

Outsmarting Network Security with SDN Teleportation

Kashyap Thimmaraju
Security in Telecommunications
TU Berlin
Berlin, Germany
Email: kash@sect.tu-berlin.de

Liron Schiff
GuardiCore Labs
Tel Aviv, Israel
Email: liron.schiff@guardicore.com

Stefan Schmid
Dept. of Computer Science
Aalborg University
Aalborg, Denmark
Email: schmiste@cs.aau.dk

Abstract—Software-defined networking is considered a promising new paradigm, enabling more reliable and formally verifiable communication networks. However, this paper shows that the separation of the control plane from the data plane, which lies at the heart of Software-Defined Networks (SDNs), introduces a new vulnerability which we call *teleportation*. An attacker (e.g., a malicious switch in the data plane or a host connected to the network) can use teleportation to transmit information via the control plane and bypass critical network functions in the data plane (e.g., a firewall), and to violate security policies as well as logical and even physical separations. This paper characterizes the design space for teleportation attacks theoretically, and then identifies four different teleportation techniques. We demonstrate and discuss how these techniques can be exploited for different attacks (e.g., exfiltrating confidential data at high rates), and also initiate the discussion of possible countermeasures. Generally, and given today’s trend toward more intent-based networking, we believe that our findings are relevant beyond the use cases considered in this paper.

1. Introduction

Computer networks such as datacenter networks or the Internet have become a critical infrastructure [1]. Not only a large fraction of the economic activity critically depends on the availability of such networks, but also governments increasingly rely on existing and shared infrastructures, mainly for their cost benefits [2].

This dependency on public and shared infrastructures raises concerns. While the Internet has certainly been a huge success, and over the last years, there has been much innovation on the higher network layers (e.g., application layer) and the lower network layers (e.g., data-link and physical layer), the Internet core suffers from ossification [3]. In particular, it is questionable whether today’s network technology is sufficient to ensure essential security, resilience and dependability properties. For instance, today’s Internet does not provide any means of path control, and we are still struggling to render routing protocols more secure [4].

Software-Defined Networking is a novel networking paradigm which promises to enable these necessary innova-

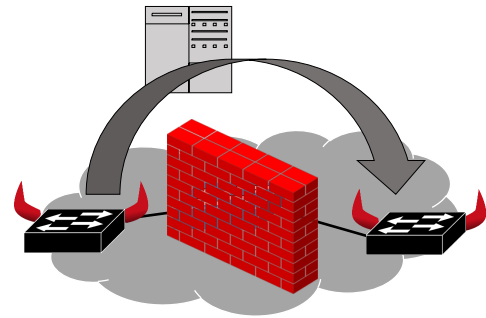


Figure 1: Illustration of teleportation: Malicious switches (with red horns) exploit the control platform for hidden communication, possibly bypassing data plane security mechanisms such as a firewall.

tions, also in terms of security, through its decoupling and consolidation of the control plane, its formally verifiable policies [5], [6], [7], [8], as well as by introducing new functionality [9], [10], [11], [12].

However, Software-Defined Networks (SDNs) also introduce new security challenges. In particular, we in this paper study threats introduced by an *unreliable south-bound interface*, i.e., we consider a threat model in which switches or routers do not behave as expected, but rather are malicious [13], and e.g., contain hardware backdoors [14]. While many existing network security and monitoring tools rely on the trustworthiness of switches and routers, this assumption has become questionable: Attackers have repeatedly demonstrated their ability to compromise switches and routers [15], [16], [17], thousands of compromised access and core routers are being traded underground [18], networking vendors have left backdoors open [19], [20], national security agencies can bug network equipment [14], hacker tools to scan and eventually exploit routers with weak passwords, default settings are openly available on the Web, etc. However, the impact of malicious hardware is not well understood and underexplored today.

In particular, this paper shows how an outsourced and

consolidated control plane—as it lies at the heart of the SDN paradigm, introduces an opportunity for *teleportation*: Malicious SDN switches may transmit information via the logically centralized software control plane, *completely* bypassing data plane elements (such as other switches, middleboxes, etc.). By violating logical and even physical separations, teleportation can constitute a serious security threat. For example, teleportation could be used by one malicious switch to discover (and communicate information to) other malicious switches, bypassing security checks in the data plane. As we will show in this paper, teleportation can also be exploited by malicious hosts, triggering (benign) switches to teleport information for them.

We argue that teleportation can be seen as a flexible communication channel which constitutes a threat in various situations, for example (see Figure 1):

- 1) **Bypassing critical network components:** By implicitly communicating information via the control plane, it is possible to circumvent critical network components, such as switches, middleboxes or policy enforcers located in the data plane. For example, teleportation can in principle be used to bypass middleboxes performing security checks (e.g., network intrusion detection systems), middleboxes in charge of billing (e.g., a Radius server), or QoS enforcers (e.g., a leaky bucket policer).
- 2) **Rendezvous and attack coordination:** While already a single malicious switch, for example located *inside* an enterprise network, may cause significant harm and violate basic security policies, the situation becomes worse if multiple malicious switches cooperate [21]. Malicious or Trojan switches (e.g., switches containing a hardware/software backdoor) may use teleportation as a rendezvous protocol, to discover each other, and subsequently coordinate an attack.
- 3) **Exfiltration:** Teleportation can also be used to exfiltrate sensitive information between networks that have no data plane connectivity.
- 4) **Eavesdropping and data tampering:** Particularly serious threats are introduced if malicious hosts and switches collude. For instance, in a scenario with collusion, teleportation can be used for eavesdropping. We show that a malicious switch and host can carry out a man-in-the-middle attack that serves benign hosts with malicious web pages.

Teleportation can be difficult to detect: The teleported information follows the normal traffic pattern of control communication, not between switches directly but *indirectly* between any switch and the controller. Moreover, the teleportation channel is *inside* the (typically encrypted) OpenFlow channel. Accordingly, it cannot easily be detected with modern Network Intrusion Detection Systems (NIDSs), even if they operate in the control plane.

1.1. Our Contributions

This paper makes the following contributions:

- 1) We identify a new vulnerability, namely *teleportation*, which targets the very core of the software-defined networking paradigm, namely the separation of the control and the data plane. In particular, we consider the threats introduced by a malicious data plane. Indeed, recent incidents related to the trustworthiness of routers and switches, indicate that our threat model is a relevant one.
- 2) We model and characterize possible teleportation channels theoretically.
- 3) We recognize and demonstrate four teleportation techniques in software-defined networks utilizing state-of-the-art OpenFlow controllers, in particular ONOS [22] among others.
- 4) We present and demonstrate multiple simple and sophisticated attacks. In particular, we show that teleportation can also be exploited by malicious hosts in scenarios where all switches are benign.
- 5) We evaluate the performance and quality of the teleportation channel in terms of throughput, jitter and packet loss respectively, and also evaluate the resource footprint in terms of CPU and memory consumption at the controller.
- 6) We initiate the discussion of possible countermeasures. In particular, we propose to combine intrusion detection with waypoint-enforcement, forcing *Packet-out* messages (from controller to switches) to pass through the waypoint if mandated by a security policy.

We have already notified the open source community about some of the issues reported in this paper, and first actions have been taken (see CVE-2015-7516 [23]).

More generally, in the light of today’s trend toward more intent-based networking, we believe that our work touches a topic whose relevance may increase in the near future and go beyond the use cases considered in this paper.

1.2. Paper Organization

The remainder of this paper is organized as follows. Section 2 introduces the necessary background on OpenFlow and SDN. Section 3 introduces our threat model, and Section 4 characterizes possible teleportation channels. Section 5 describes teleportation techniques; based on these techniques, we demonstrate and discuss different attacks in Section 6. Section 7 describes our performance evaluation of the out-of-band forwarding channel. Section 8 initiates the discussion of countermeasures. After reviewing related work in Section 9, we conclude our work in Section 10.

2. Preliminaries

This paper considers Software-Defined Networks (SDNs) which outsource and consolidate the control over the network

switches to a logically centralized software controller. The separation of the control and data plane has the potential to simplify the network management, as many networking tasks are inherently non-local. Moreover, SDN and especially its de facto standard protocol, OpenFlow, introduce interesting new flexibilities, e.g., in terms of traffic steering: Routes may not necessarily be destination based, and can depend on layer-2, layer-3 and even layer-4 properties of the packets.

At the heart of an SDN lies a control software, running on a set of servers. These controllers receive information and statistics from switches, and depending on this information as well as the policies they seek to implement, issue instructions to the switches.

OpenFlow follows a match-action paradigm: The controllers install rules on the switches which consist of a match and an action part; the packets (i.e., flows) matching a rule are subject to the corresponding action. That is, each switch stores a set of (flow) tables which are managed by the controllers, and each table consists of a set of flow entries which specify expressions that need to be matched against the packet headers, as well as actions that are applied to the packet when a given expression is satisfied. Possible actions include dropping the packet, sending it to a given egress port, or modifying its header fields, e.g., adding a tag. The match-action paradigm is attractive as it simplifies formal reasoning and enables policy verification.

By default, if a packet arrives at a switch and does not match an existing rule, the packet (usually without payload if the switch supports packet buffering) is forwarded to the controller. This event is called a *Packet-in*. Upon a *Packet-in* event, the controller can decide how to react to packets of the corresponding type, and add/delete/modify flow rules accordingly issuing *Flow-mod* messages to the switch (and maybe to other switches proactively on this occasion as well). A controller can also decide to send out a packet explicitly from a switch, issuing a so-called *Packet-out* command to the switch.

