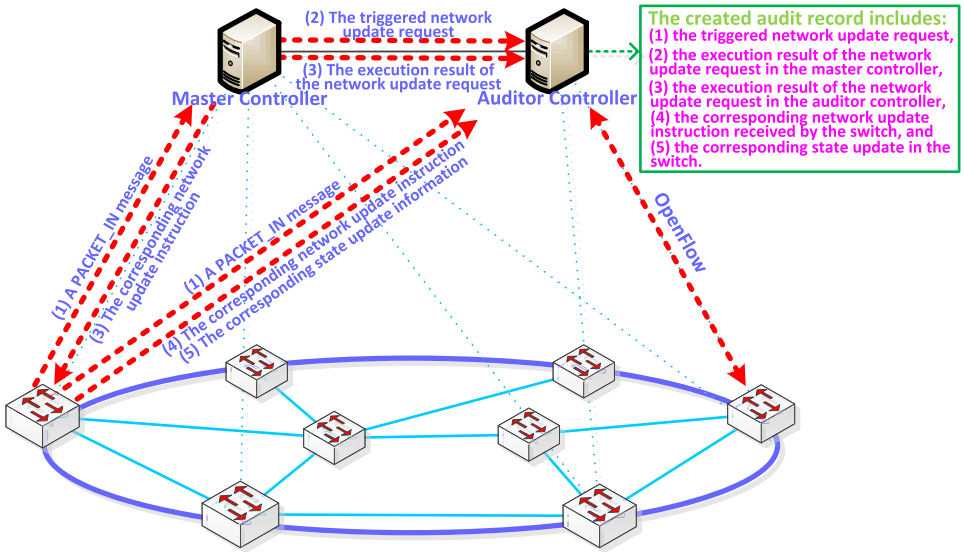


Fig. 2. SDN-RDCD's sketch map.

is capable of obtaining corresponding network update requests directly from switches rather than receiving them from the master controller.

According to the above introduction, each audit record created by an auditor controller will normally include five kinds of information: (1) the network update request, (2) the execution result of the network update request in the master controller, denoted by R_M, (3) the execution result of the network update request in the auditor controller, denoted by R_A, (4) the corresponding network update instructions received by relevant switches, denoted by I_S, and (5) the corresponding state updates in relevant switches, denoted by R_S. Based on the five kinds of information, an auditor controller is capable of making its own determination on whether the master controller or some relevant switch is compromised, and will inform operators when any compromised device is detected. The five main audit results in an auditor controller are concluded as follows.

1) $R_M = I_S = R_S = R_A$. This means the corresponding four kinds of information are consistent and the network update request was correctly handled by the master controller and relevant switches.
2) $R_M = I_S = R_S \neq R_A$. This means the execution results of the network update request in the master and auditor controllers are inconsistent and the master controller might be compromised.
3) $R_M = I_S \neq R_S$. This means some network update instructions were incorrectly performed in some switches and those switches might be compromised. (For instance, the master controller instructs some of its switches to deploy a specified transmission route for a data flow, while those switches deploy another route.)
4) $R_M \neq I_S$. This means some network update instructions were possibly manipulated during the transmission from the master controller to some switches and man-in-the-middle attacks might be happened.
5) Unmatched switch state update. This means a state update in some switch fails to match any audit record and the switch is thus suspected to surreptitiously perform a state update without the network update instruction from the master controller.

Fig. 2 illustrates a sketch of the proposed method (i.e., SDN-RDCD). Without loss of generality, Fig. 2 shows the sequence of messages for dealing with a network update event triggered by a `PACKET_IN` message, and this network update event only involves one switch for the sake of clarity. Further, Fig. 3 illustrates the procedures of an auditor controller for performing the audit task. As shown in Fig. 3, when receiving a kind of audit information, the auditor controller will first determine its kind. If it is a legitimate network update request delivered from the master controller, the auditor controller will then create an audit record for it according to the audit ID, re-execute it and add the execution result to this audit record. If it is one of the rest kinds of audit information, the auditor controller will then add it to a matching audit record according to the audit ID and determine whether the relevant handling information is consistent in real-time.

C. SDN-RDCD Algorithm for Controllers

Based on the procedures as shown in Fig.3, the SDN-RDCD algorithm (i.e., Alg. 1) for controllers is designed. In Alg. 1, C_M denotes the master controller, C_A is the set of auditor controllers, C is the controller running the SDN-RDCD algorithm, $Request$ denotes the set of network update requests, $Request_i$ ($i = 1, 2, \ldots, N$) is the ith network update request, ID_i is the audit ID of $Request_i$, $Record_i$ is the audit record of $Request_i$ in an auditor controller, S is the set of all the switches managed by C_M, S_i is the set of the switches involved in $Request_i$, and s_{ik} is the kth switch in S_i. R_{iM}, R_{iA}, I_{iS}, and R_{iS} are the execution result of $Request_i$ in C_M, the execution result of $Request_i$ in the auditor controller, the network update instructions of $Request_i$ received by S_i, and the state update for $Request_i$ in S_i, respectively. l_{ik} ($l_{ik} \in I_{iS}$) is the network update instruction for $Request_i$ received by s_{ik}, and r_{ik} ($r_{ik} \in R_{iS}$) is the state update for $Request_i$ in s_{ik}.

D. Effectiveness Analysis

We in this subsection analyze the effectiveness of the SDN-RDCD algorithm for enhancing security. The objective of this algorithm is to detect potentially compromised SDN

