Link Discovery Attacks in Software-Defined Networks: Topology Poisoning and Impact Analysis

Sonali Sen Baidya and Rattikorn Hewett
Department of Computer Science, Texas Tech University, Lubbock, TX, USA
Email: {Sonali.Sen-Baidya, Rattikorn.Hewett}@ttu.edu

Abstract—Software Defined Networking (SDN) has become a popular technology that offers advantages of programmable and flexible network management over the legacy practice. The centralized SDN controller is an important enabler of these benefits. One of the most crucial tasks of the SDN controller is link discovery as it provides topology of the network essential for the controller to direct or create rule forwarding and routing mechanisms. Much research on SDN security has been studied but only recently that security of OpenFlow link discovery protocols and topology poisoning have been addressed. Existing work includes link fabrication attacks via compromised hosts and defense systems with authentication. This paper discusses SDN link discovery process and its vulnerability to link discovery attacks including new attacks via compromised switches. We present a simple but effective defense mechanism using active ports that can detect both host-based and switch-based link discovery attacks. Finally, the paper presents an analytical and empirical analysis of the impacts of topology attacks on routing. The paper discusses attack details, proposed methods and results of these analyses.

Index Terms—SDN; Software-Defined networking; link discovery; attack; impact analysis

I. Introduction

Software-Defined Networking (SDN) is a networking paradigm that innovates network infrastructures to programmable and flexible network management and administration [1]-[3]. SDN separates the control logic to a centralized controller (in the control plane) that directs the switches/routers (in the data plane) to forward the packets or create and install forwarding rules in their flow tables. This allows complex networks to be centrally managed without the need to deal with distributed low-level network functionalities [2]. In addition, the programmability of SDN eases the modification and reconstruction of important network properties that result in flexible development, and relatively rapid adoption with low processing costs.

To efficiently and effectively manage the network, the controller needs accurate and complete topology information of the entire network. Therefore, *link discovery* (or *topology discovery*) is one of the most critical services of SDN controller [4]. To do this, SDN on *OpenFlow* [1], a communication protocol to interface between the controller and the data plane devices. Unfortunately, the OpenFlow Discovery Protocol (OFDP), the de-facto standard for implementing topology discovery in most SDN controller platforms, has been shown to be insecure [5]. Topology discovery attacks is the subject of our research in this paper.

Much research has studied SDN security both in the data plane that inherits issues from traditional networking and new security issues in the control plane [5]-[9]. However, most do not address the fundamental vulnerabilities of the OpenFlow-based controllers [10]. This paper focuses on the latter, particularly the security issues of the controller's link discovery process. The OpenFlow discovery protocol is known to be vulnerable due to the lack of authentication mechanisms in packet control messages [5]. This leads to security threats and attacks [11]. A successful attack can poison the network topology information, convincing the controller of the falsified view of the network topology. As a result, an attacker can re-route traffic over false network links enabling man-in-the-middle or denial-of-service attacks via compromised machines [12].

A variety of link discovery attacks and counter measures have been researched [2], [4]-[7], [11], [12]. Early attacks include link fabrication attacks through compromised hosts [7]. By impersonating the end-host, an attacker can create spoofed links by injecting fake control packets into the network via one or more compromised hosts causing traffic to be re-directed to the attacker [2], [5], [10]. No existing work discusses attacks via compromised switches even though the idea is relatively simple and natural.

Alharbi *et al* [5] describe detailed mechanisms of these OpenFlow discovery protocol attacks along with empirical impact analysis on routing to verify the possibility of the attack. While this is useful, it would be desirable to have an analytical approach to impact analysis as well. Existing defense mechanisms include a system that automatically detects attacks by observing anomalous network behaviors [6] and by adding extra authentication to the packet control messages using a keyed-hash message authentication code [2], [5]. However, like many intrusion detection techniques, such defense mechanisms rely on the quality of data and can only detect anticipated anomalies.

This paper discusses details of SDN link discovery process and illustrates its vulnerability to link discovery attacks. The contributions of the paper include:

- Identification of new link discovery attacks via compromised switches (or switch-based link discovery attacks).

