Journal of Network and Computer Applications 68 (2016) 126–139

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Review

A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures

Wenjuan Li [a], Weizhi Meng [a,b,*,1], Lam For Kwok [a]

[a] Department of Computer Science, City University of Hong Kong, Hong Kong SAR, China
[b] Infocomm Security Department, Institute for Infocomm Research, Singapore

ARTICLE INFO

Article history:
Received 6 December 2015
Received in revised form
12 April 2016
Accepted 14 April 2016
Available online 19 April 2016

Keywords:
Software-Defined Networking
OpenFlow
Control and data planes
Security challenges and countermeasures
Review and survey

ABSTRACT

Software-Defined Networking (SDN) has been proposed as an emerging network architecture, which consists of decoupling the control planes and data planes of a network. Due to its openness and standardization, SDN enables researchers to design and implement new innovative network functions and protocols in a much easier and flexible way. In particular, OpenFlow is currently the most deployed SDN concept, which provides communication between the controller and the switches. However, the dynamism of programmable networks also brings potential new security challenges relating to various attacks such as scanning, spoofing attacks, denial-of-service (DoS) attacks and so on. In this survey, we aim to give particular attention to OpenFlow-based SDN and present an up-to-date view to existing security challenges and countermeasures in the literature. This effort attempts to simulate more research attention to these issues in future OpenFlow and& SDN development.

Contents

* Corresponding author at: Infocomm Security Department, Institute for Infocomm Research, Singapore.
E-mail addresses: wenjuan.li@my.cityu.edu.hk (W. Li), yuxin.meng@my.cityu.edu.hk, yuxin.meng@my.cityu.edu.hk (W. Meng), cslfkwok@cityu.edu.hk (L.F. Kwok).
[1] Previously known as Yuxin Meng.

http://dx.doi.org/10.1016/j.jnca.2016.04.011
1084-8045/

1. Introduction

With the development of computer networks, current network systems and data centers are becoming more and more feature-rich, complex and data excessive, so that the system designers often need to modify network software and orchestrate computer and network resources according to the specific requirements. However, traditional network architectures are ill-suited to fulfill the above requirements from enterprises, carriers, and end users. For instance, the decision-making capability of legacy networks is distributed across various network components, making a tedious job of adding any new network devices or services (Jammal et al., 2014). As a result, network management and configuration are becoming extremely laborious and error-prone.

In order to overcome these limitations, Software-Defined Networking (SDN) is developed by network research community, where network control is decoupled from the forwarding mechanism and can be directly programmable (Open Networking Foundation, 2012). That is, by decoupling the control logic from the individual network devices into accessible computing devices, SDN makes more of the network system components programmable and provides unified controls to the applications, such that researchers, system designers and administrators can design new network functions and protocols in a much easier and flexible way. A comparison between the legacy infrastructure and the SDN infrastructure is depicted in Fig. 1.

Generally, SDN relies on the fact that the simplest function of a switch is forwarding packets based on a set of rules. One objective of SDN is to perform network tasks, which cannot be completed without additional software, by providing open user-controlled management of the forwarding hardware of a network element. Another objective is to distribute part of the network complexity from the hardware-based network devices to the software-based controller (Lara et al., 2014). To summarize, the major merit of SDN is logically constructing a centralized controller through the separation of control plane and data plane, which enables the centralized control and simplifies network management.

To date, the most deployed SDN concept is OpenFlow, which is a communication protocol that gives access to the forwarding plane of a network switch or router over the network (Tootoonchian and Ganjali, 2010). This standard essentially opens up the Internet to researchers, allowing them to define data flows through software, giving engineers' access to flow tables, and designing rules to guide switches how to direct network traffic. It is also able to protect the proprietary routing instructions that differentiate one company's hardware from another (Greenk, 2009). Traditionally, when a data packet arrives at a switch, this switch mainly checks the packet's destination and forwards it according to the pre-defined rules. All packets going to the same place are routed along the same path and treated in the same way. By contrast, in an OpenFlow-based SDN, researchers can add to, subtract from, and otherwise meddle with these rules. Due to these advantages, OpenFlow-based applications become popular and have been studied in many areas such as VLAN (Yamasaki et al., 2011), wireless sensor networks (Luo et al., 2012), cellular networks (Li et al., 2012), mobile network (Pentikousis et al., 2013), Telecom domain (Hampel et al., 2013), intrusion detection (Mehdi et al., 2011) and so on.

Overall, SDN provides highly programmable switch infrastructures and computes optimal flow routing rules from remote users to virtually spawned computing resources (Porras et al., 2012). However, as the network expands both in the number of switches and the number of end hosts, the SDN controller may become a key bottleneck. For example, Tavakoli et al. (2009) estimated that a big data center which was composed of 2 million virtual machines may generate up to 20 million flows per second. Then, Benton et al. (2013) introduced some specific attacks for OpenFlow such as man-in-the-middle and denial of service risks. Some practical examples of how SDN can be both used and misused were described in Crenshaw (2012). Thanks to these efforts, security challenges and issues of SDN have gained considerable attention from both researchers and practitioners (Kreutz et al., 2013).

In this paper, we survey existing research efforts relating to both security challenges and promising solutions for OpenFlow-based SDN. In the literature, many SDN-related survey papers are available regarding SDN and its evolution (Farhady et al., 2015; Hakiri et al., 2014; Hu et al., 2014; Jammal et al., 2014; Jagadeesan and Krishnamachari, 2015), whilst only a few surveys focus on security issues (Alsmadi and Xu, 2015; Scott et al., 2013). Among them, Scott et al. (2013) in 2013 first discussed several security challenges in SDN without an analysis model. Additionally, they did not provide a discussion of the potential countermeasures. Then, Alsmadi and Xu (2015) conducted a survey on SDN security by means of STRODE threats model, which is the most related work. But they did not give a detailed analysis of OpenFlow security. Different from them, in this paper, we provide a new security analysis based on the CIA triad and focus on OpenFlow-based SDN, because OpenFlow is prominently successful and others are not as successful in practice (i.e., have not been adopted by the major router vendors) (Farhady et al., 2015). As a result, the analysis model and emphasis differentiate our work from others

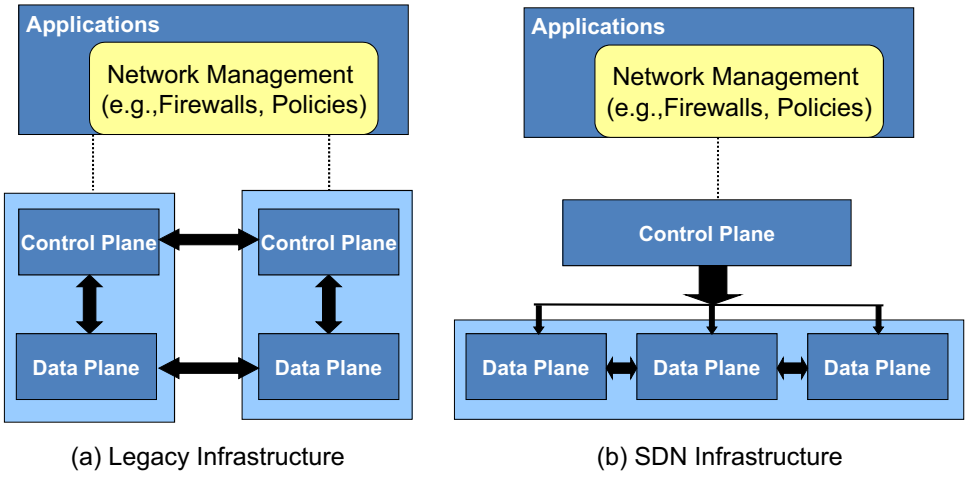

Fig. 1. A comparison between (a) the legacy infrastructure and (b) the SDN infrastructure.

Table 1
Main SDN interfaces (it is worth noting that east–westbound APIs are mainly used in distributed SDN architectures).

Terms	Description
Northbound APIs	These APIs offer the software interfaces between the application layer and the control layer, through providing universal data and functionality for network applications.
Southbound APIs	These APIs offer the software interfaces between the control layer and the infrastructure layer, and OpenFlow is the most popular southbound interface.
Eastbound APIs	These APIs are able to interconnect traditional IP networks with SDN networks, which often demand a translation module in-between.
Westbound APIs	These APIs are able to provide information between multiple SDN controllers in different domains, and enable management of distributed SDN architecture.

like (Alsmadi and Xu, 2015; Scott et al., 2013). Our work aims to complement the existing SDN security surveys and provide an up-to-date view in this area.

The remaining parts of this paper are organized as follows. Section 2 introduces the background of both SDN and OpenFlow. Then, Section 3 surveys security challenges and issues for OpenFlow-based SDN and Section 4 surveys existing promising countermeasures in defending against these security challenges and enhancing the robustness of SDN. Later, Section 5 discusses several future research directions in this area. Finally, Section 6 concludes this paper.

2. Background of SDN and OpenFlow

There is a misconception that SDN and OpenFlow are equivalent. In fact, SDN is an emerging network architecture in which the network control is decoupled from forwarding and is directly programmable, while OpenFlow is a foundational element for SDN and it is the first standard interface of communications defined between the control and forwarding layers of a Software-Defined Network architecture. Another possible reason for such misconception is that SDN was coined after the design of OpenFlow (Greenk, 2009). Thus, this section illustrates the architecture of SDN and OpenFlow to ensure a better understanding.

2.1. Software-Defined Networking

Software-Defined Networking (SDN) consists of a set of underlying programmable switches and a cluster of control entities, attempting to migrate as many network functionalities as possible into user-definable software. As shown in Fig. 1, the difference between SDN and traditional networks is that a software component running on a server or a CPU is added to the architecture of the network, where the software component is used as the control plane of the network. This centralized control server can enable dramatically simplified and flexible network programming. It can enhance the benefits of data center virtualization, increasing resource flexibility and utilization, and reducing infrastructure costs and overhead.