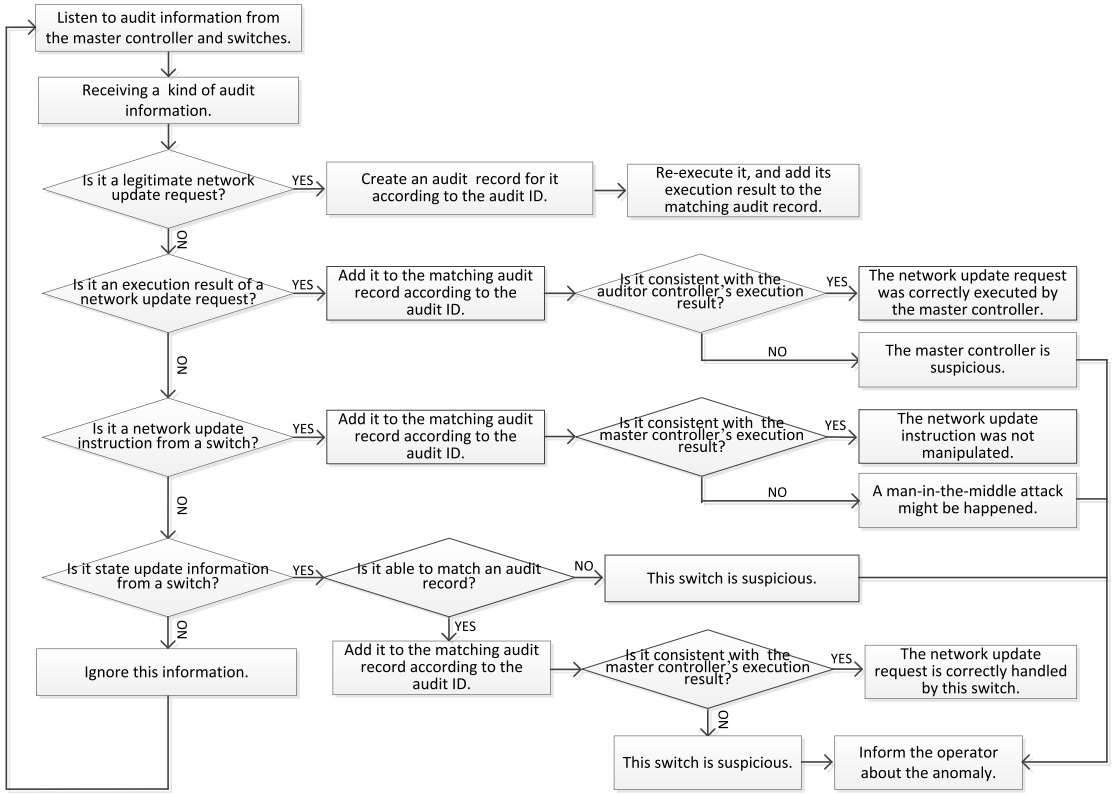


Fig. 3. Audit procedures in an auditor controller.

controllers and switches by recognizing inconsistent or unexpected behaviors among these devices. However, SDN-RDCD also has a risk of being perceived, attacked and compromised. Specifically, if every auditor controller is compromised or the audit information of every auditor controller from the master controller and switches is collaboratively manipulated, SDN-RDCD will be ineffective. Nevertheless, even if only one auditor controller recognizes inconsistent or unexpected behaviors of some devices, operators will identify an anomaly with suspicious devices (including auditor controllers), and will find out the source cause by analyzing their logs.

Based on the above analysis, we assume a worst case scenario: a compromised master controller or switch is capable of avoiding the detection of SDN-RDCD by removing or manipulating the audit information that is adverse to the attacker before the information is delivered to auditor controllers, and a compromised auditor controller is able to ignore adverse audit results. We are to conduct the theoretical proof of the effectiveness of the SDN-RDCD algorithm for SDN security enhancement in this worst case.

Specifically, we aim at proving its effectiveness in preventing a generic kind of hijacking attack on SDNs that needs to control one or several specified switches for achieving the attack intention, e.g., maliciously damaging, resource stealing, monitoring and manipulating (see Section II.B for detail). This kind of attack is capable of being achieved by directly compromising all the specified switches or by compromising their master controller. We now present the proof.

Proof. Let P' denote the success probability of such a generic kind of hijacking attack under SDN-RDCD, P the same success probability but without SDN-RDCD, M an event that the master controller is successfully compromised, S an event that all the specified switches are successfully compromised, A an event that every auditor controller is successfully compromised. We thus have the following two equations.

$$P = P(M \cup S) = P(M) + P(S) - P(M \cap S). \quad (1)$$

$$P' = P((M \cap A) \cup (M \cap S) \cup S) = P((M \cap A) \cup S) = P(M \cap A) + P(S) - P(M \cap A \cap S). \quad (2)$$

Equation (1) means this kind of attack is able to be launched by compromising the master controller or all the specified switches. Equation (2) means this kind of attack is able to be successfully conducted by compromising the master controller and every auditor controller at the same time or by compromising all the specified switches. By compromising the master controller and every auditor controller at the same time, operators will not perceive any anomaly caused by this kind of attack under SDN-RDCD. Besides, by compromising all the specified switches, these switches are competent to manipulate or remove the audit information that is adverse to the attacker before it is delivered to auditor controllers. In fact, the actual success probability of such a kind of attack under SDN-RDCD is less than P', since the attacker may fail to perceive the existence of SDN-RDCD and to successfully disable SDN-RDCD after it compromises controllers or switches.

Further, the security enhancement for resisting this kind of attack is able to be expressed by the following equation.

$$\triangle P = P - P' = P(M) - P(M \cap S) - P(M \cap A) + P(M \cap A \cap S). \quad (3)$$

Fig. 4 illustrates the part of $\triangle P$ clearly. Since $\triangle P$ is nonnegative, it means that the SDN-RDCD algorithm makes it less possible for attackers to conduct this kind of attack on SDNs.

Algorithm 1 SDN-RDCD Controller Algorithm for Detecting Compromised SDN Devices in Real-Time

A series of legitimate network update requests are successively initiated by a variety of entities.
if C is C_M **then**
  send $Request_i$ $(i = 1, 2, \ldots, N)$ to C_A
  execute $Request_i$ $(i = 1, 2, \ldots, N)$
  send R_{iM} $(i = 1, 2, \ldots, N)$ to C_A
else if $C \in C_A$ **then**
  if received $Request_i$ from C_M **then**
    create $Record_i$ using ID_i
    add $Request_i$ to $Record_i$
    re-execute $Request_i$
    add R_{iA} to $Record_i$
    if R_{iM} has been added to $Record_i$ **then**
      if $R_{iA} \neq R_{iM}$ **then**
        the master controller is suspicious
      end if
    end if
  else if received R_{iM} from C_M **then**
    add R_{iM} to $Record_i$
    if R_{iA} has been added to $Record_i$ **then**
      if $R_{iM} \neq R_{iA}$ **then**
        the master controller is suspicious
      end if
    end if
  else if received l_{ik} from s_{ik} **then**
    add l_{ik} to $Record_i$
    if l_{ik} is not consistent with R_{iM} **then**
      a man-in-the-middle attack might be happened
    end if
  else if received r_{ik} from s_{ik} **then**
    add r_{ik} to $Record_i$
    if r_{ik} is unable to match any audit record **then**
      s_{ik} is suspicious
    end if
    if r_{ik} is not consistent with R_{iM} **then**
      s_{ik} is suspicious
    end if
  end if
end if