An attractive alternative to the hop-by-hop installation of new flows, reacting to a new packet repeatedly along the path (multiple *Packet-ins*), is the so-called “pave-path technique”: Once the controller receives a first *Packet-in* event from some switch, it proactively updates the other switches along the path. Such an “intent-based” controller behavior can render the reaction to network events and set up of host-to-host/network connectivity (according to current policies) more efficient.

While SDNs are logically centralized, the control plane can be physically distributed, e.g., for fault-tolerance or performance reasons. Accordingly, OpenFlow supports multiple controllers for a single switch. The controllers and switch exchange *Role-request* and *Role-reply* messages respectively to assert the various roles (*Master*, *Equal* and *Slave*). There may be only one *Master* controller for a given switch while multiple *Equal* and *Slave* controllers are permitted.

The OpenFlow standard [24] specifies basic security mechanisms. For example, the communication between the controller and switch can be authenticated and encrypted, using TLS over TCP/UDP.

Finally, we note that although some of our techniques are generally applicable in networks separating the control plane and the data plane, while others exploit OpenFlow specific features, when clear from the context, in this paper we will treat SDN and OpenFlow as synonyms.

3. Threat Model

We in this paper consider a threat model where OpenFlow switches, hosts, or both, may not behave correctly but are malicious.

We do not place any restrictions on what a malicious switch can and cannot do. For example, a malicious switch can fabricate and transmit any type of OpenFlow message, it can arbitrarily deviate from the OpenFlow specification, and it can even use multiple identities, all at the risk of being detected. However, the malicious switch cannot choose where it will be placed in the network. In order to collude, the malicious switches have been programmed to recognize some data and/or timing pattern. Similarly, we do not place any restrictions on what a malicious host can and cannot do. For example, a malicious host may masquerade its Media Access Control (MAC) and/or Internet Protocol (IP) addresses, use an incorrect gateway, falsify Address Resolution Protocol (ARP) requests/responses, and so on. The attacker could also be an insider, i.e., an authorized user who intends to subvert his/her current organization. We also consider the case where malicious hosts and switches collude. We assume that an attacker has sufficient resources and know-how to compromise hosts/switches and therefore do not concern ourselves with how the host/switch is compromised. For example, the attacker can exploit a buffer overflow vulnerability in the switch software to compromise the switch [13].

The OpenFlow controller and its applications on the other hand are trusted entities and are available to the switches: For example, they are based on static and dynamic program analyses. The OpenFlow channel is reliable and may be encrypted.

4. Modeling Teleportation

With these concepts in mind, we now model and characterize a novel threat called *teleportation* which targets the heart of SDNs: The outsourcing and consolidation of control over multiple data plane elements. In particular, we argue that we can see an OpenFlow controller as a “reactor”: It reacts (in a best-effort and timely manner) to events generated by the network operator, the OpenFlow switches, and timeouts; as a response, the controller sends OpenFlow commands to switches. Accordingly, we argue that the following 3-stage functionality is fundamental in the SDN paradigm.

- 1) **Switch to controller:** A source switch intentionally or unintentionally sends modulated information to the controller (e.g., by adding specific events, delaying existing events, etc.).

- 2) **Controller to switches:** The controller reacts to the received events, by sending commands to one or multiple other switches.
- 3) **Destination processing:** A destination switch processes incoming commands from the controller. In case of a malicious switch, the switch may search for some message properties, temporal patterns, etc., and hence infer the information modulated by the source, or by simply forwarding the information (to a potentially malicious host).

Based on this controller model, we can identify two kinds of teleportation channels ¹:

- **Explicit teleportation:** The teleported information actually appears in the messages exchanged. The message may for example contain steganographic contents.
- **Implicit teleportation:** The teleportation relies on modulating information implicitly. For example, it is based on timing (e.g., message transmissions are delayed according to some pattern) or it is based on shared resources, whose availability is changed over time (e.g., leveraging mutual exclusion).

5. Teleportation Techniques

Having established a conceptual model of teleportation, we next present techniques that can realize teleportation in today's SDNs. In particular, we have identified the following three fundamental SDN functionalities which can be exploited for teleportation:

- 1) **Flow (re-)configurations:** In an SDN, a controller needs to react to various data plane events (such as so-called *Packet-ins* in OpenFlow or link failures), and configure and reconfigure flows and paths accordingly. Triggering and exploiting such events can be used for teleportation.
- 2) **Switch identification:** In an SDN, switches are responsible for introducing and uniquely identifying themselves to the controller. This is required as policies are often specific to the switch. Unique switch identifiers are also necessary to correctly construct and enforce policies on the switches and in the controller. We will show that such switch identification mechanisms can be exploited for teleportation.
- 3) **Out-of-band forwarding:** An SDN controller must not only be able to receive events and control packets from switches, but also to instruct switches to forward specific messages. This basic functionality in SDNs can be exploited by a malicious switch or host to forward entire packets via the controller.

In the remainder of this section, we will discuss these teleportation techniques in more detail in turn.

¹ We note that our terminology of teleportation can be viewed as analogous to covert channels. Explicit teleportation is analogous to covert storage channels and implicit teleportation is analogous to covert timing channels.

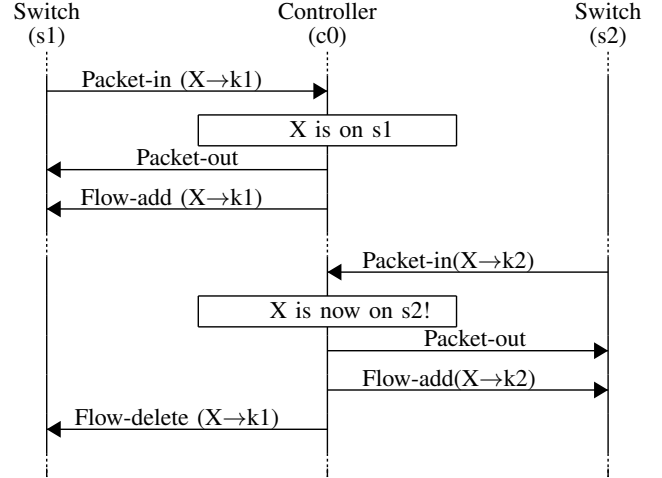


Figure 2: Message sequence pattern for path update teleportation. Switch $s2$ teleports information to $s1$ when $s1$ receives the Flow-delete message from controller $c0$.

5.1. Flow (Re-)Configurations

We distinguish between two types of flow re-configuration events: *path update* and *path reset*.

Path Update: Our first teleportation technique is based on *path updates*. Path updates are a fundamental controller functionality, and come in the form of different controller features such as *Mobility*, *VM Migration* or simply *MAC Learning*. The basic scheme is as follows: A controller typically maintains some mapping of which hosts (MAC addresses) are connected to which ports (on the switch). If a host suddenly appears on another switch, the controller installs new flows for the host on the new switch, and also deletes the corresponding flow rules on the old switch. We define this type of installation and deletion of flows by the controller on switches as *path update*. Specifically, a path update involves the use of *Packet-in*, *Flow-mod* and *Packet-out* messages. Malicious switches can use path update for implicit teleportation.

For the teleportation with path update, a switch triggers the deletion of rules at other switches. Malicious switches can teleport information between themselves by prompting path updates for the same host using *Packet-ins*. Note that during a path update, the *Packet-out* is sent to the destination reported in the *Packet-in* which may generate data plane traffic. To prevent data plane traffic, the malicious switch can use a destination host that is connected to itself (so that the *Packet-out* is sent back to it). The message sequence pattern for path update teleportation is shown in Figure 2.

We can summarize the scheme presented so far with the following abstract steps: A switch $s1$ announces X , a switch $s2$ announces X thereby stealing X from $s1$, where stealing is detected by the “victim” ($s1$). Also note that announcing is possible once some host which is connected to the malicious switch (e.g., $k1$ at $s1$) is learned by the controller.

process incoming OpenFlow message :

```

on start teleportation :
  - announce all  $\{X_{i,j}\}_{j \in [m]}$  and  $\{X_{j,i}\}_{j \in [m]}$ 
on received Flow-delete for  $X_{i,j}$  for some  $j \in [m]$  :
  - announce  $X_{i,j}$ 
  - add  $j$  to Discovered_Switches

```

Algorithm 1: Generalized pseudo-code executed by switch s_i to teleport information using path update.

Based on these basic steps, we can generalize our scheme for m malicious switches. Each malicious switch, s_i , with id $i \in [m]$, should implement an event handler (see Algorithm 1), in addition to the normal (non-malicious) behavior. We assume that all switches are programmed with the same list of m^2 special MAC addresses $\{X_{i,j}\}_{i,j \in [m]}$ (the *pre-shared secrets*). Note that once switch i discovers switch j , it can contact it by sending packets with source $X_{i,j}$ and destination address $X_{j,i}$.