Manuscript received January 2, 2020; revised July 2, 2020.
Corresponding author email: sonali.sen-baidya@ttu.edu.
doi:10.12720/jcm.15.8.596-606

- A simple defense mechanism using active ports that can detect both switch-based and host-based link discovery attacks.
- An analytical approach to analyze impacts of topology attacks on routing along with an empirical approach to verify consistency of the results.

The rest of the paper is organized as follows: Section II provides the background information about the SDN and Link Discovery process. Section III describes Link discovery attacks including previous host-based link discovery attacks and its defense mechanism in Subsection *A*, the switch-based discovery attacks in Subsection *B*, and the proposed detection technique in Subsection *C*. Section IV presents impact analysis with analytical approach in Subsection *A* and empirical approach in Subsection *B*. Section V discusses related work and Section VI concludes the paper.

II. Background

This section describes an overview of SDN in Section *A* and the link discovery service in Section *B*.

A. Software-Defined Networking (SDN)

Fig. 1 shows SDN's architecture that can be viewed in three planes: data plane, control plane and application plane. By separating the data plane and the control plane, and enabling network programmability with a centralized controller, one can adapt network management to changes (e.g., new configurations, size, or topology) easily. Thus, SDN provides flexible and easily scalable network management as opposed to the traditional networking that ties the functions from the two planes to the same device (e.g., switch) [1].

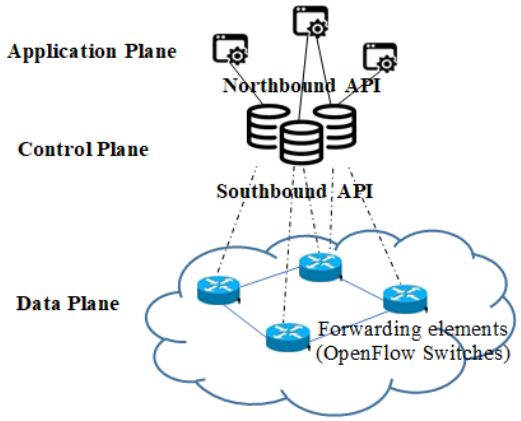


Fig. 1 Architecture of Software-Defined Networking. (SDN)

The data plane deals with hardware level packet processing based on the policies from the control plane. The data plane consists of forwarding devices (e.g., switches, routers) including physical and virtual switches that are responsible for data transmission. Each switch has a corresponding programmable flow table that defines an action for each packet related to a specific flow (called a *flow rule*). When a new packet arrives at a switch, the switch checks if the packet matches any flow rule in the switch's flow table. If so, the packet will be processed according to the existing flow rule. Otherwise, the switch consults with the controller to provide appropriate action (e.g., how to process the particular packet, new flow rule to be installed in the flow table).

The controller in the control plane directs basic network services and operations (e.g., packet routing, traffic monitoring, and network access control) in the data plane. The software controller has a complete view of the network topology, network traffic and status of the switch ports (e.g., active or inactive). Thus, it has the ability to make appropriate routing decisions to improve the network traffic. It exercises the direct control over the data plane through well-defined application programming interfaces (API) based on a logically centralized, abstract view of the network. Thus, the controller is a core SDN's component that can have great influence on the network.

The network is also programmable through software applications situated in the application plane running on top of the control plane. This plane has a set of applications that implement some network control functions like security, routing, load balancers, fault-tolerance, recovery, etc. Thus, in addition to providing core services, the application plane allows other network applications to be implemented as well (e.g., cloud network virtualization and data center network optimization) [2].

The separation of the control plane and the data plane is realized by the Southbound Application Programming Interface (API), as shown in Fig. 1. The most notable of southbound API protocol is *OpenFlow*. It is one of the first standard protocols that is used for managing the communication between the control plane and the data plane. OpenFlow protocol allows the SDN controller to configure switches via the packet forwarding rules. The protocol also allows switches to notify the controller about special events, e.g., the receipt of a packet that does not match any existing forwarding rules. Each OpenFlow switch has one or more flow tables containing the packet-handling rules. These rules direct the OpenFlow switches in forwarding the packets. The network managers use these flow tables to modify the layout of the network and the traffic flows.