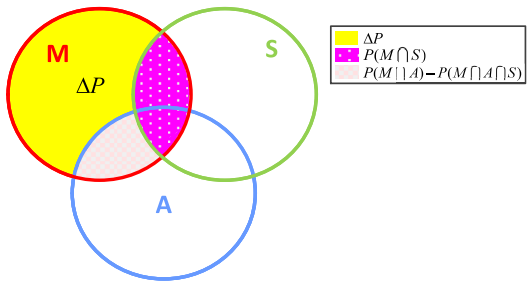


Fig. 4. $\triangle P$ illustration.

We thus have proved that the effectiveness of the SDN-RDCD algorithm for SDN security enhancement. In particular, if M is independent with S and A, respectively, we thus obtain the following simplified equation from Equation (3).

$$\triangle P = P - P' = P(M)(1 - P(S \cup A)). \tag{4}$$

Besides, we also present the proof that the SDN-RDCD algorithm is able to detect the anomaly caused by the above kind of attack under our assumption, i.e., at least one of the switches involved in a hijacking attack and at least one of the relevant auditor controllers are honest (see Section III.A for detail).

Proof: Let P_S denote the success probability of detecting an anomaly caused by the above kind of attack. We thus have the following equation.

$$\begin{aligned} P_S &= 1 - P(A \cup (M \cap S) \cup S) = 1 - P(A \cup S) \\ &= 1 - P(A) - P(S) + P(A \cap S), \end{aligned} \tag{5}$$

where $P(A \cup (M \cap S) \cup S)$ means the failure probability of detecting the anomaly. According to our assumption, we have $P(A) = P(S) = 0$. Associating with Equation (5), we thus have $P_S = 1$, which means that SDN-RDCD is sure to detect an anomaly caused by this kind of attack under our assumption.

E. System Implementation

1) Synchronization of Network-Related and Application-Related States: According to SDN-RDCD, it is essential to achieve the delivery of audit information between master and auditor controllers, and to maintain consistent network-related and application-related states among master and auditor controllers for keeping their consistent execution results of network update requests. In fact, current distributed controllers, e.g., ONOS [37], [38] and ODL [39], have implemented the synchronization of network view among master and slave controllers relying on distributed techniques, e.g., Paxos [48], Zookeeper [49], Hazelcast [50], Raft [51], and Infinispan [52]. As a result, when master and auditor controllers are the same kind of controller, the existing synchronization mechanism is able to be utilized. Otherwise, it is necessary to achieve one of those distributed techniques in each of heterogeneous master and auditor controllers for achieving the synchronization of their network-related and application-related states.

2) Switch State Update Traceability: According to SDN-RDCD, each legitimate network update request in the master controller is assigned with a unique audit ID, and the audit ID accompanies the corresponding network update request from its execution in the master controller to its final implementation in relevant switches. Moreover, an auditor controller will create an audit record for each network update request and categorize received audit information according to the audit ID. To achieve the delivery of the audit ID between the internal modules and functions of a controller or switch, a new audit ID parameter needs to be added. Further, to realize the delivery of the audit ID between controllers and switches, we accommodate the audit ID information by extending the OpenFlow protocol with a new audit ID field. In particular, the audit ID is connected with the initiator of the network update request, which is valuable to support the root cause analysis of anomalies and to detect malicious initiators.

3) Delivering Switch-Side Audit Information to Auditor Controllers: The switch-side audit information (i.e., network update instructions and switch state update information) needs to be delivered from switches to auditor controllers. We implement this delivery by utilizing and adjusting an existing message mechanism provided by OpenFlow, i.e., `FLOW_REMOVED` message mechanism. The `FLOW_REMOVED` message is sent by a switch, and this message is used to inform its controllers (including both master and slave controllers) that a specified flow table entry was removed in this switch. We made some appropriate adjustments to this message mechanism for achieving the delivery, and term the new one as AUDIT message mechanism. The original `FLOW_REMOVED` message mechanism is still remained at the same time. We utilize the AUDIT message to accommodate the network update instruction and the switch state update information, respectively. In particular, to achieve the delivery of switch-side audit information with low overhead, the AUDIT message is adjusted to deliver the audit information only to auditor controllers.

4) Capture of Switch State Update Information: According to SDN-RDCD, any state update that is happened in a switch needs to be captured, and its audit ID (if it has) and state update content need to be delivered to auditor controllers. To this end, each state update related function in a switch, e.g., the function of adding or deleting a flow table entry, the function of reading flow table statistics, and the function of turning on or turning off a specified switch port, needs to be adjusted to enable the capture and the collection of relevant state update information by equipping with corresponding codes.

F. Overhead Analysis

The overhead of SDN-RDCD mainly consists of three parts.

The first part is from the master controller since it needs to deliver the network update request and its execution result to auditor controllers. Because the traffic brought by the delivery is little, the overhead of this part is negligible.

The second part is caused by switches due to the fact that they need to deliver the network update instruction and the state update information to auditor controllers. Compared with the tremendous traffic in the data plane, the traffic brought by the delivery is rather small, and will produce a very limited adverse effect on the performance of switches. Moreover, this part of overhead is able to be further reduced in data centers by removing the mirroring process of network update instructions, considering the man-in-the-middle attack happens less in this kind of setting.

The last part is brought by auditor controllers, on account of executing network update requests and performing the audit task. Compared with the overhead of handling a network update request in the master controller, this part of overhead in an auditor controller is much less, since the auditor controller has no need to produce and deliver corresponding network update instructions to each of relevant switches by a southbound protocol. In particular, in a data center, considering its considerable computing resources and the limited number of auditor controllers, this kind of overhead is ignorable. Moreover, by selecting idle slave controllers as auditor controllers, the overhead of this kind is less important.

We will validate the above analysis results by experiment in Section IV.