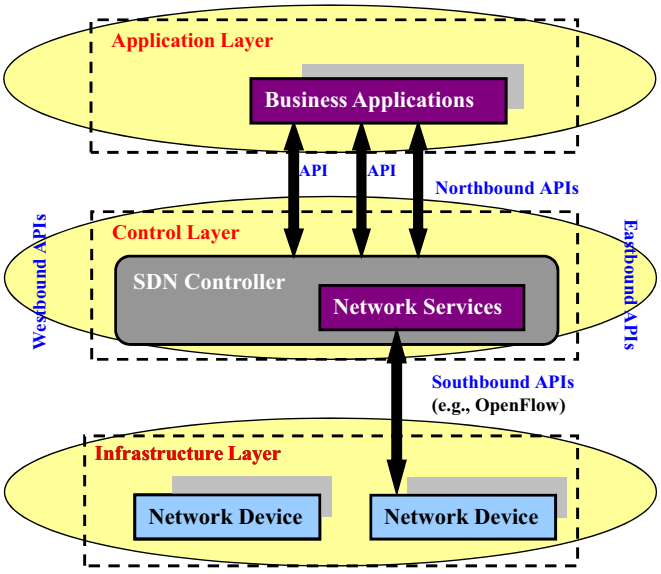

Fig. 2. The three-layer architecture of SDN.

More specifically, the three-layer architecture of SDN is depicted in Fig. 2, including application layer, control layer and infrastructure layer. Application layer can enforce its policies without a direct interaction with the infrastructure layer, through the northbound API that is supported by the control layer. On the other hand, the interactions between the control layer and infrastructure layer are supported by the southbound APIs. Logically, network management is centralized in software-based SDN controllers. Therefore, the network is able to act as a single and logical switch to both the applications and the policy engines. Under this architecture, the underlying physical network and the topology are hidden to users. Due to this design, enterprises can gain independent control over the entire network from a single logical point, which greatly simplifies the network design and operations.

By centralizing network states in the control layer, a system designer or network administrator in the application layer can react to network events in real-time and deploy new applications and services more quickly. This architecture also supports a set of APIs that make it possible to realize common network services, like routing and multicast, to meet individual and business objectives. Thus, with the implemented APIs between the SDN controller and the application layer, business applications can operate on an abstraction of the network in leveraging network services and capabilities without being tied to the details of their specific implementation (Open Networking Foundation, 2012). For instance, when a new flow reaches an SDN switch, this switch can send a route request to the centralized controller for the next forwarding path. The controller computes a routing path and distributes the forwarding rule to all relevant switches through a secure channel. As a result, all relevant switches can update their flow tables. The main SDN interfaces are summarized in Table 1.

Overall, SDN is able to manage the entire network through maintaining a global view and provide many benefits (e.g., on-demand resource allocation, secure cloud services, and virtualized networking). More specifically, one feature of the SDN is the capability to provide a wide abstraction of network. The abstraction enables SDN providing an easier way to configure a service or device by hiding the complexity of the network. The devices themselves can only accept instructions from the controllers without understanding and processing thousands of protocol standards. Another feature of the SDN is that it enables innovation and flexibility. The reason is obviously that the hardware-based devices are usually hard to modify while the software-based controller is easier to make a change and interaction.

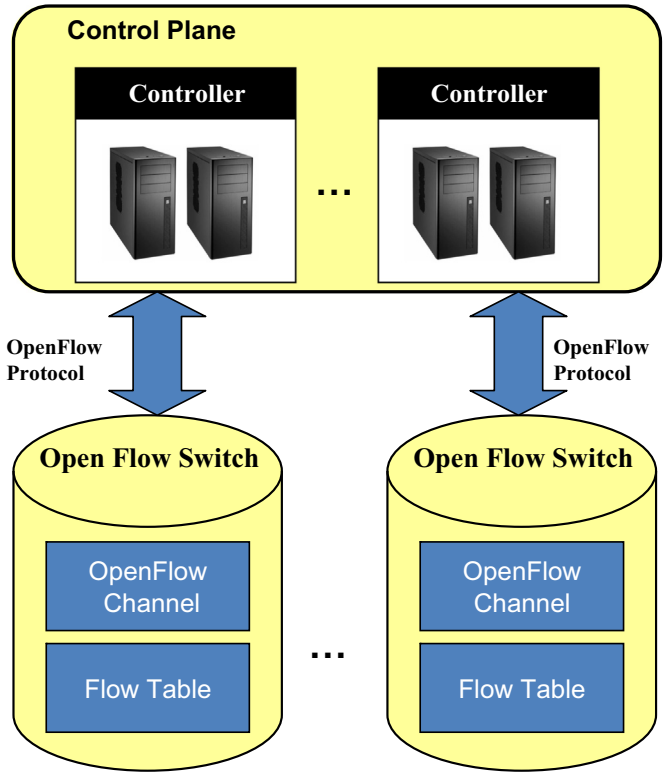

Fig. 3. Basic OpenFlow components and communications.

2.2. OpenFlow

OpenFlow is defined by the Open Networking Foundation (ONF) as the first standard interface of communications between the control and the infrastructure layer of an SDN architecture. It provides a means to control a switch without requiring the vendors to disclose any source code of their devices. In other words, OpenFlow allows direct access and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual (Open Networking Foundation, 2012). For instance, it provides access to the flow table and instructs switches how to direct network traffic. In this case, network managers can change network flows in a short period. According to Jammal et al. (2014), there are two major types of OpenFlow-based switches: OpenFlow-only and OpenFlow-hybrid. The former can only support OpenFlow operations, whereas the latter can support both OpenFlow operations and normal Ethernet switches.

As illustrated in Fig. 3, OpenFlow is mainly composed of three components: OpenFlow switch, OpenFlow channel and OpenFlow controller.

- *OpenFlow switch*. These switches are managed by OpenFlow controllers over a secure channel using the OpenFlow protocol. A switch often consists of one or more flow tables that performs packet lookup and forwarding. In particular, a flow table is composed of a list of flow entries while each entry contains header fields, counters and actions. *Header fields* are used to match against packets and contain information such as VLAN ID, source and destination ports, IP address and so on. *Counters* are mainly used to keep statistics about packets such as the number of packets, the number of bytes and so forth. *Actions* give instructions on how to process and match packets in a flow, such as forward to a given port, forward to a controller and drop the packet (Open Networking Foundation, 2012).
- *OpenFlow channel*. This channel acts an interface to connect OpenFlow switches and OpenFlow controllers. Through this interface, the controller configures and manages the switch, receives events from the switch, and sends packets out the switch. Three main types of messages can be sent through this channel: controller-to-switch, asynchronous and symmetric. *Controller-to-switch messages* are sent by the controllers to directly manage and inspect the state of the switch. *Asynchronous messages* are sent by the switch to update the controller about network events and changes to the switch state. *Symmetric messages* can be initiated by either the switch or the controller and sent without solicitation.
- *OpenFlow controller*. This centralized controller is responsible for maintaining, distributing and updating policies and instructions to the network devices. It can determine how to handle packets without valid flow entries, and can manage the switch flow table by adding or removing flow entries over the secure channel. In addition, an OpenFlow switch can establish connection and communication with one or multiple controllers. A multiple-controller architecture can improve network reliability when one switch fails to response. When OpenFlow operations start, the switch should connect to all its configured controllers at the same time, whereas relevant messages can only be sent to the corresponding switch. Table 2 summarizes a list of OpenFlow-based controllers.

To sum up, an OpenFlow switch might include multiple flow tables while each flow table can contain many flow entries. According to a flow table, the switch can look up its flow entries and make forwarding decisions for incoming packets. For each packet, the switch aims to find an exact matching entry. If a match is identified, the corresponding instructions can be executed. The packet is matched against the table and only the highest priority flow entry that matches the packet must be selected. If there are multiple matching flow entries with the same highest priority, the selected flow entry is explicitly undefined. On the other hand, if a packet does not match any flow entry in any flow tables (which is called “table-miss”), this packet can be sent to the controller or be dropped.

Table 2
A list of OpenFlow controllers written in C++, java and Python.

Controller	Description
NOX (Gude et al., 2008)	The first OpenFlow controller, which is written in C++.
SNAC (https://github.com/bigswitch/snac)	It is a graphical-interface-supported OpenFlow controller, which is written in C++.
RouteFlow (https://sites.google.com/site/routeflow/home)	It aims to provide virtualized IP routing services over OpenFlow enabled hardware (based on C++).
Maestro (http://zhengcai.github.io/maestro-platform/)	A scalable control platform written in Java which supports OpenFlow switches.
Beacon (https://openflow.stanford.edu/display/Beacon/Home)	It is a cross-platform and Java-based OpenFlow controller supporting both event-based and threaded operation.
Floodlight (http://www.projectfloodlight.org/floodlight/)	It is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller.
IRIS (https://github.com/bjlee72/IRIS)	It is the Openflow-based recursive SDN Openflow controller.
Jaxon (http://jaxon.onuos.org/)	It is an OpenFlow controller and works with NOX Classic.
POX (https://github.com/noxrepo/pox)	It is a networking software platform written in Python.
Ryu (https://github.com/osrg/ryu)	It is a component-based software defined networking framework.

From version 1.1, OpenFlow supports multiple flow tables and pipeline processing. Pipeline processing enables packets to be sent to subsequent tables for further processing and allows information to be communicated between tables. Table pipeline processing stops when the instruction set associated with a matching flow entry does not specify the next table; at this point the packet is usually modified and forwarded. To date, various OpenFlow versions have been released (i.e., started from version 1.0) (OpenFlow Switch Specification version 1.5.0, 2014).