Path Reset: We next discuss a second flow reconfiguration based teleportation which we refer to as *path reset*. Recall that at the heart of any SDN controller lies the functionality to set up host-to-host/network connectivity, according to the network policy (e.g., defining constraints such as bandwidth, link type and waypoints), which is translated into device level configurations (e.g., flow rules). The “pave-path technique” is an attractive alternative to the hop-by-hop installation of new flows: Once the controller receives a first *Packet-in* event from some switch, it proactively updates the other switches along the path.

In order to provide high availability, a controller also monitors the network state and makes necessary changes, such as rerouting or resetting flows on switches, when needed (e.g., due to a link failure). For example, triggered by a *Packet-in* event, a controller may learn that (parts of) the path may no longer be available, and hence initiates the reconfiguration/repair of the path. We define the reinstallation of flows by the controller on switches along a path as *path reset*. Accordingly, the path reset technique involves *Packet-in*, *Flow-mod* and *Packet-out* messages.

Malicious switches may use path reset for implicit teleportation: If the controller resets the complete path between hosts when it receives a *Packet-in* from a switch that ignores the flow rule, then information can be communicated. By doing this at multiple and specific times, a malicious switch can teleport information to other malicious switches along the path. Figure 3 illustrates the message sequence pattern for teleportation using path reset.

5.2. Switch Identification

This teleportation type exploits the fact that a switch typically must uniquely identify itself whenever it connects to the controller. For example, in OpenFlow this is usually done using the Datapath-ID (DPID) field in the *Features-reply* message. We define two switches attempting to use the same DPID to connect to the same logical controller as

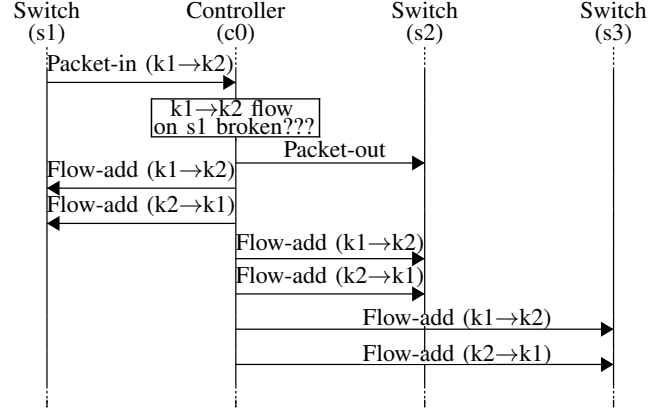


Figure 3: Message sequence pattern for path reset teleportation. Switch s_1 teleports information to s_2 and s_3 via *Flow-add* messages sent by the controller c_0 .

switch identification. The outcome can be used for implicit teleportation.

Three basic ways an OpenFlow controller can react to using the same DPID are as follows:

- 1) The controller denies the second switch a connection.
- 2) The controller terminates the first switch and connects to the second.
- 3) The controller accepts both switches but sends them different *Role-request* messages.

With any of the above outcomes, a switch can infer the (mis)use of the same DPID by another switch. By using a-priori configured single or multiple DPID values, a pair of malicious switches can establish teleportation. For example, consider the message sequence pattern in Figure 4, and assume that first switch s_1 tells controller c_0 that its DPID is 1. At a later time, switch s_2 tells c_0 that its DPID is 1. At this point, c_0 does not allow s_2 to connect with DPID 1. Since c_0 denied s_2 to connect with DPID 1, s_1 teleported information to s_2 via c_0 . With a similar message sequence pattern, the second outcome can be used for teleportation as well.

Interestingly, switch identification is not limited to scenarios with a single controller: We have found additional threats in the presence of *distributed control planes*. Moreover, we can generalize the first *switch identification* outcome to scenarios with m malicious switches, see the event-handler algorithm, Algorithm 2. The other two outcomes discussed can also be seen as event-handler algorithms.

5.3. Out-of-band Forwarding

The third and potentially most powerful teleportation technique is called *out-of-band forwarding*. It is an example of explicit teleportation. Out-of-band forwarding exploits the fact that an SDN controller is typically connected to multiple switches: Accordingly, a packet from one switch

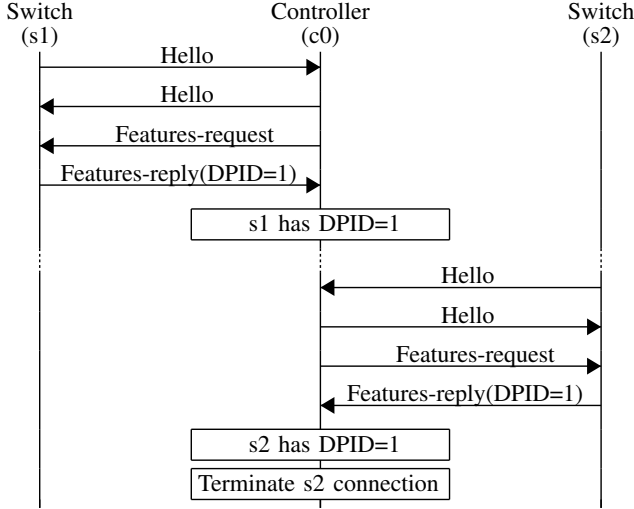


Figure 4: Message sequence pattern for *switch identification* teleportation when the controller denies the second switch a connection. When s_2 's connection is terminated, s_1 successfully teleports information to s_2 .

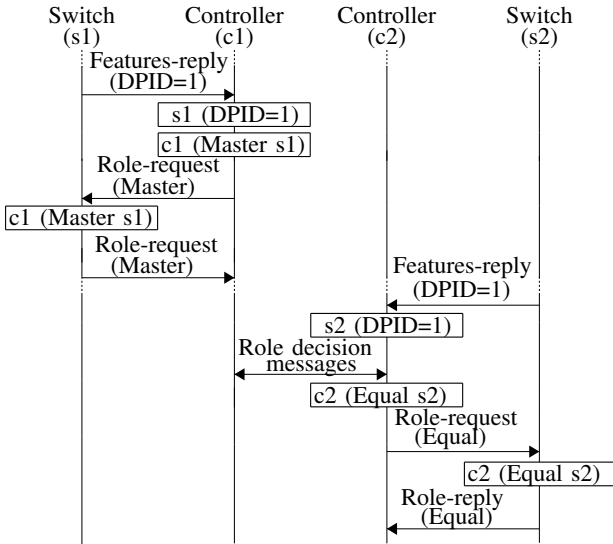


Figure 5: Message sequence pattern for *switch identification* teleportation when controllers c_1 and c_2 send different *Role-request* messages to s_1 and s_2 respectively. When s_1 receives the *Role-request=Master* message whereas s_2 receives the *Role-request=Equal* message. In this manner s_1 teleports information to s_2 when s_2 received the *Role-request=Equal* message.

can potentially reach multiple other switches in the network via the control plane. Out-of-band forwarding involves a *Packet-in* from one switch and a *Packet-out* message at another switch, with the possible side effect of *Flow-mod* messages on the switch that sent the *Packet-in* message. Out-of-band forwarding could for example include the complete Ethernet frame (typically 1500 bytes), and can even serve

```

process connect to OpenFlow controller :
  on Features-request message from controller :
    - announce DPID  $\{X_{i,j}\}_{j \in [m]}$  in Features-reply message
  on Controller denies connection to announced DPID  $X_{i,j}$  for some  $j \in [m]$  :
    - add  $j$  to Discovered_Switches

```

Algorithm 2: Generalized pseudo-code executed by switch s_i for *switch identification* teleportation when the controller denies the second switch a connection.

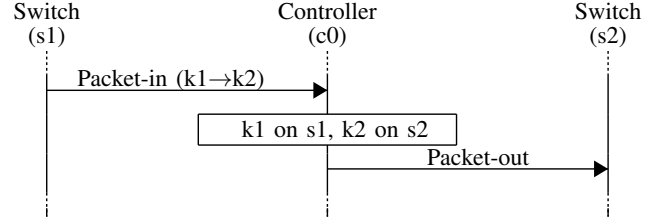


Figure 6: Message sequence pattern for *out-of-band forwarding* teleportation. The controller c_0 receives the *Packet-in* from s_1 and accordingly sends a *Packet-out* to s_2 , successfully teleporting packets from s_1 to s_2 .

as a “multicast service”. Out-of-band forwarding can be a serious threat to network security, not only because malicious traffic can bypass critical security functions in the data plane, but also because it can be exploited by switches and hosts. Figure 6, illustrates the message sequence pattern for teleportation using out-of-band forwarding.

A summary of our teleportation techniques is shown in Table 1 along with the type of teleportation and the associated OpenFlow messages.

6. Switch- and Host-based Attacks

We now demonstrate how the identified teleportation techniques can be exploited to carry out specific attacks. In particular, we show how teleportation may be exploited:

- 1) To bypass security critical network functions such as firewalls and NIDSs;
- 2) As a rendezvous protocol for malicious switches;
- 3) To exfiltrate sensitive data from remote locations;
- 4) To conduct a man-in-the-middle (mitm) attack.

Along the way, we also present a novel denial-of-service (dos) attack (published as a CVE-2015-7516).

Before presenting the attacks in more detail, we report on the setup we used to verify the attacks.

6.1. Setup

We verified all our attacks in a virtual machine, using Mininet-2.2.0 and Open vSwitch-2.0.1 for the data plane. For the control plane we used ONOS-1.1.0 as it was the state-of-the-art. At the time of our experimentation Floodlight,

TABLE 1: Summary of teleportation techniques, types and associated threats.