Besides, maintaining consistent network-related and application-related states among the master and auditor controllers also brings overhead. However, this kind of overhead should not be completely attributed to SDN-RDCD, since the master and slave controllers originally need to maintain the consistency of these states. In particular, this kind of overhead brought by SDN-RDCD will be ignorable when SDN-RDCD is performed in those settings that have infrequent changes of network-related and application-related states.

G. Scalability

Distributed control plane has solved the scalability problem of SDN. In a distributed control plane, each controller manages a number of switches in its domain, and each controller generally owns one or several slave controllers for backup considering the reliability and restorability of the control plane. SDN-RDCD thus is able to achieve its scalability by selecting one or several slave controllers as auditor controllers in each domain, respectively.

Moreover, by allocating different parts of an audit task to different auditor controllers in a domain, the scalability problem of SDN-RDCD is competent to be further solved. For instance, employ each auditor controller to take charge of the audit task for those network update requests whose audit IDs are within a specified range of audit IDs.

H. Discussion

SDN-RDCD has three significant advantages. First, it facilitates effective reliable detection of compromised SDN controllers and switches online when both the controller and switch are trustless. Second, it exerts negligible impact on the performance of controllers and switches. In particular, the processing speed of the master controller and switches will not degrade, since the audit task is carried out in auditor controllers. Third, it enables an elastic and agile auditor controller selection and transition mechanism, and is able to effectively avoid common-mode vulnerability of the master controller by utilizing heterogeneous auditor controllers.

However, SDN-RDCD alone is not enough to overcome all potential security challenges. It needs the current detection methods [29], [30], [33] for determining whether a network update request is legitimate or not before the request is delivered to SDN-RDCD. Fortunately, it is really compatible with these methods. Compared with these representative detection methods [29], [30], [32], [33] SDN-RDCD is not based on an assumption that the controller or switch is trustful, and thus it is still able to be effective when both the controller and switch are trustless. Consequently, it could be used for overcoming the limitation of these detection methods and further enforcing the security.

IV. Evaluation

Based on our prototype implementation, we performed an evaluation of SDN-RDCD by two experimental studies. In the first experimental study, we evaluated the detection rate and the detection time of compromised controllers and switches, respectively. In particular, we conducted a kind of controller hijacking attack and a kind of switch hijacking attack, respectively, and validated SDN-RDCD's effectiveness in detecting these kinds of hijacking attacks that are hard to

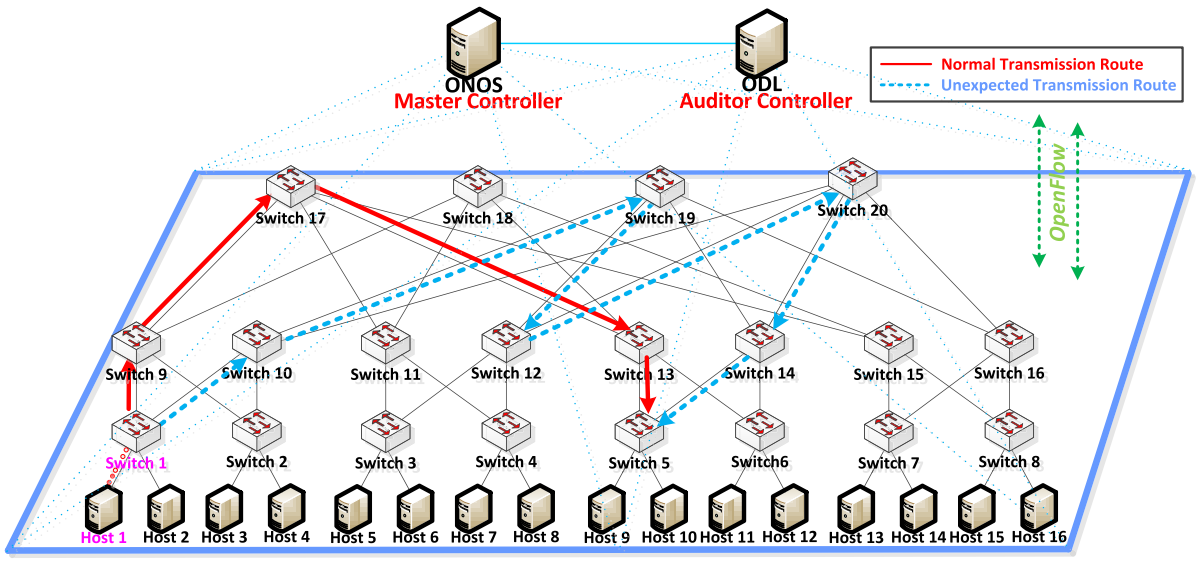


Fig. 5. Experimental topology.

detect by current detection methods. In the next experimental study, we evaluated the CPU and memory overhead caused by SDN-RDCD in the master controller, switch and auditor controller, respectively, and we also evaluated the bandwidth overhead caused by SDN-RDCD.

A. Experimental Setup

We used Mininet 2.0 [53] to generate a k = 4 3-layer fat-tree network of OpenFlow switches (Open vSwitch kernel switches, OVSes) and hosts, as shown in Fig. 5. The generated network ran on a server with a quad-core CPU (Intel Xeon 2.5GHz) and a total 16G memory. Besides, we chose ONOS [37], [38] and ODL [39] as the master and auditor controllers, respectively. The master controller (i.e., ONOS) ran on a server with a quad-core CPU (Intel Xeon 2.5GHz), and the auditor controller (i.e., ODL) ran on a similar server with a quad-core CPU (Intel Xeon 2.4GHz). Each of the servers has a total 16G memory. The system time of these three servers is synchronized by Network Time Protocol (NTP) [54]. The mean delays of the three links, i.e., the link between the first and second servers, the link between the first and third servers, and the link between the second and third servers, are 0.3018ms, 0.32275ms, and 0.328ms, respectively.

Fig. 5 illustrates the experimental topology. Each switch and host are marked with a corresponding number, respectively. All of these switches are managed by a master controller (i.e., the ONOS controller), and the master controller has one auditor controller (i.e., the ODL controller).