3. Security challenges for openflow-based SDN

OpenFlow is a leading reference implementation of the SDN. However, by decoupling the control plane into a centralized application, the SDN controller itself is likely to become a key bottleneck. In addition, OpenFlow itself may introduce many potential security challenges to the network security community (Kloeti et al., 2013). In Fig. 4(a), we depict the potential attacking targets for OpenFlow-based SDN from the perspective of intruders.

In this section, we mainly focus on the identification of potential security challenges for OpenFlow-based SDN, through referring to existing research efforts in the literature. It is worth noting that current attacks on SDN have become more complicated, which can include attack vectors across OpenFlow controller, switch and channel. To keep simplicity, we introduce the security challenges based on the three major assets in a typical OpenFlow-based SDN: switch-related, controller-related and channel-related challenges.

Moreover, we will employ the CIA triad model to evaluate those challenges in terms of *confidentiality*, *integrity* and *availability* (Perrin et al., 2012). In this context, confidentiality is a set of rules that limits access to information, integrity is the assurance that the information is trustworthy and accurate, and availability is a guarantee of reliable access to the information by authorized people.

3.1. Switch-related security challenges

The switch-related security challenges mainly refer to vulnerabilities at the switch level (data plane), where attackers may utilize to compromise the SDN. OpenFlow switch and flow table are likely to become a major target, since they contain information related to network management, routing and access control. More specifically, attackers can target the network elements within the network itself. An attacker can first attempt to gain unauthorized physical or virtual access to the network, or compromise a host that is already connected to the SDN, and then try to perform attacks to destabilize the network elements.

For example, attackers may try adversely inserting some malicious flow rules to a flow table, causing the whole security mechanism to be compromised. In addition, through flooding the flow table with a large number of useless flow rules, the data plane might be unable to store normal flow rules and response to normal requests. One example of inserting malicious rules is described in Fig. 4(b). We discuss these security challenges according to the CIA triad as below.

- *Confidentiality.* At the switch level, *confidentiality* means to preventing sensitive information (e.g., switch data and flow table) from disclosure to unauthorized parties. It is noted that attackers may try various approaches to infer interested information such as scanning, spoofing, hijacking and so on.
- *Integrity.* At this level, *integrity* refers to protecting elements'

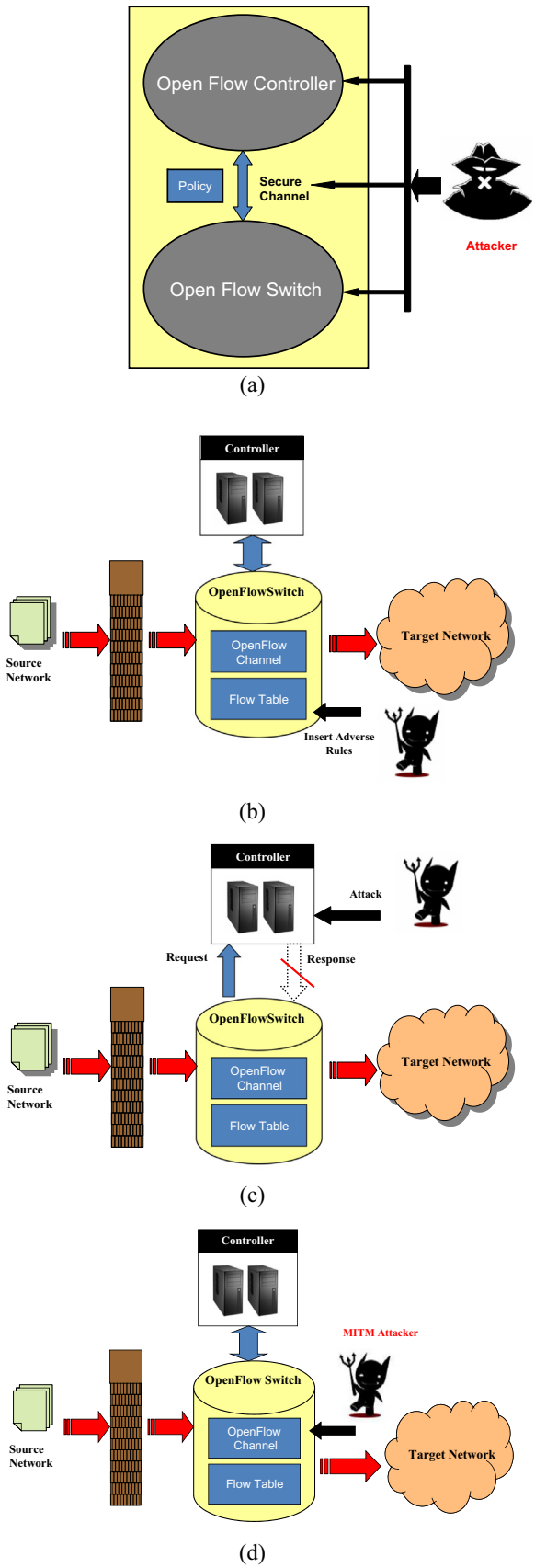

Fig. 4. (a) Potential attacks for OpenFlow-based SDN. (b) A case of security challenges at the switch level: inserting malicious rules. (c) A case of security challenges at the controller level: DoS attacks. (d) A case of security challenges at the channel level: man-in-the-middle attacks.

information from being modified by unauthorized parties, where elements here include switch itself, flow tables and hosts. OpenFlow switch allows us to update its flow table by adding or removing flow rules; however, control policy is usually not implementation-specified and enhanced in such network, which may open a hole for attackers.

- *Availability.* In the data plane, *availability* refers to ensuring that authorized parties are able to access the switch and related information when needed. As flow tables are an important element in an OpenFlow switch, which contain a list of flow rules to specify the operations, attackers may compromise it through various intrusions like denial-of-service (DoS) attacks.

At the switch level, attackers may compromise the above CIA aspects through malicious actions and inputs such as scanning, spoofing and so on.

1. *Scanning.* Scanning can be used by attackers as the first step to know the basic information about the whole network (e.g., topology) and prepare necessary knowledge for inferring sensitive information. Attackers can sniff network information such as topology, hosts' features and communication details between switches and hosts. It is often used by attackers as the first step to launch a large attack (e.g., DoS attacks).
2. *Spoofing.* This kind of attack refers to intruders successfully masquerade as another (e.g., switch or host), by falsifying data and thereby gaining an illegitimate advantage. At present, spoofing attacks in SDN mainly include Address Resolution Protocol (ARP) spoofing and IP spoofing. In particular, ARP spoofing aims to associate the attacker's MAC address with the IP address of a legitimate host (Feng et al., 2012). On the other hand, IP spoofing is the creation of Internet Protocol (IP) packets with a legitimate source IP address.
3. *Hijacking.* This type of attack aims to take control of a communication element, which is much stronger than spoofing attacks (as intruders can totally control an element rather than impersonation). In OpenFlow-based SDN, if attackers successfully hijack the switch, they can know all flow rules and communication data (Hong et al., 2015). On the other hand, if attackers hijack any host, they can impersonate a legitimate user and infer communication data from other hosts (e.g., tokens and passwords).
4. *Tampering.* This type of attack refers to unauthorized modification of network information (i.e., modifying flows in flow tables). As shown in Fig. 4(b), attackers can insert malicious flow rules that may cause network misbehavior. Hong et al. (2015) presented an attack of Fake LLDP Injection, in which an adversary generates fake LLDP packets into an OpenFlow network to announce bogus internal links between two switches. By monitoring the traffic from OpenFlow switches, the adversary can obtain the genuine LLDP packet.
5. *Denial-of-service (DoS) attack.* This type of attack attempts to make a network resource unavailable to its intended users, which includes flooding, Smurf and DNS amplification (Yao et al., 2011). Attackers may flood many legitimate flows to OpenFlow switch, causing flow tables unable to store other flows coming from other network elements. It is worth noting that such attacks can be conducted at the switch level or the flow table level. For example, Shin and Gu (2013) studied such kind of DoS attacks, named *data plane resource consumption*, by faking flow requests to cause lots of useless rules that need to be handled by a data plane. This type of attack is the major threat to availability, but it sometimes needs some supports from other intrusions such as scanning, spoofing, etc.
6. *Other attacks.* In addition to the typical attacks above, intruders can utilize relevant vulnerabilities to achieve the same goals. For example, attackers can take advantage of replay attacks to elevate privilege, which can spoof and hijack switches or devices.

3.2. Controller-related security challenges

The controller-related security challenges mainly refer to vulnerabilities at the controller level, in which attackers can utilize to compromise the CIA triad of SDN. The programmability of SDN controllers presents a double-edged sword. As SDN separates the control plane from the data plane to enable a centralized controller to deal with all incoming network flows, the controller itself is likely to become a key bottleneck for SDN and a major target for various attacks such as flooding attacks and denial-of-service (DoS) attacks.

For example, the data plane in the SDN has to ask the control plane to obtain flow rules when it receives unseen network packets (as it does not know how to manage). Thus, attacking the control plane can cause it to be unavailable to response to the requests from the data plane. An example can be illustrated in Fig. 4(c). We discuss these challenges based on CIA triad as follows.

- *Confidentiality.* At the controller level, *confidentiality* means to preventing sensitive information (e.g., controller data, rules and policies) from disclosure to unauthorized parties.
- *Integrity.* Regarding the controller, *integrity* refers to protecting controller's information from being modified by unauthorized parties, or controller's policy from being adversely tuned. OpenFlow controller should maintain many flow rules and firewall rules; however, attackers may orchestrate several rules to bypass the detection.
- *Availability.* At the control plane level, *availability* refers to ensuring that authorized parties are able to access the controller for requested information when needed. SDN controller presents a double-edged sword, because it is both a central point of influence in a network and a potential central point of failure. Thus, attackers may compromise SDN by conducting denial-of-service (DoS) attacks to the controller (Dover, 2013).