Technique	Type	Threat
Flow (Re-)Configuration	Implicit	Covert communication and coordination.
Switch Identification	Implicit	Attack coordination.
Out-of-band Forwarding	Explicit	Exfiltration, firewall/NIDS bypass and man-in-the-middle.

OpenDaylight Lithium-SR2, and RYU v3.27, still did not support the intent based framework. Indeed our experiments showed that they were only vulnerable to a subset of the attacks (e.g., switch identification, out-of-band-forwarding²) For packet generation we use ping and nmap-6.40. We use *ebtables* v2.0.10-4 (December 2011) as our transparent firewall and *Snort* version 2.9.6.0 GRE (Build 47) as our NIDS. We modified code developed by *austinmarton* [25] to set the Ethertype field in an Ethernet frame. *ettercap* 0.8.0 was used with a custom HTTP filter for the mitm attack.

6.2. Bypassing Critical Network Functions

We believe that the possibility to bypass network elements is a serious threat in modern computer networks. For example, many network policies today are defined in terms of adjacency matrices or big switch abstractions, specifying which traffic is allowed between an ingress port s and an egress network port t [26]. In order to enforce such a policy, traffic from s to t needs to traverse a middlebox instance (waypoint) inspecting and classifying the flows. The location of every middlebox may be optimized, but is subject to the constraint that the route from s to t should always go via the waypoint.

Firewall and NIDS. In order to demonstrate how a firewall may be circumvented by hosts (or switches), we set up Mininet and ONOS as shown in Figure 7. The switches do not have flow rules for $k1$ and $k2$ to communicate. The firewall $fw1$ prevents hosts on the left to communicate with hosts on the right and vice-versa. ONOS has the *Intent Reactive Forwarding (ifwd)* application enabled. *ifwd* uses the reactive “pave-path technique” (discussed above) to install flows in the switches. By default, the *ifwd* application establishes host-to-host connectivity when it receives a *Packet-in* for which no flows exist.

We send a ping packet from $k1$ to $k2$. Despite the presence of the firewall, $k1$ receives the reply from $k2$ using out-of-band forwarding teleportation. In the absence of out-of-band forwarding teleportation, the packet would have been dropped by $fw1$.

Indeed, in this case, out-of-band forwarding teleportation has the side effect of installing flows on $s1$ and $s2$ for $k1$ and $k2$ to communicate, preventing further out-of-band forwarding teleportation. By masquerading its MAC address, $k1$

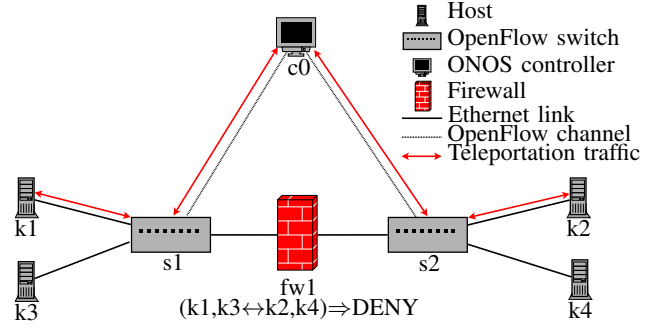


Figure 7: An SDN topology with OpenFlow switches $s1$ and $s2$ and an OpenFlow controller $c0$ (ONOS). $k1$ and $k3$ are connected to $s1$ while $k2$ and $k4$ are connected to $s2$. $s1$ and $s2$ are separated by a firewall $fw1$ that denies hosts on $s1$ to communicate with hosts on $s2$ and vice-versa. $k1$ can use *out-of-band forwarding teleportation* to transfer data to $k2$, bypassing $fw1$.

can teleport more data to $k2$ via out-of-band forwarding teleportation.

Similar to the firewall scenario, we can also use out-of-band forwarding teleportation in the presence of *Snort*, an NIDS. In particular, we can generate attack traffic using nmap to conduct TCP flag attacks or even port scans. Indeed, by masquerading the source MAC address, one can effectively carry out a wide enough port scan without having the scan pass through the firewall and being detected by the latter.

By replacing the firewall we previously described with *Snort*, we use nmap from $k1$ to carry out a TCP port scan on $k2$ using out-of-band forwarding teleportation. By inspecting the alerts in *Snort* we verified that no alerts were generated for the port scan.

Note that the host-to-host connectivity setup involves a *Packet-in* and *Flow-mod* messages whereas the out-of-band forwarding teleportation only involves *Packet-in* and *Packet-out* messages with the side effect of *Flow-mod* messages. Therefore, security policy enforcers that do not inspect and correlate *Packet-in* with *Packet-outs*, will miss out-of-band forwarding teleportation based attacks. Of course, violating *Flow-mods* may eventually be detected, but only after the data has been teleported.

6.3. Rendezvous and Malicious Switch Discovery

We next consider a rendezvous protocol in which malicious switches wish to discover one another. A rendezvous or discovery protocol can be also seen as a precursor to a much more damaging attack such as a denial-of-service, man-in-the-middle (mitm) or exfiltration.

Teleportation can be an attractive solution: A rendezvous protocol can rely on steganography, i.e., embedding patterns in teleported benign information or modulating patterns in legitimate messages. Without teleportation, by going through the data plane directly, the malicious switches risk detection.

² <https://goo.gl/FN9ULQ>

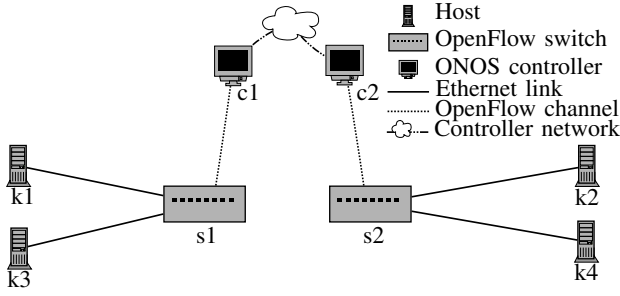


Figure 9: An SDN topology with independent OpenFlow switches controlled by independent OpenFlow controllers (ONOS). $c1$ and $c2$ share and synchronize state information via an independent controller network. $s1$ is controlled by $c1$ and $s2$ is controlled by $c2$ respectively.

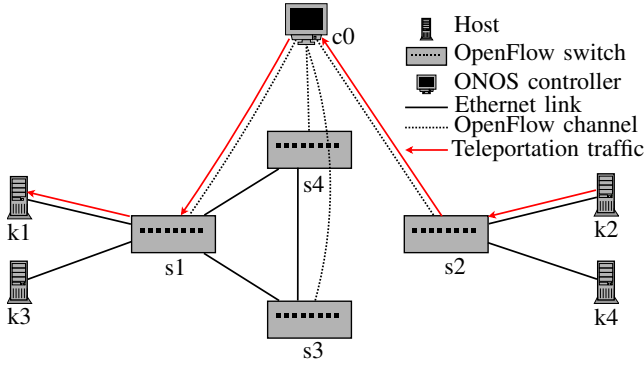


Figure 10: An SDN topology with OpenFlow switches $s1$, $s2$, $s3$ and $s4$ and an OpenFlow controller $c0$ (ONOS). $k1$ and $k3$ are connected to $s1$ while $k2$ and $k4$ are connected to $s2$. Note that $s2$ is not connected to the other switches, and thereby is isolated in the data plane. $k2$ can still exfiltrate data to $k1$ using *out-of-band forwarding teleportation* circumventing the data plane isolation.

a controller that handles other similar remote locations. We show that in such a network, not only malicious switches can exfiltrate data using out-of-band forwarding teleportation but even malicious hosts.

We set up Mininet and ONOS as shown in Figure 10. ONOS has the *ifwd* application activated. By showing how $k2$ can exfiltrate data to $k1$, we also demonstrate how $s2$ can exfiltrate data to $k1$ or $s1$.

Given that $s1$ and $s2$ do not have flow rules for traffic from $k2$ to $k1$ (as they are located in disconnected data planes), $k2$ can exfiltrate data to $k1$ by simply sending a packet (e.g., UDP packet) to $k1$ thereby exploiting out-of-band forwarding teleportation. The controller will receive the packet from $s2$ and send it to $s1$ which will then forward the packet to $k1$.

By inspecting the OpenFlow channels, we can see the out-of-band forwarding teleportation, first as a *Packet-in* and then as a *Packet-out*.

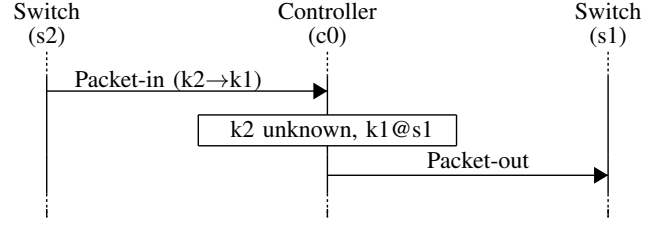


Figure 11: The message sequence pattern for evading policy conflicts using out-of-band forwarding teleportation. The side effect of *Flow-mod* messages are avoided when Jumbo frames are used from a masqueraded MAC address; only *Packet-ins* and *Packet-outs* are used.