B. Evaluation Methodology

Without loss of generality, we in the experiment chose the `PACKET_IN` message (i.e., the most representative network update request) as the initiator of network update requests. In normal condition, when receiving a `PACKET_IN` message, the routing module of the master controller would compute a appropriate transmission route for the given pair of source and destination IPs, and would instruct the switches on that transmission route to add corresponding flow table entries to their flow tables for deploying the route.

Initially, Host 1 sent a packet with the destination of Host 9 to its adjacent switch, i.e., Switch 1. Switch 1 then delivered the packet to its master and auditor controllers by the `PACKET_IN` message mechanism (see Section III.B for detail). The packet of this kind was sent by Host 1 per 0.2s, and each of these packets had different source and destination ports. As a result, both the master and auditor controllers received 5 corresponding `PACKET_IN` messages per second from Switch 1. We call these `PACKET_IN` messages as special `PACKET_IN` messages. Meanwhile, normal `PACKET_IN` messages were produced 100 per second as the background `PACKET_IN` load of the master controller. The duration of the experiment is 300s, and thus the total number of the special `PACKET_IN` messages is 1500.

In particular, we ran this experiment two times, and before the start of each time, we simulated a master controller hijacking attack and a switch hijacking attack, respectively.

After the controller hijacking attack, the master controller was compromised, and its routing module was embedded with malicious codes so as to it would deploy an transmission route, e.g., the blue route in Fig. 5, rather than the normal red one for the above-mentioned special `PACKET_IN` messages. Owing to the demand of load-balancing, some routes like the red or blue one between Switch 1 and Switch 5 are all satisfied with security policies. As a result, current representative detection methods [29], [30], [32], [33] are hard to detect this kind of anomaly.

The switch hijacking attack compromised the switches on a special route, e.g., the blue route in Fig. 5, and these switches would sneakily deploy this route for those special `PACKET_IN` messages according to the instruction of the attacker rather than the instruction of the normal master controller. However, according to our threat model (see Section III.A for detail), the attacker failed to get rid of the detection from SDN-RDCD in one of these compromised switches on that route. As a result, one of the compromised switches delivered its true audit information to the auditor controller. This kind of anomaly is also hard to detect by current representative detection methods in a reliable way.

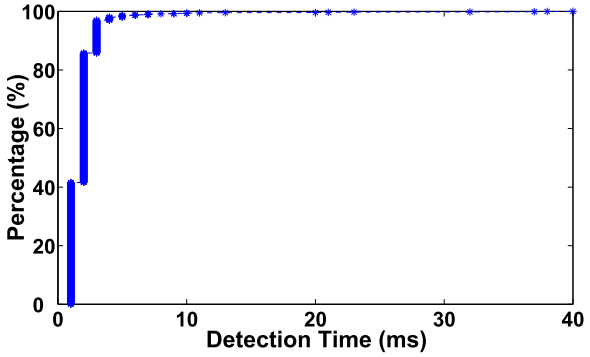


Fig. 6. Probability distribution of the detection time of the compromised master controller.

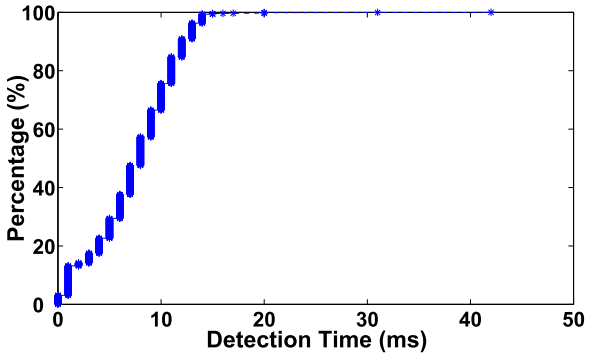


Fig. 7. Probability distribution of the detection time of the compromised Switch 1.

C. Effectiveness and Performance Evaluation

In the first experimental study, we evaluate the effectiveness and performance of SDN-RDCD. We first tested the detection rate and the detection time of the compromised master controller after it conducted misbehaviors (i.e., choosing an unexpected route for a special `PACKET_IN` message) during the first running of the experiment. We then tested the detection rate and the detection time of the compromised switches after they conducted misbehaviors (i.e., deploying an unexpected route for a special `PACKET_IN` message) during the second running of the experiment. We obtained 1500 data sets of the detection time.

According to our experimental results, we find each of the misbehaviors conducted by the compromised master controller or switches is successfully detected. Further, Figs. 6 and 7 show the probability distribution of the detection time of the compromised master controller and Switch 1, respectively. The time is accurate to 1ms resulting in the step curves in Figs. 6 and 7. The average detection time of the misbehaviors of the compromised master controller is 1.9260ms, and the average detection time of the misbehaviors of the compromised Switch 1 is 7.4867ms.

From these experimental results, we find the detection of compromised controllers and switches by SDN-RDCD is effective and fast.

D. Overhead Evaluation

In the second experimental study, we evaluate the CPU, memory and bandwidth overhead caused by SDN-RDCD. In the first running of the experiment, we tested the CPU and memory overhead of the three kinds of devices (i.e., the master controller, switches and auditor controller), respectively, and we also tested the bandwidth overhead of the links between them by Iperf [55]. Moreover, we also tested the same kinds of overhead under the same experimental scenario but without SDN-RDCD. In particular, we tested the above overhead per second during the running of the experiment, and obtained 300 data sets of each kind of overhead. Figs. 8-16 show the experimental results.

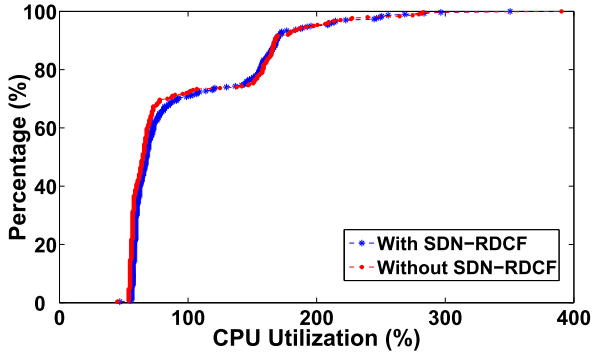


Fig. 8. Probability distribution of the CPU utilization of the master controller.