Similar to switches, at the controller level, attackers may conduct various malicious behaviors and intrusions such as scanning, spoofing, hijacking and so on.

1. *Scanning.* Similar to attacking switches, scanning can be used by attackers as the first step to understand the whole network (e.g., topology) and prepare for inferring sensitive information at the controller level. Attackers can further sniff network information through the controller.
2. *Spoofing.* If attackers successfully conduct this type of attack, they can impersonate as the controller. Then, attackers can create entries in the flow tables of network elements and the SDN engineers would not have visibility to those flows from the perspective of the production controller. In this case, attackers would have complete control of the network.
3. *Hijacking.* At the control plane, if attackers successfully hijack the controller, they can infer any sensitive information freely such as passwords, communication data and so forth. They can also tamper or modify any data and redirect traffic to any destinations. In this case, OpenFlow-based SDN will be completely compromised. For example, Hong et al. (2015) proposed an attack called host location hijacking attack, which can spoof the identity of a target host to hijack its location information inside OpenFlow controllers.
4. *Tampering.* Similar to tampering attacks at the data plane, the attacker would want to instantiate new flows at the controller by either spoofing northbound API messages or spoofing southbound messages toward the network devices. If an

attacker successfully tampers flows from the legitimate controller, then he/she would have the ability to allow traffic to flow across the SDN at their will and possibly bypass security policies.

A concrete example of tampering is *dynamic flow tunneling*: an attacker might try to orchestrate several rules, where no single flow violates any firewall rules but they can indeed violate firewall rules in a collaborative way. Porras et al. (2012) designed such an attack and indicated that OpenFlow controller can generate optimal flow routing rules from remote clients to virtually spawned computing resources. As the state of an OpenFlow switch should be reprogrammed to address the current flows, it is very difficult to predict the new policy. The main challenge is that as different control policies (or rules) may be inserted by various applications and users dynamically, it is hard to guarantee that these inserted policies or rules are not in conflict with each other.

An example of dynamic flow tunneling is given by Porras et al. (2012). Suppose there are three hosts:

(1) Source host: 10.0.0.2
(2) Target host1: 10.0.0.3
(3) Target host2: 10.0.0.4

For the firewall, assuming that there exists one rule: blocking network packets from the source host to the web service (port 80) running on the target host2, and that some OpenFlow applications aim to insert three new flow rules to the OpenFlow controller as below:

- Modifying the source IP address of a packet to 10.0.0.1, if this packet is sent from 10.0.0.2 to 10.0.0.3 (port 80).
- Modifying the destination IP address of a packet to 10.0.0.4, if this packet is sent from 10.0.0.1 to 10.0.0.3 (port 80).
- Forwarding a packet from 10.0.0.1 to 10.0.0.4 at port 80.

Thus, if the source host of 10.0.0.2 sends a packet to the target host1 at port 80, this packet can bypass the firewall as it is not directly sent to the target host2 but target host1. Specifically, we use a pair of {source IP, destination IP} to explain this case. Initially, this pair is {10.0.0.2, 10.0.0.3}, after arriving at OpenFlow controller, this pair will be modified as {10.0.0.1, 10.0.0.4}. Based on the last flow rule of forwarding, the packet from the source 10.0.0.2 can get to the target host2 of 10.0.0.4. This is an example showing that it is feasible to evade a firewall or an OpenFlow-based application by simply adding a few flow rules.

5. *Denial-of-service (DoS) attack.* An attacker might try to perform a DoS attack to the controller or use other means to cause the controller to fail. For example, attackers may try some forms of resource consumption attacks on the controller to bog it down, cause it to respond extremely slowly to incoming packets, and make it slow to send messages out. Shin and Gu (2013) demonstrated a feasible and effective DoS attack to SDN networks, which contains two steps. (1) To fingerprint whether a given network uses SDN/OF switches. The identification of a SDN switch is based on the observation that the response times for a receiving packet may be different since the flow setup time can be added in the case of new flow. (2) To generate crafted flow requests from the data plane to the control plane.
6. *Other attacks.* There are other attacks that can be used to compromise the confidentiality, integrity and availability of SDN such as replay attacks, privilege elevation and so on. Those attacks can utilize relevant vulnerabilities at the controller level to achieve the same purpose.

3.3. Channel-related security challenges

The channel-related security challenges mainly refer to vulnerabilities at the channel level, where attackers can utilize to compromise the SDN. Note that channel includes the communication between the components and the administrators.

One key challenge for OpenFlow-based SDN is that no appropriate trust mechanism between controllers and switches, which opens a hole for attackers to compromise the security by interception. Fig. 4(d) illustrates one of such challenges: man-in-the-middle (MITM) attacks, where an attacker secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other. We discuss these security challenges based on CIA triad as below.

- *Confidentiality.* For the OpenFlow channel, *confidentiality* means to providing a secure communication between controllers and switches, and defending against the disclosure of sensitive information to unauthorized parties.
- *Integrity.* At the channel level, *integrity* refers to ensuring that the transmitted information is consistent, accurate, trustworthy and non-repudiate.
- *Availability.* For the OpenFlow channel, *availability* refers to ensuring that authorized parties like controllers and switches are able to access each other when needed.

At the channel level, attackers may perform various attacks to infer sensitive information, such as man-in-the-middle attacks, monitoring, repudiation and so on.

1. *Man-in-the-middle (MITM) attacks.* Without robust trust mechanism, MITM attacks are feasible and effective. This type of attack is able to compromise all the CIA aspects. For example, attackers can impersonate the legitimate elements and send inaccurate information across the network. In addition, attackers can impersonate another party but do not response to any request, causing the paralysis of the entire network.
   In the literature, Lara et al. (2014) specified a situation that OpenFlow specification allows but not forcibly to use Transport Layer Security to secure the communication of the SDN networks. In this case, it is possible for the controller and the OpenFlow switches to use plaintext traffic. What is worse, even if the traffic is encrypted, the man-in-the-middle attack is still feasible between switches and controller (Benton et al., 2013). In this case, an attacker can intercept the traffic to compromise the security and privacy of an SDN network, whereas the network is hard to identify and response to it.
2. *Network monitoring.* Under this attack, intruders can try to monitor the OpenFlow channel and learn the communicated data between the switches and controllers. This type of attack can also be realized by MITM attacks. On the other hand, Shin et al. (2013) pointed out that OpenFlow offers very limited support for network monitoring, which makes it become very difficult for many security applications to learn and retrieve (critical) changes in network traffic patterns. In this case, these security applications cannot detect and identify an attack effectively and efficiently.
3. *Repudiation.* This situation means that an entity denies to be involved in a communication. As a result, non-repudiation appears so as to ensure such denial does not occur. This issue can also be caused by MITM attacks, through hijacking the channel between two parties and persuading that they are the other party.
4. *Other attacks.* Many other attacks like replaying attacks can facilitate the success of MITM attacks. The OpenFlow applications with possible vulnerabilities can be exploited to launch attacks.

For instance, Spoofing and hijacking attacks can be conducted to achieve this purpose.

3.4. Discussions

As compared with traditional networks, OpenFlow-based SDN is an emerging architecture, which separates the control plane from the forwarding plane to support virtualization. With the rapid development and increasing adoption, it has already become a big target for attackers. There are three major components in such network: switch, controller and channel.

At the switch level, there are numerous southbound APIs and protocols used for the controller to communicate with the network elements. Among them, OpenFlow is the most widely adopted protocol. Other protocols include Open vSwitch Database Management Protocol (OVSDB), Path Computation Element Communication Protocol (PCEP), Interface to the Routing System (I2RS) and so on. Each of these protocols has their own mechanisms of securing the communications between network elements. However, most protocols are very new and may leave a hole during the implementation (i.e., does not implement in a secure way). For example, an attacker would like to eavesdrop on flows to check what flows are in use and what traffic is being permitted across the network. Thus, he or she can try to eavesdrop on southbound communication between the network element and the controller. The sniffed information could be useful for a replay attack or simply for reconnaissance purposes.

At the controller level, various attacks are developed in practice. For example, controllers usually run on some Linux operating systems, where the vulnerabilities of that OS may become vulnerabilities for the controller. As another example, attacking the security of the northbound protocol can be a vector. There are many northbound APIs but most of them are not standardized. If attackers could leverage the vulnerable northbound API, then the attacker would have full control over the OpenFlow controller network through hijacking, man-in-the-middle attacks, etc. In this case, attackers are able to create their own SDN policies and thus gain control of the SDN environment. In addition, the OpenFlow controllers have to communicate with the high level applications, where attackers can use malicious applications to trick the controller (i.e., allowing malicious elements to join the network) (Scott, 2014).

For the OpenFlow channel, the key challenge is the lack of trust mechanism between the controllers and the switches. For instance, Transport Layer Security (TLS) is allowed but not enforced to secure the communication of the SDN networks. Even worse, many actual controllers are not even implementing or adopting it. It is worth noting that ARP poisoning will become feasible, if SSL encryption is not employed between the switches and the controllers.

In Table 3, we summarize those security challenges and related attacks in our discussions. Nevertheless, it is worth noting that with the development of OpenFlow-based SDN, attacks are often becoming more complicated through taking advantage of the vulnerabilities across different layers. These hybrid security challenges can largely threaten the security of such networks. For instance, hijacking and spoofing attacks can compromise more than one aspect of the CIA triad model through attacking OpenFlow switches and controller. It seems that this trend will continue and more advanced attacks could be designed by attackers in future to invade and compromise the OpenFlow-based SDN. Thus, effective countermeasures should be designed to provide protections in different layers.