Similar to Southbound, the Northbound API represents the interface through which the communication between the application plane and control plane takes place. This Northbound interface abstracts the low-level instruction sets used by southbound interfaces.

B. Link Discovery Service

Topology discovery is a crucial core SDN controller's service since topology information is a fundamental building block for network management (e.g., packet routing, network virtualization and optimization, and mobility tracking). Here we use the terms "topology discovery" and "link discovery" synonymously, as the latter constitutes the former.

Packet sending between switches: Although there is no formal standard for topology discovery in SDN, most

controller platforms implement topology services using OpenFlow Discovery Protocol, which has become the de-facto standard [5]. The OpenFlow Discovery Protocol sends the Link Layer Discovery Protocol (LLDP) packet for link discovery between switches. The LLDP packet has a structure as shown in Fig. 2.

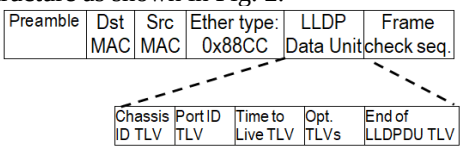


Fig. 2. Structure of LLDP packet.

As shown in Fig. 2, the LLDP Packet includes the LLDP Data Unit, which can be categorized into several *type-length-value* (TLVs) including *Chassis ID* (a sending switch ID), *Port ID* (egress port ID for outgoing packet), and *Time-to-live*. These TLVs are followed by optional TLVs such as *Datapath ID* (DPID), and *End of LLDP Data Unit* TLV.

Packet sending between controller and switches: OpenFlow supports *Packet-In* and *Packet-Out* messages for sending a data packet from SDN switch to the controller and vice versa, respectively. *Packet-Out* message also allows the controller to send, to the switch, additional instructions (or *action list*) on how to forward the data packet. These messages are important for link discovery mechanism to be described below.

Discovering existing switches: The controller realizes the existence of switches and their key properties (e.g., ID, ports, MAC address) through the initial OpenFlow protocol handshake between the switches and the controller. Fig. 3 illustrates message exchanges in the handshake process.

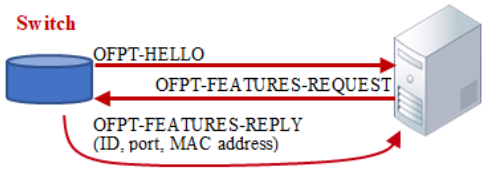


Fig. 3. Handshake protocol.

Upon joining the network, the switch sends OFPT_HELLO to the controller, which in turn sends an OFPT_FEATURES_REQUEST to the switch. The switch then sends OFPT_FEATURES_REPLY along with its key properties (i.e., switch ID, MAC address, active port). The controller keeps the record of each existing switch and their properties.

Fig. 4 illustrates a link discovery from Switch S_1 to Switch S_2. Here the controller is aware of the existence of Switches S_1 and S_2 via initial handshake process when they join the network. These switches and their key properties are stored in a "switch" table on top right corner of the figure. Note that the table does not include active ports connecting with the controller.

SDN link discovery has three basic steps.

Step 1: the controller creates an LLDP packet for each *active* port on each switch recorded by the controller in the table. Each LLDP packet has corresponding *Chassis ID* and *Port ID* TLVs. For example, based on the table in Fig. 4, LLDP PACKET (S_1, P_3) and LLDP PACKET (S_2, P_2) are created. Since we focus on a link from Switch S_1, we only show the LLDP PACKET (S_1, P_3) in the figure.

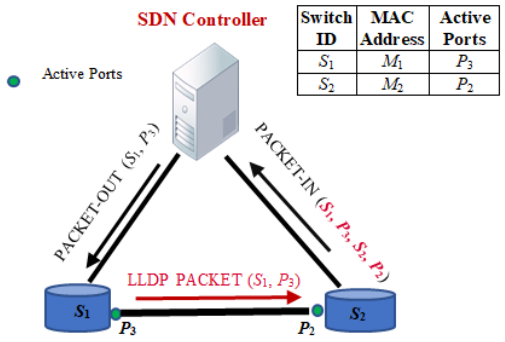


Fig. 4. Link discovery mechanism

Step 2: Based on the LLDP packets created in Step 1, the controller then sends *Packet-Out* message to the switch having the LLDP packet and instructs the switch to forward the LLDP packet through the specified port. As shown in Fig. 4, based on the LLDP PACKET (S_1, P_3) created, the controller sends PACKET-OUT(S_1, P_3) to S_1 and instructs to forward the LLDP packet via Port P_3.