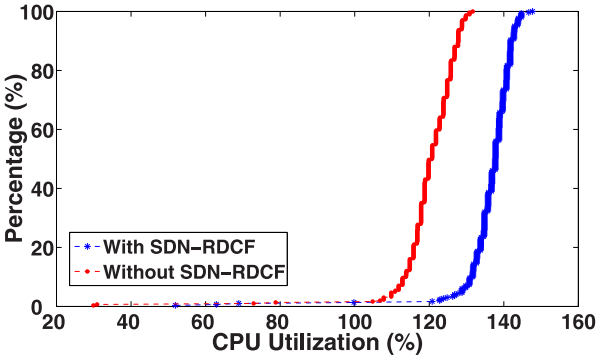


Fig. 9. Probability distribution of the CPU utilization of all the switches.

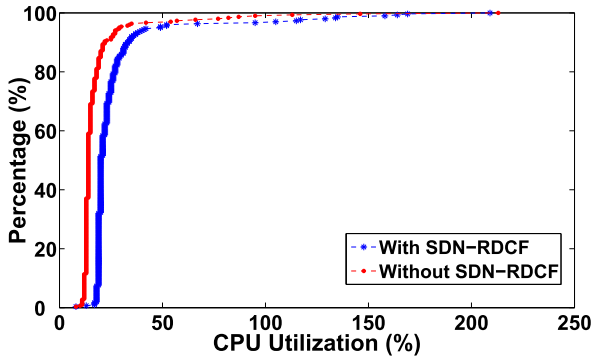


Fig. 10. Probability distribution of the CPU utilization of the auditor controller.

Figs. 8, 9 and 10 illustrate the probability distribution of the CPU utilization of the master controller, all the switches and the auditor controller, respectively. The shapes of the probability distribution curves in each of Figs. 8, 9 and 10 are similar, which indicates that the increased CPU overhead in these three kinds of devices is stable. According to the experimental results as shown in Figs. 8, 9 and 10, when removing SDN-RDCD, the measured average CPU utilization of these three kinds of devices is 93.6600%, 119.8280% and

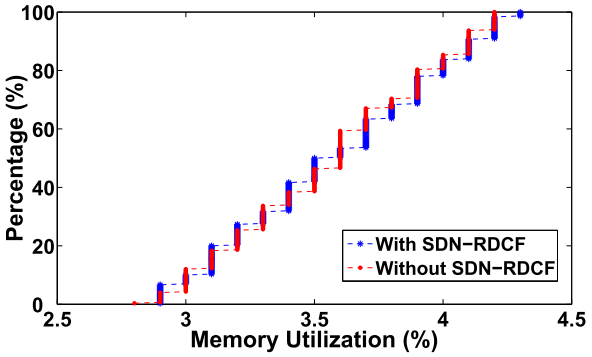


Fig. 11. Probability distribution of the memory utilization of the master controller.

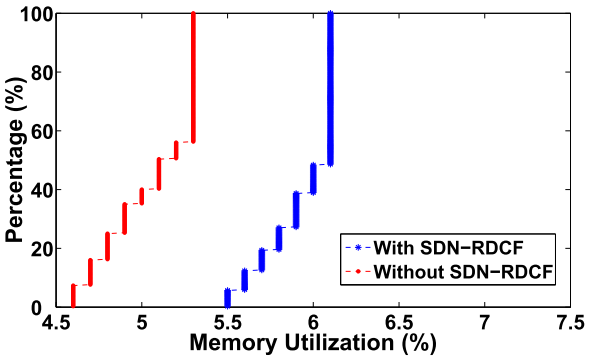


Fig. 13. Probability distribution of the memory utilization of the auditor controller.

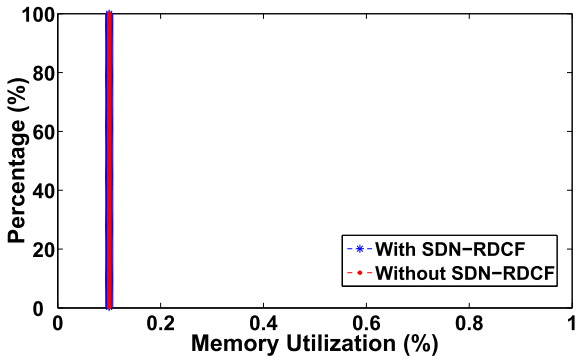


Fig. 12. Probability distribution of the memory utilization of all the switches.

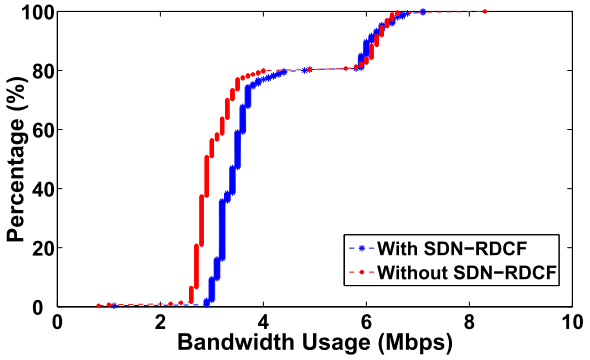


Fig. 14. Probability distribution of the bandwidth usage of the link between the master and auditor controllers.

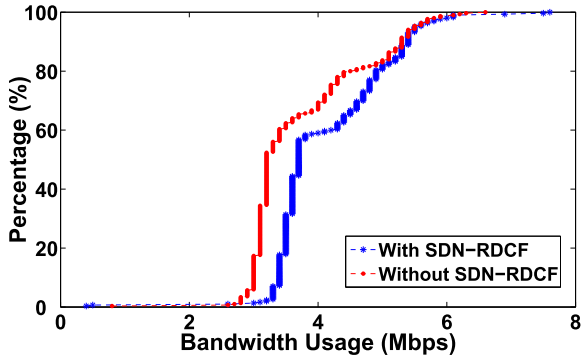


Fig. 15. Probability distribution of the bandwidth usage of the link between the master controller and the switch.

18.2533%, respectively, and when running SDN-RDCD, their average CPU utilization becomes 95.7707%, 136.3420% and 27.1167%, respectively. As a result, the average CPU overhead caused by SDN-RDCD in these three kinds of devices is 2.1107%, 16.5140% and 8.8633%, respectively. In particular, the increased switch CPU utilization, i.e., 16.5140%, is the sum of the increased CPU utilization of all the switches, and the CPU overhead in each switch is actually small. Consequently, the CPU overhead caused by SDN-RDCD in these devices is very limited.