Table 3
A summary of security challenges in terms of confidentiality, integrity and availability.

Our category	CIA triad	Attacks
Switch-related challenge	Confidentiality	Scanning, spoofing, hijacking, replaying attacks, etc.
	Integrity	Tampering, hijacking, replaying attacks, etc.
	Availability	Scanning, DoS, etc.
Controller-related challenge	Confidentiality	Scanning, spoofing, hijacking, replaying attacks, etc.
	Integrity	Tampering, hijacking, replaying attacks, etc.
	Availability	Scanning, DoS, etc.
Channel-related challenge	Confidentiality	MITM attacks, network monitoring, etc.
	Integrity	MITM attacks, repudiation, etc.
	Availability	MITM attacks, hijacking, etc.

4. Openflow-based SDN enhancement

In this section, we summarize and analyze existing research efforts in mitigating the aforementioned security challenges. These countermeasures are described according to OpenFlow switch, controller and channel, but it is worth noting that most current approaches in the literature can enhance more than one SDN layer by covering several vulnerabilities, with the purpose of defending increasingly advanced attacks.

4.1. Countermeasures for switch-related security challenges

As described earlier, OpenFlow switch is vulnerable to various attacks such as scanning, spoofing, hijacking, tampering, DoS and so on. In order to address these security challenges, some mechanisms can be made to enhance the security at the switch level such as authorization and policy enforcement. A list of existing efforts are given in Table 4.

Spoofing defend. Validating elements' address is very important to defend against spoofing attacks. Feng et al. (2011) proposed three extensions of OpenFlow about flow table, control mode and OpenFlow protocol and designed a commercial OpenFlow-enabled router, named OpenRouter, aiming to facilitate the deployment of OpenFlow in production network. Xiao et al. (2013) extended this idea and proposed an OpenFlow-based method of intra-AS source address validation named O-CPF, to filter out traffic with forged source IP, based on the idea of centralized computing forwarding path.

In addition, Mendonca et al. (2012) introduced AnonyFlow, an in-network anonymization service designed to efficiently and seamlessly provide privacy to users as they communicate with other endpoints and services. Jafarian et al. (2012) introduced a moving target technique called OpenFlow Random Host Mutation (OF-RHM) aiming to mutate IP addresses of end-hosts randomly and frequently, which can avoid being targeted by attackers. In addition, end hosts in OpenFlow-based SDN can be mapped to actual or physical IP addresses. Later, Bifulco and Karame (2014) introduced NPoL, an OpenFlow application which can be used by registered users to acquire secure location proofs from the network operator.

Tampering defend. This type of attack is mainly caused by the lack of policy enforcement. The dynamic and traffic-dependent modifications induced by middleboxes make it difficult to reason about the correctness of network-wide policy enforcement, so there is a need of flow tracking capability to ensure consistent policy enforcement in the presence of such dynamic traffic

Table 4
A list of existing solutions for switch-related security challenges.

Work in literature	Implementation	Year	Confidentiality	Integrity	Availability
Al-Shaer and Al-Haj (2010)	FlowChecker	2010	√	√	
Feng et al. (2011)	OpenRouter	2011	√	√	√
Khurshid et al. (2012)	VeriFlow	2012	√	√	
Mendonca et al. (2012)	AnonyFlow	2012	√	√	
Jafarian et al. (2012)	OF-RHM	2012	√	√	
Bifulco and Karame (2014)	NPoL	2014	√	√	
Lara and Ramamurthy (2014)	OpenSec	2014	√	√	√
Matias et al. (2014)	FlowNAC	2014	√	√	√
Wang et al. (2014)	OF-GUARD	2014	√	√	√
Kamisinski and Fung (2015)	FlowMon	2015	√	√	

modifications. Fayazbakhsh et al. (2013) argued there is a need of using flow tracking to ensure consistent policy enforcement in the presence of such dynamic traffic modifications. They then proposed FlowTags, an extended SDN architecture in which middleboxes add Tags to outgoing packets, aiming to provide the necessary causal context (e.g., source hosts or internal cache/miss state). These Tags are used on switches and (other) middleboxes for systematic policy enforcement.

Misconfiguration identification. OpenFlow makes it easier for researchers to run their own experiments by providing a virtual slice and configuration in real networks. Misconfiguration problems can arise when a user writes conflicting rules for single flow table or even within a path of multiple OpenFlow switches. FlowChecker (Al-Shaer and Al-Haj, 2010) was proposed to identify intra-switch misconfiguration within a single flow table. It encodes tables' configuration using Binary Decision Diagrams and then uses the model checker technique to model the inter-connected network of OpenFlow switches. Kamisinski and Fung (2015) focused on detection of compromised switches through real-time analysis of the periodically collected reports. Two types of malicious behavior of compromised switches were investigated such as packet dropping and packet swapping. They then proposed FlowMon, a malicious switch monitoring and detection system using the OpenFlow protocol. For their solution, the controller analyzes the collected port statistics and the actual forwarding paths to detect malicious switches in SDN.

The OpenFlow switch enables exciting new network functionality, at the risk of programming errors that make communication less reliable. To address this issue, Khurshid et al. (2012) designed VeriFlow, a separate layer between the SDN controller and network devices aiming to dynamically check for network-wide invariant violations when each forwarding rule is inserted. This tool can run in real-time, but a major weakness is that it requires a complete view of the entire network, which is very difficult to achieve in practice. Sonchack et al. (2015) presented OFX, an OpenFlow extension framework that can provide a better tradeoff between performance and deployability for SDN security applications by allowing them to dynamically install software modules onto network switches.

Policy enforcement. Lara and Ramamurthy (2014) proposed OpenSec, an OpenFlow-based security framework that allows a network security operator to create and implement security policies written in human-readable language. The user can describe a flow in terms of OpenFlow matching fields, define which security services must be applied to that flow (deep packet inspection, intrusion detection, spam detection, etc.) and specify security levels that define how OpenSec reacts if malicious traffic is detected. Then, Wang et al. (2014) introduced OF-GUARD, a scalable, efficient and lightweight framework for SDN networks to prevent data-to-control plane saturation attack by using packet migration and data plane cache. Packet migration detects flooding attacks and aims to protect switches and controller when an attack occurs. Data plane cache is a machine that stores proactive flow rules, caches table-miss packets and distinguishes fake packets from normal packets. In fact, this mechanism can protect both switch and controller levels according to specific requirements (i.e., they proposed FloodGuard that can protect the controller, Wang et al., 2015).

In addition, Matias et al. (2014) presented FlowNAC, a Flow-based Network Access Control solution that allows granting users the rights to access the network depending on the target service requested. It uses a modified version of IEEE 802.1X to authenticate users and service-level access control based on proactive deployment of flows. Under this context, SDN can add the appropriate granularity (fine- or coarse-grained) depending on the target scenario and dynamically identify the services at the data plane as a set of flows to enforce the adequate policy. Recently, Liu et al. (2016) designed a two-layer OpenFlow switch topology for implementing security policies, which considers the limitation of flow table size in a single switch and the complexity of configuring security policies to these switches. They also introduced a safe way to update the configuration of these switches one by one for better load balance when traffic distribution changes.

On the whole, flow checking, software attestation and trust management mechanisms are promising to handle those security challenges at the switch level.

4.2. Countermeasures for controller-related security challenges

To protect the SDN controller from various threats such as DoS attacks, trust models and validation mechanisms should be added to protect this key component. Several existing approaches are summarized in Table 5.

Spoofing defend. The validation of source addresses can be performed at the controller. For example, Yao et al. (2011) proposed a solution at the controller named Virtual source Address Validation Edge (VAVE), which can verify the address of external packets. Then, Matias et al. (2012) implemented an address resolution mapping (ARM) module in the controller, which can track MAC addresses from authorized users. In this case, controllers can discard ARP responses that are not validated by this module. Feng et al. (2012) demonstrated a solution of intra-AS IP source address validation with OpenRouter and showed OpenRouter is feasible to validate the source address of a packet. Hong et al. (2015) presented TopoGuard, a security extension to the OpenFlow controllers, which provides automatic and real-time detection of Network Topology Poisoning Attacks. The evaluation on the Floodlight controller showed that TopoGuard can effectively secure network topology while introducing a minor impact on normal operations of OpenFlow controller.

DoS migration. Regarding this issue, Tootoonchian and Ganjali (2010) presented HyperFlow, a distributed event-based control

Table 5
A list of existing solutions for controller-related security challenges.

Work in literature	Implementation	Year	Confidentiality	Integrity	Availability
Tootoonchian and Ganjali (2010)	HyperFlow	2010			√
Suh et al. (2010)	CONA	2010		√	√
Yao et al. (2011)	VAVE	2011	√	√	√
Porras et al. (2012)	FortNOX	2012	√	√	√
Canini et al. (2012)	NICE	2012	√	√	
Matias et al. (2012)	ARM	2012			√
Wen et al. (2013)	PermOF	2013	√	√	
Hu et al. (2014)	FlowGuard	2014	√	√	
Fichera et al. (2015)	OPERETTA	2015			√
Wang et al. (2015)	FloodGuard	2015			√
Hong et al. (2015)	TopoGuard	2015	√	√	√

plane for OpenFlow. By passively synchronizing network-wide views of OpenFlow controllers, HyperFlow localizes decision making to individual controllers, thus minimizing the control plane response time to data plane requests. In other words, it is logically centralized but physically distributed. It also enables interconnecting independently managed OpenFlow networks, an essential feature missing in current OpenFlow deployments.