Step 3: Any received LLDP packets must be sent to the controller by *Packet-In* message containing the receiving switch ID and ingress port ID along with meta data of the origin switch sending the LLDP packet (i.e., the Chassis ID and its egress port ID). The evidence of the LLDP packet being forwarded from the sending to receiving switches leads to the controller's conclusion of the existence of the link and thus, link discovery. For example, as shown in Fig. 4, The LLDP PACKET (S_1, P_3,) contains meta data of the origin switch sending the LLDP packet (i.e., Switch S_1 and Port P_3,). Thus, the received Switch S_2 (via Port P_2) sends PACKET-IN (S_1, P_3, S_2, P_2) to the controller. Here the last two parameters S_2, P_2 are meta data of the *receiving* switch, while S_1, P_3 contains data of the *sending* switch. The controller infers its discovery of link from Switch S_1 (Port P_3) to Switch S_2 (Port P_2).

III. Link Discovery Attacks

The described link discovery mechanism is vulnerable in that there is no authentication of LLDP control messages [5]. Thus, any LLDP packet received by the controller is accepted as a genuine packet. Consequently, an attacker can inject fabricated LLDP control messages to poison the topology information of the controller. The attacker can falsify the LLDP packet content or fabricate the link discovery by creating a link that does not actually exist [2]. This section describes link discovery attacks and defense mechanisms for existing host-based attacks in Section *A* and our switch-based attacks in Section *B*.

A. Host-based Link Discovery Attacks & Defense

Most existing studies [5], [6], [10]-[12] of link discovery attacks deal with a situation when a host connecting a switch in a network has been compromised. As an example, consider an extended network of the

network in Fig. 4 where now Host H_1 connects with Switch S_1 (via Port P_4) and Host H_2 connects with Switch S_2 (via Port P_3). Suppose H_1 has been compromised and an attacker aims to create a fake link between switches S_1 (via Port P_4) and S_2 (via Port P_3), which does not exist via these ports.

Link Fabrication: The controller creates LLDP packets for all the active ports for switch S_1, namely LLDP PACKET(S_1, P_3) and LLDP PACKET(S_1, P_4). Then it sends out *packet-out* messages to all active ports, namely PACKET-OUT(S_1, P_3) and PACKET-OUT(S_1, P_4) to forward the corresponding LLDP packets.

Being at H_1, the attacker captures the LLDP PACKET(S_1, P_4) and change the packet into LLDP PACKET(S_2, P_3) and sends the spoofed packet to S_1.

Based on the discovery mechanism (Step 3), when switch S_1 receives the LLDP packet (via Port P_4) it sends a *packet-in* message to the controller, which includes the sender's and receiver's information. Specifically, S_1 sends PACKET-IN (S_2, P_3, S_1, P_4), where the first two parameters are from original when LLDP PACKET(S_2, P_3) being forwarded. The controller infers that there is a link between Switch S_2 (via Port P_3) and Switch S_1 (via Port P_4) when it does not actually exist as desired by the attacker's goal.

Defense Mechanisms: Previous approaches can be divided into two groups: strengthening authentication of LLDP packets by using cryptography [5] and detection techniques for link discovery attacks [6], [7]. We will describe basic ideas of one of the latter approach.

The key element of the detection technique proposed by Hong *et al.* [11] is to realize that in a non-malicious network, the LLDP packet will never be sent from the host. In link discovery process, the controller generates LLDP packets for all active switches to be forwarded through the links. The recipient switch will send the *packet-in* message to the controller to infer link but the recipient host will not as it does not participate in the link discovery process.

However, in the attack case, the compromised host will behave as if it was a switch by sending the spoofed LLDP packet to the switch. The recipient switch carries on and the controller wrongly accepts the spoofed information.

Hong et.al propose *TopoGuard*, a security extension of the controller that raises an alert when it detects a LLDP packet being sent from a host. For example, in the scenario described above, when the *TopoGuard* observes PACKET-IN(S_2, P_3, S_1, P_4) and detects that Port P_4 connects with a host, it raises an alert and stops the discovery update.