6.5. Evading Policy Conflicts

For an attacker, remaining stealthy is key to persistent existence. One of the side effects of using the out-of-band forwarding teleportation is the *Flow-mod* messages issued by the controller. The *Flow-mod* messages may generate policy conflicts (unauthorized/conflicting flow rules), alerting the administrator. A stealthier version of using the out-of-band forwarding teleportation would be to prevent the *Flow-mod* side effect. This would not only prevent policy conflicts, but also leave minimal traces on the source and sink switches.

In order to demonstrate this attack, we set up Mininet and ONOS with *ifwd* activated as shown in Figure 10. $k2$ can exfiltrate data to $k1$ using out-of-band forwarding teleportation without triggering *Flow-mod*'s on $s2$ and $s1$ by masquerading its source MAC address and *ETHER_TYPE* (e.g., Jumbo frame: 0x8870).

If the packet processor and intent framework cannot correctly identify a packet, their behavior may violate security policies. Note that it is enough if the *ETHER_TYPE* is set to a value that ONOS does not recognize, and we are not restricted to Jumbo frames only. The message sequence pattern for out-of-band forwarding teleportation without the *Flow-mod* side effect is shown in Figure 11.

By inspecting the OpenFlow channels, we can verify that the packet was indeed teleported via out-of-band forwarding teleportation first as a *Packet-in* and then as a *Packet-out*. By inspecting the flows on the switches, we can verify that no new flows are present.

Remark on a Denial-of-service Attack. Interestingly, we observed that a side effect of our out-of-band forwarding teleportation is a novel denial-of-service attack. If in our evading policy conflicts example, the host sends the same packet (Jumbo frame) again, then *ifwd* encounters a null-pointer exception and disconnects the switch that sent it the packet. This shows how a malicious host can cause the switch it is connected to, to be disconnected from the controller even when a packet it sends is not corrupted.

We emphasize that this is a side effect of out-of-band forwarding teleportation only, and not a teleportation issue in itself. Fortunately, the issue has been resolved by the

ONOS community after we contacted them (published as CVE-2015-7516).

6.6. Man-In-The-Middle

While we have so far focused on attacks where either only switches or only hosts are malicious, we now detail an attack that involves a malicious switch and a malicious host. The damage of such a collaboration can be severe, for example, the attackers could serve benign hosts with malicious web pages. In order to exemplify the attack we use HTTP rather than HTTPS.

For this attack, we set up Mininet and ONOS with *ifwd* activated as shown in Figure 12. *s1* and *k2* are both malicious while the others are not. *k3* is a benign web server. *s1* teleports specific HTTP traffic towards *k2*. *k2* modifies the HTTP traffic and teleports it back to *s1* who then forwards it to *k1*. In order to emulate the malicious switch, we introduced a flow rule (shown in Listing 1) that rewrites the destination MAC address for TCP traffic with PSH and ACK flags sent from *k3* to *k1*, to *k2*. This modified packet is then passed through the flow table lookup again by using the *resubmit* action in Open vSwitch. *k2* runs *ettercap* to modify the TCP/HTTP payload and forwards the packet to the correct destination. Specifically, we created an *ettercap* filter that looks inside HTTP responses from *k3* for the word “good”, replaces it with “evil”, and sends it to *k1*. The firewall *fw1* is meant to block traffic between hosts on the right and the left.

When *k1* requests the *index.html* page from *k3*, based on the flow rule installed on *s1*, only HTTP responses from *k3* are teleported to *s2* and forwarded to *k2*, through the out-of-band forwarding teleportation. Subsequently, *k2* modifies only the *index.html* web page and has *s2* teleport it back to *s1* via out-of-band forwarding teleportation. Indeed, the side effect is *Flow-mod* messages to *s1* and *s2*.

By viewing the *index.html* file received at *k1* we verified that the mitm attack was successful. The benign and malicious web pages are shown in Listing 2 and Listing 3 respectively. By inspecting the flow counters on the switches we verified that necessary packets did not pass through the data plane.

Note that we did not introduce code into the Open vSwitches to handle the mitm, therefore once the flows are installed on the switches, the firewall will block all traffic between *s1* and *s2* and vice-versa.

7. Out-of-Band Forwarding Performance

Having identified and demonstrated the various attacks in this section we describe our evaluation of the *out-of-band forwarding* channel. In particular we measure the throughput, jitter and packet loss of the channel, and the resource footprint of this channel in terms of CPU usage and memory consumption at the controller.

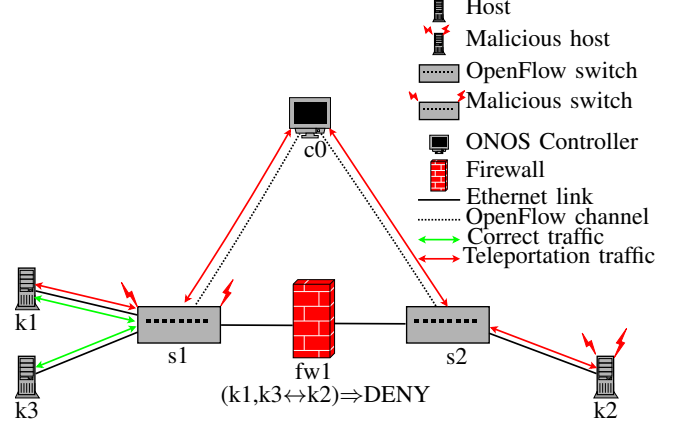


Figure 12: An SDN topology with OpenFlow switches *s1* and *s2* with *c0* the OpenFlow controller (ONOS). *k1* and *k3* are connected to *s1* while *k2* is connected to *s2*. *fw1* denies *k2* to communicate with *k1* and *k3* and vice-versa via the data plane. *s1* and *s2* being malicious, exploit the out-of-band forwarding teleportation to eavesdrop and modify communication data between *k1* and *k3* bypassing *fw1*.

```
priority=50001,tcp,in_port=2,
dl_src=00:00:00:00:00:03,
dl_dst=00:00:00:00:00:01,
tp_src=80,tcp_flags+=psh+ack
actions=mod_dl_dst:00:00:00:00:00:02,
resubmit:0
```

Listing 1: An Open vSwitch flow rule that was introduced into the malicious switch (*s1*) to teleport HTTP traffic with the PSH and ACK flags to the benign switch *s2*. The matching packets have the destination MAC address modified and resubmitted to the flow-table lookup which results in *Out-of-Band Forwarding* teleportation.

7.1. Setup

In order to measure the throughput, jitter and packet loss of the out-of-band forwarding channel, we set up three dedicated systems: one system (64 bit Intel Core i7-3517U CPU @ 1.90 GHz with 4GB of RAM) running ONOS-1.5, another system (Intel Core 2 CPU @ 2.13 GHz with 4GB of RAM) running Mininet for the switches, and a third system (Intel Core i5-5200U CPU @ 2.20GHz with 16GB of RAM) running OFCProbe [29] for load generation. Only the three systems are networked together via a Netgear 100Mbps switch. On the Mininet system, we use a simple line topology consisting of two hosts and two switches, where, host1 is connected to switch1 which is connected to switch2; switch2 in turn is connected to host2. The switches accordingly connect to ONOS as their controller.

7.2. Methodology

In order to emulate the malicious switch, we simply install a flow rule on switch1 with the highest priority so that

```

root@Mininet-vm:~# curl http://10.0.0.2
<html>
<head>
<title>Welcome page</title>
<body>
good
</body>
</html>

```

Listing 2: HTML code from the benign web server. Note the word “good” is present in the body of the HTML code.

```

root@Mininet-vm:~# curl http://10.0.0.2
<html>
<head>
<title>Welcome page</title>
<body>
evil
</body>
</html>

```

Listing 3: HTML code modified by the malicious switch *s1* and host *k2*. Note the word “evil” is present in the body of the HTML code.

the out-of-band forwarding applies by default, i.e., Packet-Ins are sent to ONOS and forwarded accordingly.

We measure the throughput, jitter and packet loss using iPerf3 running on the hosts in Mininet using UDP packets. We consider UDP packets with a payload of 512 bytes to be teleported. Note that for the 512 bytes to be teleported, the overhead in bytes for encapsulation (in the following order: Ethernet, IP, TCP, OpenFlow, Ethernet, IP, UDP) is 110 bytes (for a Packet-in) and 108 bytes (for a Packet-out). Therefore, a 10 Mbps teleportation channel corresponds to approximately 2009 packets (Packet-ins) per second. For the CPU and memory usage on the controller, we use *taskset* to pin ONOS to a single CPU and use *top* to measure the CPU and memory usage. For the load generation: OFCProbe, emulates 20 switches that trigger Packet-Ins to the controller following a Poisson distribution ($\lambda=1$). The throughput, jitter, packet-loss, CPU and memory usage is sampled every second for 600 seconds.

7.3. Evaluation

We first study the throughput of the UDP-based teleportation channel, then consider the packet loss and jitter characteristics, and finally examine the resource footprint in terms of CPU and memory in turn.

7.3.1. Throughput. In Fig. 13 we visualize the throughput of the teleportation channel as box plots. In Fig. 14 and 15, we visualize the throughput of the teleportation channel without and with load resp. as scatter plots.

We first observe that the teleportation channel can indeed sustain very high transmission rates (Tx), of up to 40Mbps in both scenarios. In the scenario without load (Fig. 13

and 14), we see that the channel becomes saturated around rates slightly higher than 60Mbps, after which the throughput suffers. In the scenario with load (Fig. 13 and 15), the variance of the throughput is naturally higher, but nevertheless it can sustain rates which are almost as high as without load.