Then, Suh et al. (2010) proposed a Content-oriented Networking Architecture (CONA) where the hosts request contents from its agents. In CONA, an access router is extended to identify what contents are requested, and its agent receives a content request from an attached host and delivers the requested content. In this way, CONA achieves the accountability and can take a countermeasure against resource-exhaustive attacks like DDoS. A prototype was built on the NetFPGA OpenFlow platform but its performance was not evaluated in large and practical experiments. Fichera et al. (2015) presented OPERETTA, an OpenFlow Remedy to TCP SYNFLOOD attacks, which relies on the use of an SDN approach to protect the OpenFlow controller from TCP SYNFLOOD attacks. It can be implemented in the controller that manages incoming TCP SYN packets and rejects fake connection requests.

Later, Wang et al. (2015) studied the data-to-control plane saturation attack in reactive controllers and developed FloodGuard, a lightweight and protocol-independent defense framework, to protect the controller from being overloaded. In particular, their approach consisted of two techniques: proactive flow rule analyzer and packet migration. The former combines symbolic execution and dynamic application tracking to derive proactive flow rules in runtime. The latter migrates, caches, and processes table-miss packets by using rate limiting and round robin scheduling.

Design enhancement. For the design, Canini et al. (2012) proposed NICE, a tool that efficiently uncovers bugs in OpenFlow programs, through a combination of model checking and symbolic execution. They also presented a simplified OpenFlow switch model (to reduce the state space), and effective strategies for generating event interleavings to uncover bugs. Then, Porras et al. (2012) proposed FortNOX, a software extension that provides role-based authorization and security constraint enforcement for the NOX OpenFlow controller (Gude et al., 2008). It can check flow rule contradictions in real time and there are three authorization roles: human administrators, security applications and non-security-related OF applications. The rule is that no other applications, other than human administrators, can insert flow rules which conflict with those rules inserted by a security application.

Kreutz et al. (2013) presented a general design of a secure and dependable SDN control platform, but no implementation and experimental results were provided. Then, Wen et al. (2013) proposed PermOF, a fine-grained permission system containing a set of OF-specific permissions and an isolation mechanism to enforce the permissions, aiming to safeguard the controller platform. Hu et al. (2014) later introduced FlowGuard, a comprehensive framework, aiming to facilitate accurate detection and effective resolution of firewall policy violations in dynamic OpenFlow-based networks. It checks network flow path spaces to detect firewall policy violations when network states are updated. Kotani and Okabe (2014) proposed a filtering mechanism to reduce Packet-In messages without dropping important ones for network control. In their mechanism, switches record the values of packet header fields before sending Packet-In messages, which are specified by the controllers in advance, and filter out packets that have the same values as the recorded ones.

Overall, to overcome this kind of security issues, it is considered that trust models, security policy enforcement and monitoring tools (Yuzawa, 2013) are effective. Additionally, replication and recovery mechanisms can be applied as well.

4.3. Countermeasures for channel-related security challenges

To mitigate these security challenges, a clean and secure channel should be safeguarded. Obviously, encryption is a necessary and the first step to achieve this goal. For example, applying TLS or SSH or other methods to securing northbound communications and controller management. Then, the communications from the applications and services, which request services or data from the controller, should be secured using authentication and encryption methods. Several existing research studies are listed in Table 6.

Monitoring. To protect the OpenFlow channel, appropriate monitoring is useful to detect malicious states in a quick manner. Liyanage et al. (2014) proposed a secure control channel architecture based on Host Identity Protocol (HIP) to enhance the communication between the switches and the controllers. The architecture uses the IPsec tunneling and security gateways technologies to protect the channel. The experiments reveal that the proposed architecture protects the control channel against various IP based attacks such as DoS, spoofing, replay and eavesdropping attacks. However, they noticed that there is a performance penalty of security due to the establishment of IPsec tunnel. A suggestion to this drawback is to utilize security specific hardware and keep the established tunnels for a longer period.

DoS migration. For this issue, Sherwood et al. (2010) proposed FlowVisor to slice the network hardware by placing a layer between the control plane and the data plane. In implementation, FlowVisor uses OpenFlow and sits between an OpenFlow switch, the forwarding element and multiple OpenFlow controllers. The major issue is that it has no instantiate network security constraints within a slice.

Then, Shin et al. (2013) proposed AVANT-GUARD to integrate both connection migration and actuating triggers in a reference SDN (OpenFlow) software switch. This tool can significantly reduce the amount of data-to-control-plane interactions under DoS attacks, and solve the communication bottleneck between the data

Table 6
A list of existing solutions for channel-related security challenges.

Work in literature	Implementation	Year	Confidentiality	Integrity	Availability
Sherwood et al. (2010)	FlowVisor	2010	√	√	
Shin et al. (2013)	AVANT-GUARD	2013	√	√	√
Liyanage et al. (2014)	HIP	2014	√	√	√

plane and the control plane. The objective of connection migration is to add intelligence to the data plane to differentiate those sources that will complete TCP connections from sources that will not. Actuating triggers enable the data plane to asynchronously report network status and payload information to the control plane. However, it cannot defend against application layer DoS attacks.

Overall, encryption, credential verification and trust mechanisms can be applied to enhance the channel security between the controller and switches.

4.4. Further discussion

It is worth emphasizing that current solutions usually focus on several weak points of OpenFlow-based SDN by combining a set of techniques rather than solving only one security issue. For instance, AVANT-GUARD integrates both connection migration and actuating triggers to address saturation attacks and the responsiveness challenge in SDN (Shin et al., 2013). Wang et al. (2015) developed FloodGuard, which can address the data-to-control plane saturation attack. This attack can produce a large amount of table-miss 'packet_in' messages to consume resources in both control plane and data plane.

Besides, monitoring mechanism should be implemented in any SDN. As an example, Shin and Gu (2012) proposed a framework, called CloudWatcher, to provide monitoring services for large and dynamic cloud networks. It can detour network packets using preinstalled devices and all these operations can be implemented by means of a simple policy script. A weakness is that this approach cannot produce a routing path and may have problems if there are many new flows.

In addition, intrusion detection systems (IDSs) can be tuned from traditional network (Meng et al., 2014) and applied in SDN to identifying malicious flows, rules and commands (Kamisinski and Fung, 2015). As an example, Mehdi et al. (2011) showed how to implement four prominent traffic anomaly detection algorithms in an SDN using Openflow compliant switches and NOX controller. It is expected that more security applications and techniques could be developed to enhance OpenFlow-based SDN in future research studies.

Scanning defend. Network scanners are developed by attackers to collect information about SDN as an initial step to disclose sensitive information. In general, encryption algorithms can be used to defend against this type of attack. In addition, intrusion detection systems can be applied to both switches and controllers to identify various attacks as well (Mehdi et al., 2011; Seeber and Rodosek, 2015). Additionally, some methods focus on constructing filters to redirect malicious traffic. For example, Schehlmann and Harald (2013) presented COFFEE, which can process the whole traffic to filter out candidates of a command-and-control communication (Botnet). By contrast, some methods attempt to take active countermeasures to increase the cost of intruders (Kampanakis et al., 2014).

Spoofing and hijacking defend. Spoofing and hijacking are often part of a large attack such as flooding and DoS attacks. These spoofed and hijacked elements like devices can be further utilized as botnets to launch distributed DoS attacks. Generally, authentication and address validation techniques can be used to defend against these attacks. For example, network information can be frequently changed and hidden from externals, including not only IP addresses, but also topology and routing tables (Alsmadi and Xu, 2015).

DoS defend. This type of attack is a major threat for OpenFlow-based SDN, as it can disable the whole network such as degrading network performance, dropping packets and so on. In addition, as OpenFlow controller is a key component for network control and management, it has become a bottleneck and is greatly threatened by such attacks. Reliable encryption cannot secure SDN from being attacked, but enhanced trust mechanisms and access policies can mitigate such attacks. At the switch level, it is desirable to have a capability to re-evaluate flow table rules, ensuring services to legitimate hosts and denying unauthorized access. Current OpenFlow switches have not implemented such intelligence while extra overhead is the major concern.

On the whole, a desirable security mechanism should ensure both the controller and the switches can quickly recover from large volume traffic attacks. Proper monitoring and response mechanisms are expected to be active when detecting DoS attacks. For instance, Shirali-Shahreza and Ganjali (2013) proposed FleXam, a flexible sampling extension for OpenFlow that enables the controller to access packet-level information, which can help detect DoS attacks. Moreover, flow information is useful in detecting most DoS attacks, through analyzing traffic. As an example, flow headers can be used to detect DoS attacks if there is an unbalance between incoming and outgoing traffic.

5. Future directions

SDN is an emerging architecture, which is suitable for the high-bandwidth, dynamic nature of today's applications. It can help organizations accelerate application deployment and delivery, and dramatically reduce IT costs through policy-enabled workflow automation. It also enables cloud architectures by delivering automated, on-demand application delivery and mobility at scale (Cisco: Software Defined Networking, SDN, 2015). For the future design of OpenFlow-based SDN, more research efforts should be made to protect its switches, controller and communication channel.

OpenFlow switches. At the switch level, more research efforts are expected to be made to securing flows and protecting various switch vulnerabilities.

Based on our analyses above, intruders are likely to conduct various attacks on flows, flow rules and flow tables. Such intrusions can be triggered by faulty (non-malicious) devices or by malicious users. For example, an attacker can use network elements, such as switches, servers and personal computers, to launch a DoS attack against OpenFlow switches (Kreutz et al., 2013). To safeguard the OpenFlow switches, intrusion detection systems can be deployed as the first defense line, with support for runtime root-cause analysis which could help identify abnormal flows. In addition, under SDN, it is possible to directly apply security policies in traffic flows.