B. Switch-based Link Discovery Attacks

In this section, we introduce a *switch-based link discovery attack*, a slightly different link discovery attack where a switch has been compromised (e.g., by flooding the switch's Mac table with fake Mac addresses) and propose a detection technique that can detect both host-based and switch-based attacks. Attacks through compromised switches do not only have the equivalent capabilities as those through malicious hosts but can also have strong impacts and high severity.

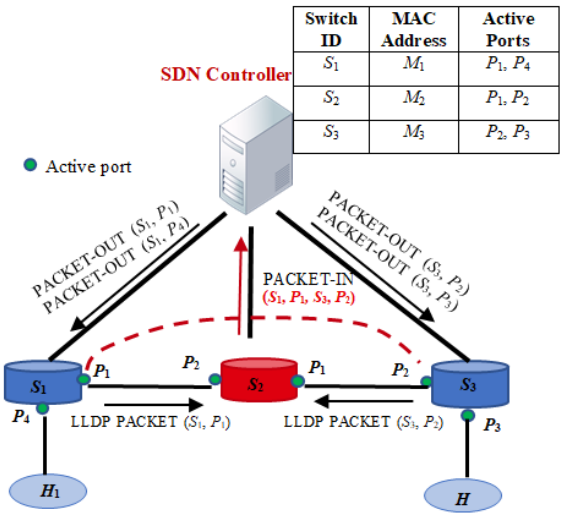


Fig. 5. Switch-based link discovery attack.

The basic idea of switch-based is similar to that of host-based in that once the switch is compromised, an attacker can obtain information to poison the network topology. However, methods for obtaining such information may be slightly different. To illustrate the attack, assume that an attacker has compromised Switch S_2 as shown in red Switch S_2 in Fig. 5. In this scenario, the attacker aims to create a fake link between Switch S_1 and Switch S_3, which does not actually exist. As shown in Fig. 5, in Steps 1 and 2 of the link discovery process, the controller creates LLDP packets for the active ports P_1 and P_4 of Switch S_1 and P_2 and P_3 for Switch S_3, respectively and sends corresponding *packet-out* messages to the respective switches with additional instructions for them to forward the created LLDP packets to adjacent switches. For example, as shown in Fig. 5, the controller sends PACKET-OUT(S_1, P_1) to S_1 that forwards LLDP PACKET(S_1, P_1) to S_2. Similarly, S_3 forwards LLDP PACKET(S_3, P_2) to S_2. Note we omit showing LLDP packets that are not relevant to the attack (e.g., LLDP packets from S_1 and S_3 to hosts H_1 and H_2, respectively).

In Step 3 of the link discovery process, each LLDP packet received must be sent to the controller with *packet-in* message that contains information of the sending and receiving switches. For this example, both LLDP packets (i.e., LLDP PACKET(S_1, P_1) and LLDP PACKET(S_3, P_2)) are received by S_2. Since S_2 has been compromised, the attacker receives and intercepts the LLDP packets and then extracts the MAC address of the switch in the packet (i.e., those of S_1 and S_3). Since MAC addresses are the unique identifier of a switch that the controller uses, capturing MAC addresses will enable the attacker at S_2 to mimic Switches S_1 and S_3. To create a fake link from S_1 to S_3, the attacker (S_2) mimics S_3 as a receiving switch of this fake link and thus, sends a *packet-in* message to the controller, specifically a malicious PACKET-IN(S_1, P_1, S_3, P_2) instead of the correct PACKET-IN(S_1, P_1, S_2, P_2) and PACKET-IN(S_3, P_2, S_2, P_1). As a result, the controller wrongly discovers that there is a link from S_1 (via P_1) to S_3 (via P_2). Note

that the attack needs to forge the controller. Hence, if the malicious switch simply forwards LLDP packets across its ingress/egress ports (e.g., P_2 and P_1 in S_2, to the adjacent switches S_1 and S_3 in this case), instead of sending a corrupted LLDP packet to the controller, the controller would not have had the view of the changed network and the attack would not have been accomplished (i.e., the controller does not discover wrongly a fake link from S_1 (via P_1) to S_3 (via P_2)).