Figs. 11, 12 and 13 show the probability distribution of the memory utilization of these three kinds of devices, respectively. The increased memory utilization of these three kinds of devices is also stable like the situation of CPU utilization. According to the experimental results as shown in Figs. 11, 12 and 13, when removing SDN-RDCD, the measured average memory utilization of these three kinds of devices is 3.5657%, 0.1000% and 5.0703%, respectively, and when running SDN-RDCD, their average memory utilization becomes 3.5770%, 0.1000% and 5.9487%, respectively. As a result, the average memory overhead caused by SDN-RDCD in these three kinds of devices is 0.0113%, 0% and 0.8783%, respectively, and the memory overhead is relatively small. Besides, the accuracy of the memory utilization data results in the step curves as shown in Figs. 11, 12 and 13.

Figs. 14, 15 and 16 show the probability distribution of the bandwidth usage of the three links, i.e., the link between the master and auditor controllers, the link between the master controller and the switch (i.e., the link between the second and first servers in the experiment), and the link between the switch and the auditor controller, respectively. According to the experimental results as shown in Figs. 14, 15 and 16, when removing SDN-RDCD, the measured average bandwidth usage of these three links is 3.6027Mbps, 3.7333Mbps and 4.6457Mbps, respectively and when running SDN-RDCD, their average bandwidth usage becomes 3.9350Mbps, 4.1300Mbps and 5.9593Mbps, respectively. As a result, the bandwidth overhead caused by SDN-RDCD in these three links is 0.3323Mbps, 0.3967Mbps and 1.3137Mbps, respectively, and the bandwidth overhead is rather small.

E. Evaluation Conclusion

According to the first experimental study, we validate that SDN-RDCD is able to detect compromised controllers and switches in a real-time and reliable way. Meanwhile, according to the second experimental study, we validate that the CPU,

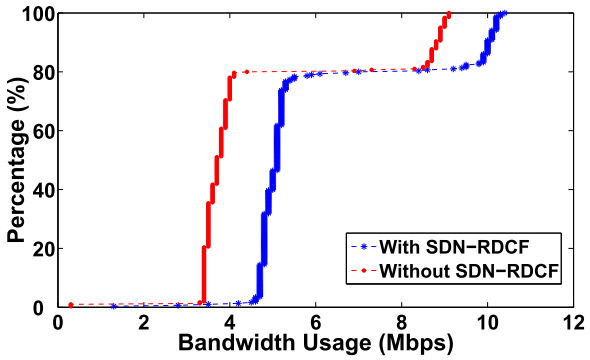


Fig. 16. Probability distribution of the bandwidth usage of the link between the switch and the auditor controller.

memory and bandwidth overhead caused by SDN-RDCD is low and exerts acceptable impact on the performance of the master controller, switch and auditor controller.

V. Conclusions

Real-time detection of compromised SDN devices is valuable for operators to protect network systems from further attacking and damaging in time. We in this paper proposed an online method to detect compromised SDN devices in a reliable way. This method enables backup controllers to audit the handling information of network update events collected from the master controller and switches, and to detect compromised SDN devices by recognizing inconsistent or unexpected handling behaviors among the master controller, backup controllers and switches. Controllers and switches are designed to support the collection of the audit information, and the detection algorithm of this method for controllers is presented. Further, we validated the proposed method both theoretically and experimentally.

References

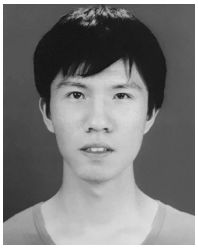
[1] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.
[2] T. Nadeau and K. Gray, *SDN–Software Defined Networks*. Newton, MA, USA: O'Reilly Media, 2013.
[3] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
[4] A. Haeberlen, P. Kouznetsov, and P. Druschel, "PeerReview: Practical accountability for distributed systems," *ACM SIGOPS Symp. Oper. Syst. Princ.*, vol. 41, no. 6, pp. 175–188, 2007.
[5] W. Zhou *et al.*, "Secure network provenance," in *Proc. 23rd ACM Symp. Oper. Syst. Princ.*, 2011, pp. 295–310.
[6] A. Bates *et al.*, "Let SDN be your eyes: Secure forensics in data center networks," in *Proc. NDSS Workshop Secur. Emerg. Netw. Technol.*, 2014, pp. 1–7.
[7] P. Porras *et al.*, "A security enforcement kernel for OpenFlow networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 121–126.
[8] D. Yu, A. W. Moore, C. Hall, and R. Anderson, "Authentication for resilience: The case of SDN," in *Proc. Cambridge Int. Workshop Secur. Protocols*, 2013, pp. 39–44.
[9] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for openflow applications," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 171–172.
[10] S. Scott-Hayward, C. Kane, and S. Sezer, "OperationCheckpoint: SDN application control," in *Proc. 22nd IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2014, pp. 618–623.
[11] B. Chandrasekaran and T. Benson, "Tolerating SDN application failures with Lego SDN," in *Proc. 13th ACM Workshop Hot Topics Netw.*, 2014, p. 22.
[12] U. Toseef, A. Zaalouk, T. Rothe, M. Broadbent, and K. Pentikousis, "C-BAS: Certificate-based AAA for SDN experimental facilities," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, Sep. 2014, pp. 91–96.
[13] P. A. Porras *et al.*, "Securing the software defined network control layer," in *Proc. NDSS*, 2015, pp. 1–15.
[14] D. M. F. Mattos and O. C. M. B. Duarte, "AuthFlow: Authentication and access control mechanism for software defined networking," *Ann. Telecommun.*, vol. 71, nos. 11–12, pp. 607–615, 2016.
[15] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 55–60.
[16] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 4, pp. 436–447, Oct. 2014.
[17] H. Li, P. Li, S. Guo, and S. Yu, "Byzantine-resilient secure software-defined networks with multiple controllers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 695–700.
[18] F. Botelho, A. Bessani, F. M. V. Ramos, and P. Ferreira, "On the design of practical fault-tolerant SDN controllers," in *Proc. 3rd Eur. Workshop Softw. Defined Netw. (EWSDN)*, Sep. 2014, pp. 73–78.
[19] C. Qi *et al.*, "An intensive security architecture with multi-controller for SDN," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 401–402.
[20] S. Khan *et al.*, "FML: A novel forensics management layer for software defined networks," in *Proc. 6th Int. Conf. Cloud Syst. Big Data Eng.*, Jan. 2016, pp. 619–623.
[21] S. Khan *et al.*, "Software-defined network forensics: Motivation, potential locations, requirements, and challenges," *IEEE Netw.*, vol. 30, no. 6, pp. 6–13, Nov./Dec. 2016.
[22] D. Spiekermann, J. Keller, and T. Eggendorfer, "Network forensic investigation in OpenFlow networks with ForCon," *Digit. Invest.*, vol. 20, pp. S66–S74, Mar. 2017.
[23] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar, "SnortFlow: A openflow-based intrusion prevention system in cloud environment," in *Proc. 2nd GENI Res. Educ. Exp. Workshop (GREE)*, Mar. 2013, pp. 89–92.
[24] T. Xing, Z. Xiong, D. Huang, and D. Medhi, "SDNIPS: Enabling software-defined networking based intrusion prevention system in clouds," in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2014, pp. 308–311.
[25] C. Jeong, T. Ha, J. Narantuya, H. Lim, and J. Kim, "Scalable network intrusion detection on virtual SDN environment," in *Proc. 3rd IEEE Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2014, pp. 264–265.
[26] M. A. Lopez, D. M. F. Mattos, and O. C. M. Duarte, "An elastic intrusion detection system for software networks," *Ann. Telecommun.*, vol. 71, nos. 11–12, pp. 595–605, 2016.
[27] E. Al-Shaer and S. Al-Haj, "FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures," in *Proc. 3rd ACM Workshop Assurable Usable Secur. Configuration*, 2010, pp. 37–44.
[28] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model checking invariant security properties in OpenFlow," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2013, pp. 1974–1979.
[29] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 467–472, 2012.
[30] P. Kazemian *et al.*, "Real time network policy checking using header space analysis," in *Proc. NSDI*, 2013, pp. 99–111.
[31] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proc. NSDI*, 2012, p. 9.
[32] 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. Softw. (NetSoft)*, Apr. 2015, pp. 1–6.
[33] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting security attacks in software-defined networks," in *Proc. NDSS*, 2015, pp. 1–15.
[34] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: Defending SDNs from malicious administrators," in *Proc. HotSDN*, 2014, pp. 103–108.
[35] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Analysis of operating system diversity for intrusion tolerance," *Softw., Pract. Exper.*, vol. 44, no. 6, pp. 735–770, 2014.
[36] S. Neti, A. Somayaji, and M. E. Locasto, "Software diversity: Security, entropy and game theory," in *Proc. 7th USENIX Conf. Hot Topics Secur.*, 2012, p. 5.