In conclusion, our results show that the performance of teleportation can go far beyond a small number of packets per second, which underlines the relevance (and potential threat) of such channels.

7.3.2. Packet Loss and Jitter. Fig. 16 shows the packet loss for the scenario without load and with load, respectively. The experiments confirm the quality of the considered teleportation channel: Up to 40 Mbps, the packet loss is small despite some variance, and naturally increases beyond 10% for rates more than 50 Mbps. Again, only beyond the critical rates of slightly more than 50 Mbps packet loss becomes significant. Indeed, we can see a direct correlation between the packet loss and the drop in throughput.

We plot the jitter without load resp. with load in Fig. 17³. Also in terms of this metric, we can see that the teleportation channel offers a good quality also for high rates. The load on the controller again introduces some variance to the jitter, however, it does not influence the median value by much.

7.3.3. Resource Footprint. To better understand the resource requirements of the teleportation channel, as well as the reasons behind the throughput drop at high rates, we measured the CPU load and memory footprint on the controller.

Fig. 18 visualizes the CPU usage as a box plot, while Fig. 19 and 20 visualizes the CPU loads over time. We observe that for a 10 Mbps channel, the CPU utilization has a median value of 55, which is fairly high, but not alarming. We also observe that at rates around 20 Mbps, the additional CPU load introduced for an extra 10 Mbps (20 Mbps vs 30 Mbps channels) is small. The influence of the load on the controller is discernible by the variance introduced and a slight increase in the utilization. However, again at around 50 Mbps, the effects become larger: We can clearly see the relationship between the throughput and CPU load, and when the CPU consumption begins to climb, the throughput begins to drop. Indeed, for transmission rates beyond 50 Mbps, the CPU utilization is so high that it can easily be detected. This is also the time around which the jitter tends to increase by a small amount.

With respect to the memory consumption, Fig. 21 shows that between 10 and 50 Mbps the memory consumption is within a close range (13-15 MB) regardless of whether the load is induced or not. For 60Mbps and above, the memory consumption is higher. Nonetheless, the impact of teleportation on the memory is negligible.

³ Due to noise in our measurement setup, we obtained some outliers in the jitter experiments. Therefore, we followed the median absolute deviation [30] method, with a tolerance of 3.5 to remove such outliers.

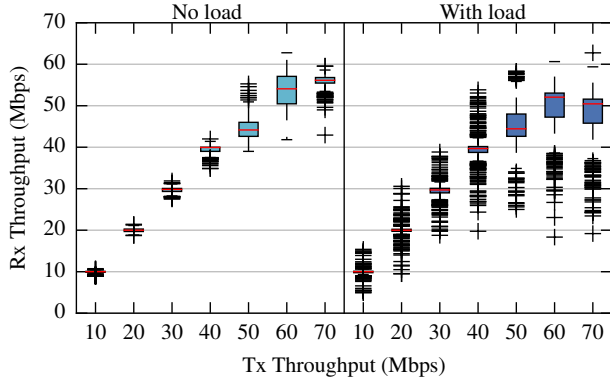


Figure 13: Received throughput using *Out-of-Band Forwarding* without and with load on the controller.

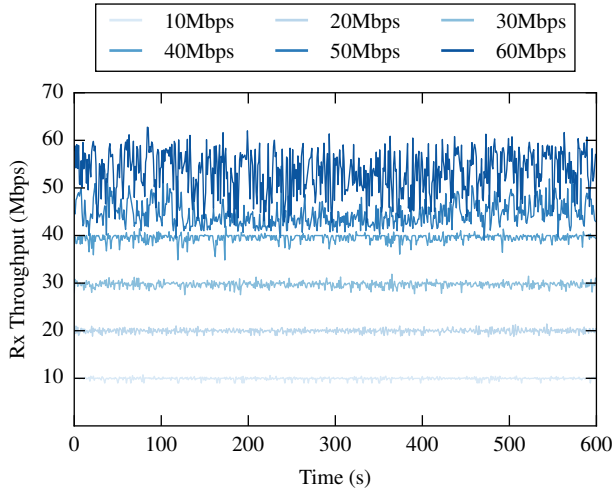


Figure 14: Received throughput using *Out-of-Band Forwarding* without load on the controller.

7.4. Summary

Our first experiments show that teleportation channels in the order of 10 Mbps are feasible, providing low packet loss and low jitter. Moreover, these channels introduce a moderate resource overhead in terms of CPU and a low overhead in terms of memory. Hence, such traffic may go unnoticed given the normal traffic patterns (even if the regular traffic rate is orders of magnitude smaller). However, we also observe that beyond a certain teleportation rate, the CPU load will increase and become the bottleneck for teleportation, limiting the throughput, introducing high packet loss rates, and jitter. Therefore, we expect a sophisticated attacker to target teleportation rates resulting in resource footprints which are obfuscated by the regular load.

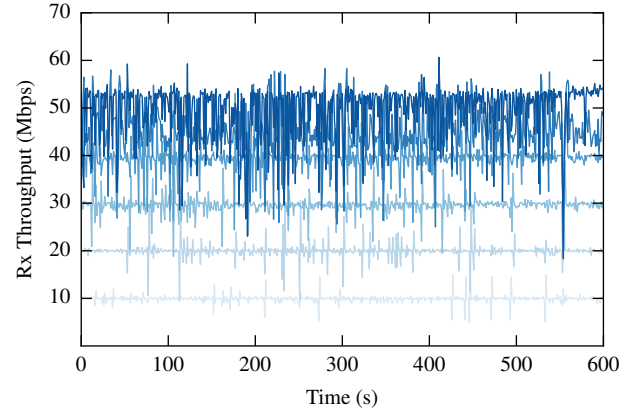


Figure 15: Received throughput using *Out-of-Band Forwarding* with load on the controller. Legend as in Fig. 14

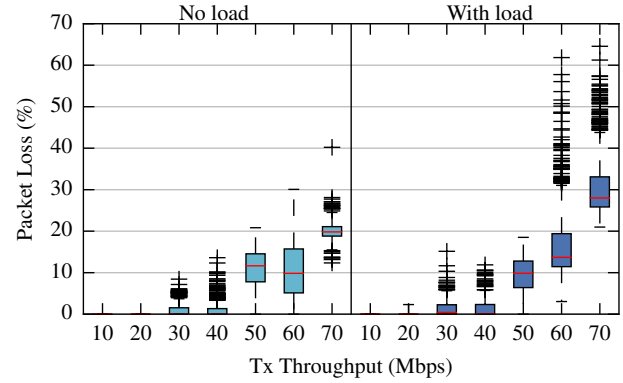


Figure 16: Received packet loss using *Out-of-Band Forwarding* without and with no load on the controller.

8. Countermeasures

Having showcased a variety of attacks using teleportation, we now start exploring possible countermeasures. Although we have demonstrated all the attacks using ONOS we believe that these issues are likely to become more general in nature. They are becoming important with the shift towards automated and intent aware controller frameworks allowing for simpler and agnostic controller applications. Based on our experiments we have also seen that the resources required and utilized for teleportation, even at high rates are moderate. Therefore, it may be difficult to distinguish the attack traffic from the benign traffic. Accordingly we believe that, with the separation of the control and data plane, it is now important to monitor and police the communication channel between the separated planes due to the increased attack surface [13].

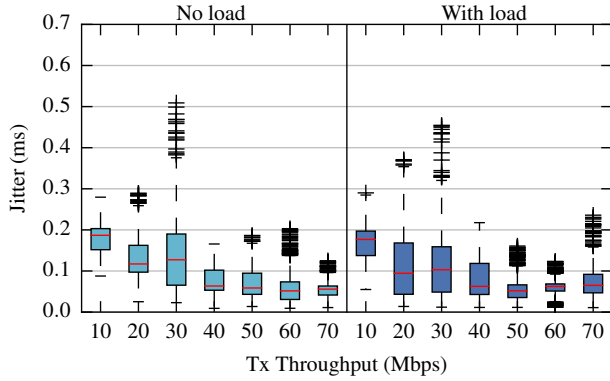


Figure 17: Received jitter using *Out-of-Band Forwarding* without and with no load on the controller.

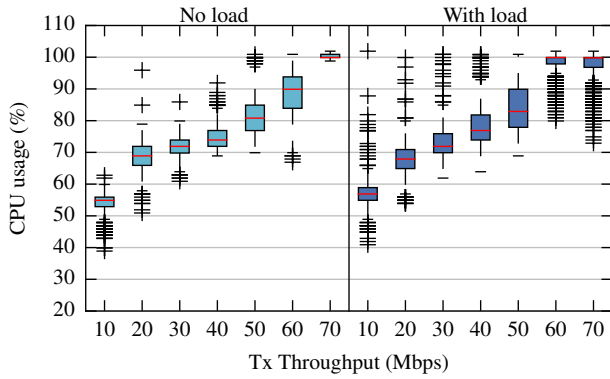


Figure 18: CPU load using *Out-of-Band Forwarding* without and with no load on the controller.