C. Proposed Defense Mechanism

Our defense mechanism is for the controller to validate the link discovered (i.e., those sent via the packet-in messages as well as attacks via compromised switch and host) and alert a suspicious link instead of accepting any link discovered. The principle behind the link validation criterion is that no active port should be used to connect more than one link at a time. By monitoring all active ports of every switch and every active link in the network, our defense mechanism makes sure that no active port can be shared for connecting multiple links at the same time. If a new discovered link sent (via *packet-in* message) violates this principle, then the defense mechanism declares the new discovered link to be *non-legitimate* (or *invalid*) and sends an alert to the controller to either reject it (aggressive resolution) or to further investigate and decide if the link should be rejected (conservative resolution).

To maintain the status of active links with corresponding ports of each switch, the controller is assumed to have the ability to monitor and obtain updates of this information as that used in [13]. The SDN controller stores the network inventory and traffic data from the control and data planes for processing and generating network statistics. In this paper, we slightly modify the monitoring parameters, for example, by including the port-link mapping between the switches. Specifically, let T be a table, where each column represents active switch and each row represents an instance of an active link.

Defense on Switched-based Link Discovery Attack. As an example, consider a scenario in Fig. 5. The controller creates LLDP PACKET(S_1, P_1) in Step 1, then sends the PACKET-OUT(S_1, P_1) message to S_2 in Step 2, causing the LLDP packet to be sent to S_1. The receiving switch $S2$ sends PACKET-IN(S_1, P_1, S_2, P_2), which in turn is discovered as a link from S_1 (via P_1) to S_2 (via P_2). This link is represented in the first row of Table I. The entry is an active port of the switch column for each end of the link.

TABLE I: ACTIVE LINKS OF NETWORK IN FIGURE 5.

Link	S_1	S_2	S_3
(S_1, S_2)	P_1	P_2	
(S_3, S_2)		P_1	P_2
(S_1, S_3)	P_1		P_2

Similarly, the link (S_3, S_2) in second row of the table is discovered. Since its active ports of both ends have not been shared with other links, therefore the link is legitimate to be added in the controller's topology view of the network.

Now consider the compromised Switch S_2, where the attacker has spoofed S_3 and sends a malicious fake *package-in* message to the controller (i.e., PACKET-IN(S_1, P_1, S_3, P_2) from S_2 in Fig. 5). If we were to add this link into the controller's topology view, the monitoring table would appear as shown with an additional last link in the last row. But now the defense mechanism observes that active ports P_1 of switch are shared by two links connecting with S_1, namely link (S_1, S_2) and link (S_1, S_3). Thus, the defense mechanism will alert the controller of a potential threat to take further action. Consequently, the last link (S_1, S_3) is detected as non-legitimate and will not be allowed to add in the table.

Fig. 6 summarizes the algorithm described above in more details.

Procedure *Link validation*(NewLink((S_1, P_1, S_2, P_2), T)

Inputs: NewLink (S_1, P_1, S_2, P_2);
T, a table of active links, ports, switches as in TABLE I

Output: Is Link (S_1, P_1, S_2, P_2) legitimate?

1 Legitimate ← True

2 **If** P_1 is a host port (i.e., port of switch attached to a host) of S_1 P_2 is a host port of S_2
then Legitimate ← False;
Send alert to controller and exit

3 $P(S_1)$ ← {Port P | P is an active port of Switch S_1 of an active link in the table}

4 $P(S_2)$ ← {Port P | P is an active port of Switch S_2 of an active link in the table}

5 **If** $P_1 \in P(S_1)$ or $P_2 \in P(S_2)$
then Legitimate ← False;
Send alert to controller and exit

Fig. 6. Proposed simple defense mechanism

Defense on Host-based Link Discovery Attack. The proposed mechanism is simple and general in that it is also applicable to Host-based Link Discovery Attacks as well. As an example, consider a scenario in Section III.*A*, which is a network in Fig. 4 with a Host H_1 connects with Switch S_1 (via Port P_4) and Host H_2 connects with Switch S_2 (via Port P_3), where H_1 has been compromised.