On the other hand, taking advantage of switch vulnerabilities, attackers can easily compromise the whole network security. For example, hijacked switches could be used to drop or slow down packets in the network, clone and deviate network traffic (e.g., for data theft), or even inject traffic or forged requests to overload the controller or neighboring switches. For defense, software

attestation such as autonomic trust management is a promising solution. Moreover, monitoring mechanisms are desirable to detect abnormal behaviors of network devices, which can help identify such threat in a fast manner.

OpenFlow controller. The controller is a critical element in OpenFlow-based SDN, more research regarding access control and policy enforcement is desirable to defend against various intrusions.

DoS attacks are a big threat for the controller, as it is the entity that dictates the network behavior. Once an attacker gains access to the control plane, it may be capable of aggregating enough power force to launch DDoS attacks to other elements and allow data leakage in normal traffic flows. As a defense, establishing trust models with multiple certification authorities is a promising solution to reduce unauthorized links. Additionally, the use of dynamic, automated and assured device association mechanisms can be considered to enhance the controller. Taking dynamic address allocation as an example, it can help quickly identify malicious entities and reduce unauthorized traffic (e.g., zombie network).

In addition, attackers can utilize malicious applications from the top level to compromise the controller and cause adverse configuration commands to the underlying infrastructure. To defend against such threat, autonomic trust management could be used to guarantee that one application is trusted during its lifetime. Moreover, it is important to secure all the sensitive elements inside the controller and apply policy enforcement to restrict which interfaces an application can use and what kind of rules it can generate to program the network.

OpenFlow channel. The channel is a very important element for OpenFlow-based SDN, which links the controller and switches. Actually, most enhancement and protections at the channel level can benefit both the controller and switches. In the literature, more studies are expected to research how to design a secure and clean communication channel.

Ensuring TLS in all switches and controllers is a simple way, whereas this may add additional configuration cost on the network operators. Another method of protecting the plaintext communication is to use auto-generated keys and trust-of-first-use method that performs in the same manner as SSH. This may require a network operator to verify the thumbprint of each device's public key, when the first time a switch connects to a controller (Benton et al., 2013).

To defend against man-in-the-middle attacks, it requires to establish a more robust authentication mechanism between the controller and the switches. Furthermore, it may limit the window for a malicious behavior to a small time-frame between when the switch is connected to a network and before it has connected to the controller for the first time.

SDN applications. Our current survey mainly discusses the security of OpenFlow-based SDN itself; however, many studies in the literature have given attention to how to utilize such network to address various security issues. These promising applications help motivate and accelerate the development of OpenFlow & SDN. For example, Braga et al. (2010) proposed a lightweight method to detect flooding attacks based on traffic flow features. Their method was implemented over a NOX-based network, where OpenFlow switches keep flow tables with statistics about all active flows. It then uses Self Organizing Maps, a kind of unsupervised artificial neural networks, and trains them with the selected features. As compared to traditional approaches, this method can extract needed information with a very low overhead. The major weakness is that they did not evaluate its performance in real scenarios.

SDNs are also considered as a solution for large networks. For instance, Casado et al. (2006) developed SANE, a protection layer for enterprise networks to govern all connectivity in the enterprise and a logically centralized server to grant access to services. The major disadvantage is that the highly centralized control network environment cannot be opened to others. By extending this idea of SANE, the same authors proposed Ethane (Casado et al., 2007), a new architecture to allow managers defining a single network wide fine-grain policy and directly enforcing it. Ethane is easier to be implemented than SANE; however, the same problem is that centralized control is vulnerable to attacks without openness.

In addition, Nayak et al. (2009) proposed Resonance, a system enforces dynamic access control policies based on both flow-level information and real-time alerts. The major weakness is that it must depend on the existence of a secure and reliable channel between the controller and switches. It uses programmable switches to manipulate traffic at lower layers; these switches take actions (e.g., dropping or redirecting traffic) to enforce high-level security policies based on inputs from both higher level security policies and distributed monitoring and inference systems.

Moreover, Shin et al. (2013) designed FRESCO, a security application development framework that enables the rapid design and modular composition of OpenFlow-enabled detection and mitigation modules. It exports a scripting API that enables security practitioners to code security monitoring and threat detection logic as modular libraries. These modular libraries represent the elementary processing units in FRESCO, and can be shared and linked together to provide complex network defense applications. It can offer minimal overhead and rapid creation of security functions; however, the framework itself should be further evaluated in the aspect of security. Overall, OpenFlow and SDN present promising applications to mitigate other network security issues, which is an interesting direction. In return, this can help promote their own development.

6. Conclusions

With the separation of the control plane from the data plane, SDN brings a fascinating evolution of networking architectures; however, its network programmability and control logic centralization also increase security risks. OpenFlow is currently the most deployed SDN concept, but due to the rapidly evolving nature, many vendors of both switches and controllers have not fully implemented the specification and have skipped the TLS entirely, which may open a hole for attackers (i.e., launching spoofing, hijacking, DoS attacks, etc.).

In this paper, we focus on OpenFlow-based SDN and analyze security challenges based on OpenFlow switches, controller and channel, by means of the CIA triad model. It is found that security should be an important factor in future SDN design. We later summarize and analyze existing research efforts in enhancing such networks. It is expected that more research efforts should be made to validating elements' address, establishing trust mechanisms between elements, ensuring strong encryption and so on. Our effort attempts to complement existing surveys and stimulate more research studies in this area.

Acknowledgements

The work was partially funded by the Innovation to Realization Funding Scheme of the City University of Hong Kong (under the Project no. 6351018).

References

Al-Shaer, E., Al-Haj, S., 2010. FlowChecker: configuration analysis and verification of federated openflow infrastructures. In: Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration, Chicago, IL, USA, pp. 37–44.