8.1. Packet-in-Packet-out Watcher

In order to prevent the out-of-band forwarding teleportation, we strongly advise the use of a *Packet-in* and *Packet-out* watcher. It can either exist as a controller application or as an application that resides between the controller and switches akin to hypervisors. It would involve tracking and enforcing security policies for *Packet-ins* and their corresponding *Packet-outs*. Existing security enforcement kernels, hypervisors and security applications must account for *Packet-ins* and *Packet-outs* in addition to *Flow-mods* to detect and prevent out-of-band forwarding teleportation.

Note that the out-of-band forwarding teleportation could also be used by malicious controller applications. In a non-adversarial scenario, the order in which a packet's fate is decided upon by various applications can inadvertently teleport the packet. Therefore, verifying that the *Packet-out* does not reach an undesired switch/host can prevent out-of-band forwarding teleportation.

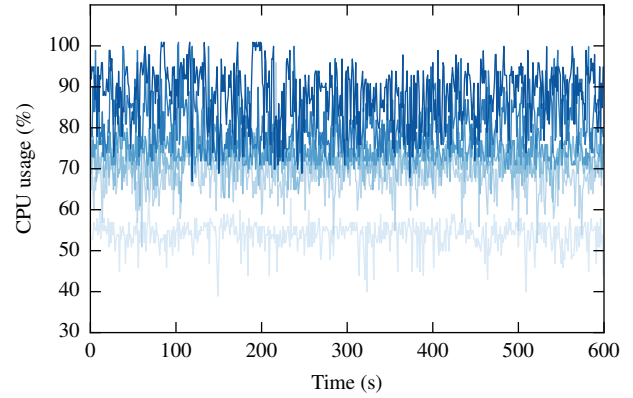


Figure 19: CPU load using *Out-of-Band Forwarding* without load on the controller. Legend as in Fig. 14

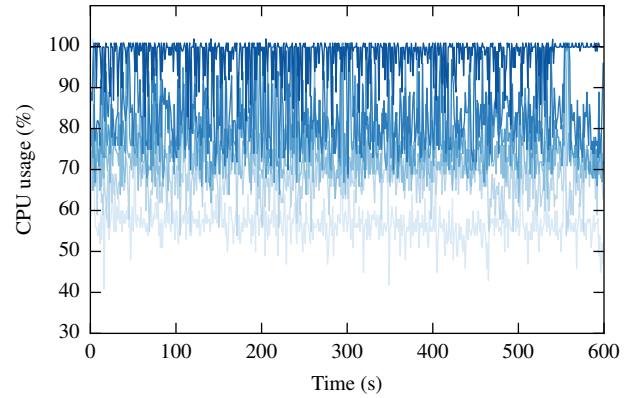


Figure 20: CPU load using *Out-of-Band Forwarding* with load on the controller. Legend as in Fig. 14

8.2. Audit-Trails and Accountability

We propose controllers to introduce secure audit-trail capabilities, and accounting, that enable network administrators to thoroughly investigate events in their networks. For example, controllers must log and alert sensitive events such as a moving MAC addresses, or, receiving a *Packet-in* when a flow has not yet timed out. Such capabilities can aid detection and prevention mechanisms. It is also useful for investigating security incidents. We recommend administrators to frequently view controller logs, investigate failed events and suspicious identities in the network.

8.3. Enhanced IDS with Waypoint Enforcement

Network intrusion detection systems are an important means to detect and limit cyber attacks today, and accordingly intrusion detection systems constitute an integral part of most networks. We strongly suggest the use of an IDS application on top of or before the controller, that can inspect *Packet-ins* and *Packet-outs* and alert on suspicious traffic. Indeed,

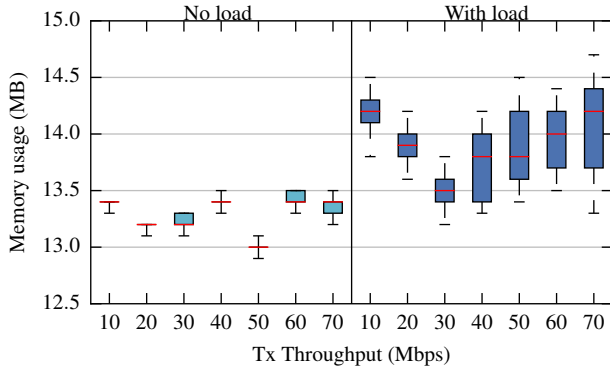


Figure 21: Memory usage using *Out-of-Band Forwarding* without and with no load on the controller.

some controllers today already offer basic functionality for waypoint enforcement. In particular, we suggest waypoint enforcement and coordinating intrusion detection systems from the control plane with the data plane. This is non-trivial, but vital for network security.

9. Related Work

While researchers have already pointed out several interesting novel challenges in providing a correct operation of networks with separate data and control planes [7], [8], [31], it is generally believed that SDN has the potential to render computer networking more verifiable [5], [6] and even secure [32], [9], [10], [11], [12].

Only recently researchers have started discovering security threats in SDN. Klöti et al. [33] report on a STRIDE threat analysis of OpenFlow, and demonstrate data plane resource consumption attacks. Kreutz et al. [34] survey several threat vectors that may enable the exploitation of SDN vulnerabilities. Benton et al. [35] analyze vulnerabilities in OpenFlow. In particular they point out the lack of TLS adoption/implementation in OpenFlow switches and controllers. In addition, they correctly identify the possibility of dos attacks on the centralized control plane. Another key challenge arising from the separation of the control and data planes, is the potential loss of network visibility. It has been shown that the network view of the controller may even be poisoned [27], [28]. Thimmaraju et al. [13], point out that threat models for the virtualized data plane need to account for a malicious/compromised data plane in SDNs, and cloud operating systems such as OpenStack.

While much research went into designing more robust and secure control planes [36], [37], less published work exists on the issue of malicious switches. A notable exception is the work by Antikainen et al. [38], who consider the possibility of a malicious relay node for a man-in-the-middle attack. Interestingly, in our paper, we have shown that the relay node can be the benign controller itself.

To the best of our knowledge, our work is the first to point out and characterize the fundamental problem of SDN

teleportation. More generally, while most prior studies about malicious switches focus on (indirect) *attacks targeting the controller*, we in this paper demonstrate new kinds of attacks which merely exploit the controller for *directly* attacking (e.g., the confidentiality or availability) of network services.

However, there are a number of interesting approaches proposed in the literature which have implications for our scenarios as well. For example, the pre- and post-conditions of Topoguard [28] can defend against our path update attack. However, if the switches are malicious, these conditions can be spoofed by the malicious switches. Also, Topoguard cannot detect teleportation using path reset, switch identification and out-of-band forwarding teleportation.

Sphinx [27] can alert on the path update teleportation. However, it cannot detect the path reset as the flow graph remains the same. Additionally, Sphinx assumes that switches cannot use the same DPIDs, therefore, we believe that our switch identification teleportation will not be detected by Sphinx. Also, our out-of-band forwarding relies on *Packet-in* and *Packet-out* messages, while *Packet-outs* are not considered by Sphinx⁴. Therefore the suggested out-of-band forwarding teleportation can evade Sphinx, until topology altering flows are installed.

Porras et al. [39] propose a security mediator that comprises of Rule Conflict Analysis, Role-based Source Authentication, State Table Manager and a Permission Mediator. We admit that the path update can be detected using this approach, however, our path reset does not introduce any conflicting rules. The *Features-reply* messages are not a part of their solution, therefore, we believe that *switch identification* teleportation can succeed. With respect to out-of-band forwarding teleportation, unless the mediator investigates the destination switch or MAC address in the *Packet-out*, the teleportation can bypass the security mediator given sufficient permissions.

SDN Hypervisors such as CoVisor [40], Flowvisor [41], FortNOX [10] depend on policies maintained in the hypervisor. Therefore, we believe that all our teleportation mechanisms hold unless a specific policy blocks it.

Dover Networks [42] discovered the behavior of Floodlight with switches using the same DPID, which we exploit for teleportation.

While *Security-Mode ONOS* [43] can enhance the security in many scenarios, by introducing roles and permissions, at least today, it does not help against teleportation: Once *ifwd* has the permission to write intents and emit packets, our teleportation succeeds. These permissions are bare necessities for *ifwd* to function.

10. Conclusions

As OpenFlow networks transition from research to production, new levels of reliability and performance are necessary [44]. This paper has identified and demonstrated a novel security threat introduced by software-defined networks separating the control plane from the data plane. In the

⁴ Unfortunately, the source code of Sphinx is not available.

TABLE 2: Summary of teleportation attacks and involved entities.

Attack	Teleportation technique	Exploited by
Bypass Firewall	Out-of-band forwarding	Switch and Host
Bypass NIDS	Out-of-band forwarding	Switch and Host
Exfiltration	Out-of-band forwarding	Switch and Host
Evading policy conflicts	Out-of-band forwarding	Switch and Host
Man-in-the-middle	Out-of-band forwarding	Switch and Host
Rendezvous	Path update	Switch
	Path reset	Switch
	Switch identification	Switch

presence of an unreliable south-bound interface (containing malicious switches): We have shown that state-of-the-art controller(s) are vulnerable to teleportation. Teleportation has numerous applications (cf. the summary in Table 2): It can be exploited to bypass security-critical network elements (e.g., to exfiltrate confidential information), as a discovery protocol for malicious switches, to evade policy conflicts as well as for man-in-the-middle attacks. Based on our preliminary evaluation, we can say that even a teleportation channel of over 10 Mbps can easily be used inside a loaded control channel.