Before the attack, the controller sends *call-out* message to S_1 to forward LLDP PACKET (S_1, P_3) to S_2. As a result, the receiving switch S_2 (via P_2) sends PACKET-IN(S_1, P_3, S_2, P_2) to the controller. Since this is the first link and none of the port is a host port, the link is legitimate and added to the first row of Table II.

Similarly, S_1 forwards LLDP PACKET (S_1, P_4) to H_1. The controller only discovers links between switches. If H_1 is not malicious there will not be additional active link added to the table. Instead, since H_1 is compromised, the attacker captures the LLDP PACKET(S_1, P_4) and sends the spoofed LLDP PACKET(S_2, P_3) to S_1. As a result, S_1 sends PACKET-IN (S_1, P_4, S_2, P_3) to the controller. If we were to update Table II with this new link, the result would have been shown as in the second row of the table. Since P_4 is a host port, our defense mechanism would determine that this new link is non-legitimate and sends alert to the controller to reject or re-examine the link

before accepting it. The second row of the table would not have been there in the table T at the control plane.

TABLE II: ACTIVE LINKS OF EXTENDED NETWORK OF FIGURE 4.

Link	S_1	S_2	S_3
(S_1, S_2)	P_3	P_2	
(S_1, S_2)	P_4		P_3

IV. IMPACT ANALYSIS OF LINK DISCOVERY ATTACKS

Impact analysis assesses likelihoods of consequences of any exploitation of vulnerabilities (or attacks) in the network. This paper focuses on impacts of the SDN Link Discovery Attacks on packet Routing, which is one of the most fundamental services of SDN.

Although we have shown that host-based and switch-based link discovery attacks are slightly different (in terms of the attacker's actions), both falsify the network topology, disrupt the network functions, and escalate to the same resulting impacts such as denial-of-service and man-in-the-middle attacks [14]. As a result, the impacts on poisoning network topology from these attacks are the same no matter how the attacks are performed. Thus, our impact analysis will not differentiate the host-based and switch-based attacks but focus on the resulting consequences to SDN's routings.

Most existing work analyzes the network empirically by verifying the feasibility of the attacks [2], [5] using simulation tools to emulate the SDN network behaviors [15]. However, we propose two approaches: *analytical* and *empirical* analyses. The proposed analytical approach is simple, but it can be applied to any network topology.

A. Analytical Approach

The analytical approach is useful for estimating security impacts to help detect *overall system* topology security flaws, should there be discovery link attacks, before *its* deployment. Although the method is simple, it is grounded by a commonly known probabilistic model. Here we assume that, unless specified, at any switch, if there are multiple routing options, all data packets are equally likely to be forwarded to each optional switch in order to provide load balancing. For example, S_1 has two options to forward the packet (i.e., to S_2 or S_4). The number of packets received by S_2 or S_2 should be 1/2 (probability of S_2 to be selected out of the two alternatives) of the number of the packets obtained by S_1.

During the routing, it is possible that the packets may not reach the destination in an expected duration due to delays in traffic or services of switches on the routing path. If the timeout occurs before all the packets are delivered, there are several routing schemes. For example, the routing manager can continue sending additional packets on a different route or can start over and lose the packets sent so far. For simplicity, the proposed analytical method will assume the latter. The selection of suitable alternative routing paths is to select the shortest path first but if the alternatives are of equal path length then the alternative route is selected at random. In addition, we use the UDP (User Datagram Protocol) communication protocol where the sender switch does not wait for acknowledgements. However, in our empirical analysis, we consider both UDP and TCP (Transmission Control Protocol), where sender expects an acknowledgement from the receiver when the packets are received.

The key element of the proposed approach is to compute the estimated the likelihood of the number of packets received on each switch, when all routes are considered, and most importantly the destination switch, as this indicates the effectiveness of the routing (or impacts of attacks on the resulting packet delivery).