[37] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 1–6.
[38] *ONOS Introduction*. Accessed: Jun. 2, 2017. [Online]. Available: http://onosproject.org
[39] *OpenDaylight Introduction*. Accessed: Jun. 2, 2017. [Online]. Available: https://www.opendaylight.org
[40] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, 2002.
[41] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "FatTire: Declarative fault tolerance for software-defined networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 109–114.
[42] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou, "OrchSec: An orchestrator-based architecture for enhancing network-security using network monitoring and SDN control functions," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–9.
[43] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Comput. Netw.*, vol. 62, no. 5, pp. 122–136, 2014.
[44] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. OSDI*, vol. 10, 2010, pp. 351–364.
[45] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and flexible cellular core network architecture," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, 2013, pp. 163–174.
[46] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, p. 3.
[47] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 19–24.
[48] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: An engineering perspective," in *Proc. 26th ACM Symp. Princ. Distrib. Comput.*, 2007, pp. 398–407.
[49] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: wait-free coordination for Internet-scale systems," in *Proc. USENIX Annu. Tech. Conf.*, vol. 8, 2010, p. 11.
[50] *Hazelcast Introduction*. Accessed: Jun. 2, 2017. [Online]. Available: https://hazelcast.org
[51] *Raft Introduction*. Accessed: Jun. 2, 2017. [Online]. Available: http://infinispan.org/ https://raft.github.io
[52] *Infinispan Introduction*. Accessed: Sep. 15, 2016. [Online]. Available: http://infinispan.org
[53] *Mininet Introduction*. Accessed: Jun. 2, 2017. [Online]. Available: http://mininet.org
[54] *NTP Introduction*. Accessed: Jun. 2, 2017. [Online]. Available: https://www.opendaylight.org
[55] *IPERF Introduction*. Accessed: Jun. 2, 2017. [Online]. Available: http://www.ntp.org

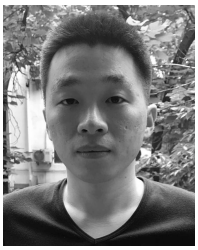
Haifeng Zhou received the Ph.D. degree in computer science and technology from Zhejiang University in 2018. His current research interests include software-defined network security, proactive network defense, intelligent networks and security systems, cloud security, software-defined networks, network traffic engineering, and innovative network and security technologies.

Chunming Wu received the Ph.D. degree in computer science from Zhejiang University in 1995. He is currently a Professor with the College of Computer Science and Technology, Zhejiang University. He is also the Associate Director of the Research Institute of Computer System Architecture and Network Security, Zhejiang University, and the Director of the NGNT Laboratory. His research fields include software-defined networks, reconfigurable networks, proactive network defense, network security, network virtualization, the architecture of next-generation Internet, and intelligent networks.

Chengyu Yang received the M.S. degree in computer science and technology from Zhejiang University in 2018. Her research interests include software-defined networks, software-defined network security, proactive network defense, and cloud security.



Pengfei Wang is currently pursuing the master's degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His current research interests include software-defined network security, proactive network defense, network security, and software-defined networks.



Qi Yang received the M.S. degree in computer science and technology from Zhejiang University in 2018. His research interests include software-defined networks, software-defined network security, cloud security, and proactive network defense.

Zhouhao Lu is currently pursuing the master's degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research interests include software-defined network security, software-defined networks, proactive network defense, intelligent networks and security systems, network security, and network traffic engineering.

Qiumei Cheng is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. Her current research interests include cloud security, software-defined network security, network virtualization, and intelligent networks and security systems.