Alsmadi, I., Xu, D., 2015. Security of software defined networks: a survey. Comput. Secur. 53, 79–108.
Benton, K., Camp, L.J., Small, C., 2013. OpenFlow vulnerability assessment. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), Hong Kong, China, pp. 151–152.
Bifulco, R., Karame, G.O., 2014. Towards a richer set of services in software-defined networks. In: Proceedings of SENT, San Diego, CA, USA.
Braga, R., Mota, E., Passito, A., 2010. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: Proceedings of the 35th Conference on Local Computer Networks (LCN), Denver, Colorado, USA, pp. 408–415.
Beacon: Java-Based OpenFlow Controller ⟨https://openflow.stanford.edu/display/Beacon/Home⟩.
Canini, M., Venzano, D., Perešíni, P., Kostić, D., Rexford, J., 2012. A nice way to test OpenFlow applications. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI), San Jose, CA, USA, pp. 1–14.
Casado, M., Garfinkel, T., Akella, A., Freedman, M.J., Boneh, D., McKeown, N., Shenker, S., 2006. SANE: a protection architecture for enterprise networks. In: Proceedings of USENIX Security Symposium, Vancouver, BC, Canada, pp. 1–15.
Casado, M., Freedman, M.J., Pettit, J., Luo, J., McKeown, N., Shenker, S., 2007. Ethane: taking control of the enterprise. In: Proceedings of the 2007 ACM SIGCOMM. Kyoto, Japan, pp. 1–12.
Cisco: Software Defined Networking (SDN), 2015 ⟨http://www.cisco.com/web/solutions/trends/sdn/index.html⟩.
Crenshaw, A., 2012. Security and Software Defined Networking: Practical Possibilities and Potential Pitfalls. Available online ⟨http://www.irongeek.com/i.php?page=security/security-and-software-defined-networking-sdn-openflow⟩.
Dover, J., 2013. A Denial of Service Attack Against the Open Floodlight SDN Controller. Dover Networks, Research Report.
Farhady, H., Lee, H., Nakao, A., 2015. Software-defined networking: a survey. Comput. Netw. 81, 79–95.
Fayazbakhsh, S.K., Sekar, V., Yu, M., Mogul, J.C., 2013. FlowTags: enforcing network-wide policies in the presence of dynamic middlebox actions. In: Proceedings of the 2nd ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), pp. 19–24.
Feng, T., Bi, J., Hu, H., 2011. OpenRouter: openflow extension and implementation based on a commercial router. In: Proceedings of the 19th IEEE International Conference on Network Protocols (ICNP), Vancouver, BC, pp. 141–142.
Feng, T., Bi, J., Hu, H., Yao, G., Xiao, P., 2012. InSAVO: Intra-AS IP source address validation solution with OpenRouter. In: Proceedings of INFOCOM.
Fichera, S., Galluccio, L., Grancagnolo, S.C., Morabito, G., Palazzo, S., 2015. OPERETTA: an openflow-based remedy to mitigate TCP SYNFLOOD attacks against web servers. Comput. Netw 92, 89–100.
Floodlight: Apache-Licensed, Java-Based OpenFlow Controller ⟨http://www.projectfloodlight.org/floodlight/⟩.
Greenk, K., 2009. TR10: software-defined networking. MIT Technol. Rev. 112 (2), 54.
Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S., 2008. NOX: towards an operating system for networks. ACM Comput. Commun. Rev. 51 (7), 105–110.
Hampel, G., Steiner, M., Bu, T., 2013. Applying software-defined networking to the telecom domain. In: Proceedings of 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Turin, Italy, pp. 133–138.
Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D.C., Gayraud, T., 2014. Software-defined networking: challenges and research opportunities for future internet. Comput. Netw. 75, 453–471.
Hong, S., Xu, L., Wang, H., Gu, G., 2015. Poisoning network visibility in software-defined networks: new attacks and countermeasures. In: Proceedings of NDSS, San Diego, USA.
Hu, F., Hao, Q., Bao, K., 2014. A survey on software-defined network and OpenFlow: from concept to implementation. IEEE Commun. Surv. Tutor. 16 (4), 2181–2206.
Hu, H., Han, W., Ahn, G.-J., Zhao, Z., 2014. FLOWGUARD: building robust firewalls for software-defined networks. In: Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN), Chicago, Illinois, USA, pp. 97–102.
IRIS ⟨https://github.com/bjlee72/IRIS⟩.
Jammal, M., Singh, T., Shami, A., Asal, R., Li, Y., 2014. Software defined networking: state of the art and research challenges. Comput. Netw. 72, 74–98.
Jagadeesan, N.A., Krishnamachari, B., 2015. Software-defined networking paradigms in wireless networks: a survey. ACM Comput. Surv. 47 (2).
Jafarian, J.H., Al-Shaer, E., Duan, Q., 2012. Openflow random host mutation: transparent moving target defense using software defined networking. In: Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, pp. 127–132.
Jaxon ⟨http://jaxon.onuos.org/⟩.
Kampanakis, P., Perros, H., Beyene, T., 2014. SDN-based solutions for moving target defense network protection. In: Proceedings of the 2014 IEEE 15th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Sydney, NSW, pp. 1–6.
Kamisinski, A., Fung, C., 2015. FlowMon: detecting malicious switches in software-defined networks. In: Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense (SafeConfig).
Khurshid, A., Zhou, W., Caesar, M., Godfrey, P.B., 2012. VeriFlow: verifying network-wide invariants in real time. In: Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks (HotSDN), Helsinki, Finland, pp. 49–54.
Kloeti, R., Vasileios, K., Paul, S., 2013. OpenFlow: a security analysis. In: Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP), Gottingen, German, pp. 1–6.
Kotani, D., Okabe, Y., 2014. A packet-in message filtering mechanism for protection of control plane in openflow networks. In: Proceedings of the 10th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 29–40.
Kreutz, D., Ramos, F.M.V., Verissimo, P., 2013. Towards secure and dependable software-defined networks. In: Proceedings of the 2nd ACM Workshop on Hot Topics in Software Defined Networking (HotSDN), Hong Kong, China, pp. 55–60.
Lara, A., Kolasani, A., Ramamurthy, B., 2014. Network innovation using openflow: a survey. IEEE Commun. Surv. Tutor. 16 (1), 493–512.
Lara, A., Ramamurthy, B., 2014. OpenSec: a framework for implementing security policies using OpenFlow. In: Proceedings of the IEEE Global Communications Conference (GLOBECOM), Austin, Texas, pp. 781–786.
Li, L.E., Mao, Z.M., Rexford, J., 2012. Toward software-defined cellular networks. In: Proceedings of 2012 European Workshop on Software Defined Networking (EWSDN), Darmstadt, Germany, pp. 7–12.
Liyanage, M., Ylianttila, M., Gurtov, A., 2014. Securing the control channel of software-defined mobile networks. In: Proceedings of the 2014 IEEE 15th International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM), Sydney, Australia, pp. 1–6.
Liu, J., Li, Y., Wang, H., Jin, D., Su, L., Zeng, L., Vasilakos, T., 2016. Leveraging software-defined networking for security policy enforcement. Inf. Sci. 327, 288–299.
Luo, T., Tan, H.P., Quek, T.Q.S., 2012. Sensor OpenFlow: enabling software-defined wireless sensor networks. IEEE Commun. Lett. 16 (11), 1896–1899.
Matias, J., Borja, T., Alaitz, M., Jacob, E., Nerea, T., 2012. Implementing layer 2 network virtualization using OpenFlow: challenges and solutions. In: Proceedings of the 2012 European Workshop on Software Defined Networking, pp. 30–35.
Matias, J., Garay, J., Mendiola, A., Toledo, N., Jacob, E., 2014. FlowNAC: flow-based network access control. In: Proceedings of the 3rd European Workshop on Software Defined Networks (EWSDN), Budapest, pp. 79–84.
Mehdi, S.A., Khalid, J., Khayam, S.A., 2011. Revisiting traffic anomaly detection using software defined networking. In: Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection (RAID), Menlo Park, California, pp. 161–180.
Mendonca, M., Seetharaman, S., Obraczka, K., 2012. A flexible innetwork IP anonymization service. In: Proceedings of the 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, pp. 6651–6656.
Meng, W., Li, W., Kwok, L.F., 2014. EFM: enhancing the performance of signature-based network intrusion detection systems using enhanced filter mechanism. Comput. Secur. 43, 189–204.
Maestro Project ⟨http://zhengcai.github.io/maestro-platform/⟩.
Nayak, A.K., Reimers, A., Feamster, N., Clark, R., 2009. Resonance: dynamic access control for enterprise. In: Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN), Barcelona, Spain, pp. 11–18.
Open Networking Foundation, 2012. Sofware-Defined Networking: The New Norm for Networks. White Paper, April ⟨https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf⟩.
OpenFlow Switch Specification Version 1.5.0, 2014. Technical Report, December 19 ⟨https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf⟩.
Pentikousis, K., Wang, Y., Hu, W., 2013. Mobileflow: toward software-defined mobile networks. IEEE Commun. Mag. 51 (7), 44–53.
Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M., Gu, G., 2012. A Security Enforcement Kernel for OpenFlow Networks. In: Proceedings of the 1st Workshop on Hot topics in Software Defined Networks (HotSDN), pp. 121–126.
Perrin, C., 2012. The CIA Triad, Retrieved May 31 ⟨http://www.techrepublic.com/blog/it-security/the-cia-triad/⟩.
POX: A Python-Based OpenFlow Controller ⟨https://github.com/noxrepo/pox⟩.
RouteFlow Open Source Project ⟨https://sites.google.com/site/routeflow/home⟩.
Ryu ⟨https://github.com/osrg/ryu⟩.
Scott, H., 2014. SDN Security Attack Vectors and SDN Hardening. Available online ⟨http://www.networkworld.com/article/2840273/sdn/⟩.
Scott-Hayward, S., O'Callaghan, G., Sezer, S., 2013. SDN security: a survey. In: Proceedings of the 2013 IEEE SDN for Future Networks and Services (SDN4FNS), Trento, pp. 1–7.
Shirali-Shahreza, S., Ganjali, Y., 2013. Efficient implementation of security applications in OpenFlow controller with FleXam. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), Hong Kong, pp. 167–168.
Schehlmann, L., Harald, B., 2013. COFFEE: a concept based on OpenFlow to filter and erase events of botnet activity at highspeed nodes. In: Proceedings of INFORMATIK, Darmstadt.
Seeber, S., Rodosek, G.D., 2015. In: Proceedings of the 9th International Conference on Autonomous Infrastructure, Management, and Security (AIMS), Ghent, Belgium, pp. 134–139.
Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., Parulkar, G., 2010. Can the production network be the testbed. In: Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI), Vancouver, BC, pp. 1–6.
Shin, S., Gu, G., 2012. CloudWatcher: network security monitoring using OpenFlow in dynamic cloud networks. In: Proceedings of the 20th IEEE International Conference on Network Protocols (ICNP), Austin, TX, pp. 1–6.
Shin, S., Porras, P., Yegneswaran, V., Fong, M., Gu, G., Tyson, M., 2013. FRESCO: modular composable security services for software-defined networks. In: Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, pp. 1–16.

Shin, S., Gu, G., 2013. Attacking software-defined networks: a first feasibility study. In: Proceedings of the 2nd Workshop on Hot Topics in Software Defined Networks (HotSDN), Hong Kong, China, pp. 165–166.
Shin, S., Yegneswaran, V., Porras, P., Gu, G., 2013. AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS), ACM Press, Berlin, Germany, pp. 413–424.
Sonchack, J., Aviv, A.J., Keller, E., Smith, J.M., 2015. OFX: enabling openflow extensions for switch-level security applications (POSTER). In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS).
Suh, J., Choi, H., Yoon, W., You, T., Kwon, T., Choi, Y., 2010. Implementation of content-oriented networking architecture (CONA): a focus on DDoS countermeasure. In: Proceedings of the 1st European NetFPGA Developers Workshop, ACM Press, Cambridge, UK, pp. 1–5.
SNAC ⟨https://github.com/bigswitch/snac⟩.
Tavakoli, A., Casado, M., Koponen, T., Shenker, S., 2009. Applying NOX to the datacenter. In: Proceedings of HotNet. ACM Press, New York City, NY, pp. 1–6.
Tootoonchian, A., Ganjali, Y., 2010. HyperFlow: a distributed control plane for openflow. In: Proceedings of the Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN). Usenix, San Jose, CA, pp. 1–6.
Wang, H., Xu, L., Gu, G., 2014. OF-GURAD: A DoS Attack Prevention Extension in Software-Defined Networks. ONS.
Wang, H., Xu, L., Gu, G., 2015. FloodGuard: a DoS attack prevention extension in software-defined networks. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN), pp. 239–250.
Wen, X., Chen, Y., Hu, C., Shi, C., Wang, Y., 2013. Towards a secure controller platform for openflow applications. In: Proceedings of the 2nd ACM Workshop on Hot Topics in Software Defined Networking (HotSDN), Hong Kong, China, pp. 171–172.
Xiao, P., Bi, J., Feng, T., 2013. O-CPF: an OpenFlow based intra-AS source address validation application. In: Proceedings of CFI, Beijing, China.
Yamasaki, Y., Miyamoto, Y., Yamato, J., Goto, H., 2011. Flexible access management system for campus VLAN based on OpenFlow. In: Proceedings of 2011 IEEE/IPSJ 11th International Symposium on Applications and the Internet (SAINT), Munich, Germany, pp. 347–351.
Yao, G., Bi, J., Xiao, P., 2011. Source address validation solution with OpenFlow/NOX architecture. In: Proceedings of the IEEE International Conference on Network Protocols (ICNP), Vancouver, BC, Canada, pp. 7–12.
Yuzawa, T., 2013. OpenFlow 1.0 Actual Use-Case: RTBH of DDoS Traffic While Keeping the Target.