Our work can also be seen as a first security analysis of the increasingly popular intent-based network mechanisms: While intent-based mechanisms are attractive for allowing (cloud) network operators resp. SDN applications to focus on “what to connect” rather than “how”, we have shown that controller managed intents need to be used with care. Indeed, our experiments with controllers that are only starting to introduce an intent based mechanism are not yet vulnerable to all the specific attacks presented in this paper. Moreover, while intent mechanism implementations can vary across controllers, we believe that the underlying issues are fundamental.

We understand our work as a first step, and believe that our paper opens several relevant directions for future research. In particular, we plan to extend our vulnerability analysis to other SDN protocols and conduct a more in-depth performance analysis. Another relevant avenue for future research regards the development of countermeasures.

Acknowledgments

Research (partially) supported by the Danish Villum project *ReNet*. We would like to thank the anonymous reviewers of Euro S&P’17 for their valuable feedback and helpful comments during the review process. We would also like to thank Prof. Jean-Pierre Seifert and Tobias Fiebig for their constructive inputs at the onset of this paper.

References

[1] Internet Science Working Group, “Internet as a critical infrastructure: Security, resilience and dependability aspects (JRA7),” 2015, accessed: 2017-02-06. [Online].

Available: <http://www.internet-science.eu/groups/internet-critical-infrastructure-security-resilience-and-dependability-aspects>

[2] M. Armbrust et al., “A view of cloud computing,” *Communications of the ACM*, pp. 50–58, April 2010.

[3] T. Anderson, L. Peterson, S. Shenker, and J. Turner, “Overcoming the internet impasse through virtualization,” *IEEE Computer*, vol. 38, no. 4, pp. 34–41, April 2005.

[4] P. v. Oorschot, T. Wan, and E. Kranakis, “On interdomain routing security and pretty secure bgp (psbgp),” *ACM Trans. on Information and System Security (TISSEC)*, no. 3, July 2007.

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

[6] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, “Veriflow: Verifying network-wide invariants in real time,” in *Proc. NSDI*, 2013, pp. 467–472.

[7] O. Padon, N. Immerman, A. Karbyshev, O. Lahav, M. Sagiv, and S. Shoham, “Decentralizing sdn policies,” in *Proc. ACM Principles of Programming Languages (POPL)*, 2015.

[8] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, “Consistent updates for software-defined networks: Change you can believe in!” in *Proc. ACM Workshop on Hot Topics in Networks (HotNETs)*, 2011, pp. 7:1–7:6.

[9] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, “Netlord: A scalable multi-tenant network architecture for virtualized datacenters,” in *Proc. ACM SIGCOMM*, 2011, pp. 62–73.

[10] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, “A security enforcement kernel for OpenFlow networks,” in *Proc. ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2012, pp. 121–126.

[11] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, “Fresco: Modular composable security services for software-defined networks,” in *Proc. NDSS*, 2013.

[12] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, “AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks,” in *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2013, pp. 413–424.

[13] K. Thimmaraju et al., “Reins to the cloud: Compromising cloud systems via the data plane,” *arXiv preprint arXiv:1610.08717*, 2016.

[14] “Snowden: The NSA planted backdoors in Cisco products,” 2014, accessed: 02-01-2018. [Online]. Available: <http://www.infoworld.com/article/2608141/internet-privacy/snowden--the-nsa-planted--backdoors-in-cisco-products.html>

[15] “The tale of one thousand and one dsl modems,” 2012, accessed: 2017-02-06. [Online]. Available: <https://securelist.com/analysis/publications/57776/the-tale-of-one-thousand-and-one-dsl-modems/>

[16] “Synful knock - a cisco router implant - part i,” 2015, accessed: 2017-02-06. [Online]. Available: https://www.fireeye.com/blog/threat-research/2015/09/synful_knock_-_acis.html

[17] F. Lindner, “Cisco IOS router exploitation,” Recurity Labs, Tech. Rep., 2009, accessed: 2017-02-06. [Online]. Available: <http://www.blackhat.com/presentations/bh-usa-09/LINDNER/BHUSA09-Lindner-RouterExploit-PAPER.pdf>

[18] S. Lee, T. Wong, and H. S. Kim, “Secure split assignment trajectory sampling: A malicious router detection system,” in *Proc. IEEE/IFIP Transactions on Dependable and Secure Computing (DSN)*, 2006, pp. 333–342.

[19] “Huawei hg8245 backdoor and remote access,” 2013, accessed: 2017-02-06. [Online]. Available: <http://websec.ca/advisories/view/Huawei-web-backdoor-and-remote-access>

[20] “Netis routers leave wide open backdoor,” 2014, accessed: 2017-02-06. [Online]. Available: <http://blog.trendmicro.com/trendlabs-security-intelligence/netis-routers-leave-wide-open-backdoor/>

- [21] S. Fagerland, W. Grange, and Blue Coat Systems Inc., “The inception framework: Cloud-hosted APT,” 2014, accessed: 2017-02-06. [Online]. Available: http://dc.bluecoat.com/Inception_Framework
- [22] “ONOS wiki home,” <https://wiki.onosproject.org/display/ONOS/Wiki+Home>, 2017, accessed: 02-01-2018.
- [23] ONOS, “Security advisories,” 2015, accessed: 2017-02-06. [Online]. Available: <https://wiki.onosproject.org/display/ONOS/Security+advisories>
- [24] *OpenFlow Switch Specification*, Open Networking Foundation, 2013, version 1.3.2 Wire Protocol 0x04.
- [25] “Send a raw ethernet frame in linux.” 2011, accessed: 2017-02-06. [Online]. Available: <https://gist.github.com/austinmarton/1922600>
- [26] N. Kang, Z. Liu, J. Rexford, and D. Walker, “Optimizing the “one big switch” abstraction in software-defined networks,” in *Proc. ACM CoNEXT*, 2013.
- [27] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, “Sphinx: Detecting security attacks in software-defined networks,” in *Proc. NDSS*, 2015.
- [28] S. Hong, L. Xu, H. Wang, and G. Gu, “Poisoning network visibility in software-defined networks: New attacks and countermeasures,” in *Proc. NDSS*, 2015.
- [29] M. Jarschel *et al.*, “Ofcprobe: A platform-independent tool for openflow controller analysis,” in *Proc. IEEE International Conference on Communications and Electronics*. IEEE, 2014, pp. 182–187.
- [30] B. Iglewicz and D. C. Hoaglin, *How to detect and handle outliers*. Asq Press, 1993, vol. 16.
- [31] W. Zhou, D. K. Jin, J. Croft, M. Caesar, and P. B. Godfrey, “Enforcing customizable consistency properties in software-defined networks,” in *Proc. NSDI*, 2015, pp. 73–85.
- [32] S. A. Mehdi, J. Khalid, and S. A. Khayam, “Revisiting traffic anomaly detection using software defined networking,” in *Proc. RAID Recent Advances in Intrusion Detection*, 2011, pp. 161–180.
- [33] R. Klöti, V. Kotronis, and P. Smith, “Openflow: A security analysis,” in *Proc. IEEE International Conference on Network Protocols (ICNP)*, Oct 2013, pp. 1–6.
- [34] D. Kreutz, F. M. Ramos, and P. Verissimo, “Towards secure and dependable software-defined networks,” in *Proc. ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013, pp. 55–60.
- [35] K. Benton, L. J. Camp, and C. Small, “Openflow vulnerability assessment,” in *Proc. ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013, pp. 151–152.
- [36] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, “A distributed and robust sdn control plane for transactional network updates,” in *Proc. IEEE INFOCOM*, 2015, pp. 190–198.
- [37] M. Canini, D. Venzano, P. Perešini, D. Kostić, and J. Rexford, “A nice way to test openflow applications,” in *Proc. NSDI*, vol. 12, 2012, pp. 127–140.
- [38] M. Antikainen, T. Aura, and M. Särelä, “Spook in your network: Attacking an sdn with a compromised openflow switch,” in *Proc. Secure IT Systems: Nordic Conf.* Springer International Publishing, 2014, pp. 229–244.
- [39] P. Porras, S. Cheung, M. Fong, K. Skinner, and V. Yegneswaran, “Securing the software-defined network control layer,” in *Proc. NDSS*, 2015.
- [40] X. Jin, J. Gossels, J. Rexford, and D. Walker, “Covisor: A compositional hypervisor for software-defined networks,” in *Proc. NSDI*, 2015.
- [41] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, “Flowvisor: A network virtualization layer,” OpenFlow, Tech. Rep., 2009.
- [42] J. M. Dover, “A denial of service attack against the open floodlight sdn controller,” Dover Networks, Tech. Rep., 2013. [Online]. Available: <http://dovernetworks.com/wp-content/uploads/2013/12/OpenFloodlight-12302013.pdf>
- [43] “Security-Mode ONOS,” 2015, accessed: 2017-02-06. [Online]. Available: <https://wiki.onosproject.org/display/ONOS/Security-Mode+ONOS>
- [44] M. Kuniar, P. Pereni, and D. Kosti, “What you need to know about sdn flow tables,” in *Proc. Passive and Active Measurement (PAM)*, 2015, pp. 347–359.