Let $L(S_i)$ be an estimated likelihood of the number of the packets S_i received. Suppose S_i has n alternatives to forward these packets. Let p_{ij} be the probability that S_i will transmit packets received to switch S_j (connecting with S_i). Thus, the estimated number of packets S_j received from S_i will be $p_{ij}.L(S_i)$. Note that if we assume that packets transmission from S_i to S_j are equal likely, then $p_{ij} = 1/n$. Suppose S_j can receive packets from m routing alternatives (i.e., incoming routes to S_j other than from S_i). We have:

$$L(S_j) = \sum_{i=1}^{m} p_{ij}\ L(S_i) = \sum_{i=1}^{m} \frac{1}{n} L(S_i) \quad (1)$$

Due to delays in network traffic or delays of a particular switch function (e.g., communication with SDN's controller to identify or create flow rule), some packets may not get transmitted in time for a set "timeout" period causing unsuccessful transmission. The packets are dropped when timeout occurs. In such a case, the routing manager searches for alternative routes and select a route to start sending the packets over again from S_1 (there maybe other variations of communication protocols but the same concept of our mathematical analysis can still be applied). As described before, most routing strategy selects the shortest path first and randomly selects among those of equal path length. We will now illustrate the proposed probabilistic impact assessment in three scenarios of the same network topology in more details.

***Scenario1 Normal with no attacks*:**

Consider a routing scenario of a network with six switches as shown in Fig. 7, where the routing starts from switch S_1 to the destiny at switch S_6. Both S_1 and S_6 are connected to host X and host Y respectively. Note the hosts are not relevant to our analytical impact assessment here but they are to attacks for empirical analyses.

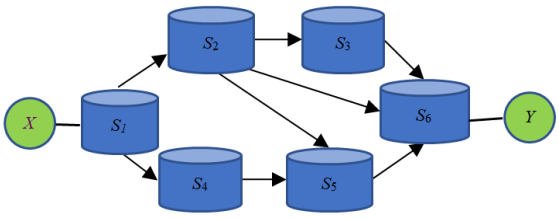


Fig. 7. Routing of packets from source S_1 to destination S_6.

In this scenario a source switch S_1 sends packets to the destination switch S_6 with a transmission rate of about 10 packets/sec through each link. When timeout occurs, the

routing manager seeks alternative route based on the path length. To simplify our illustration, here the routing manager will select the alternative paths in the following order, namely (S_1, S_2, S_6), (S_1, S_2, S_3, S_6), (S_1, S_2, S_5, S_6), and (S_1, S_4, S_3, S_6). Furthermore, attempts to send packets starting from S_1 occur as follows.

First, S_1 successfully routes 10 packets to S_6 (no timeout). Then the next 10 packets are sent through the same path (i.e., (S_1, S_2, S_6)) but the packets are delayed this time at S_2 and timeout occurs. Thus, the routing starts sending the third set of 10 packets again from S_1 using an alternative route of (S_1, S_2, S_3, S_6). This times the routing also fails because of the delays at S_3 and eventually expires S_3's timeout. The routing now tries again with the next alternative route from S_1, namely (S_1, S_2, S_5, S_6). Unfortunately, the timeout occurs at S_5, and finally the last alternative route of (S_1, S_4, S_3, S_6) is tried and successfully transmits the packets to the destination Switch S_6.

TABLE III: NUMBER OF PACKETS BEING FORWARDED - NO ATTACK

Path	S_1	S_2	S_3	S_4	S_5	S_6
(S_1, S_2, S_6)	10	½ •10				⅓ •10
(S_1, S_2, S_6)	10	½ •10				
(S_1, S_2, S_3, S_6)	10	½ •10	⅓ •10			
(S_1, S_2, S_5, S_6)	10	½ •10			⅓ •10	
(S_1, S_4, S_5, S_6)	10			½ •10	10	10
Total impacts	$L(S_1)$	$L(S_2)$	$L(S_3)$	$L(S_4)$	$L(S_5)$	$L(S_6)$
	50	20	3.3	5	13.3	13.3

Based on the scenario described, we can fill in the estimated number of packets received at each switch in a corresponding path. As shown in Table III, packets received at S_2 and S_4 from S_1 is a half of packets of S_1, while S_3 receives only one third of those received by S_2 (since there are three alternatives).

As shown at the bottom of Table III, we can compute the estimated likelihood of the number of packets received at each switch easily by summing the number of packets received by all routing activities. For examples, $L(S_1) = 50$ but $L(S_6) = 13.3$ since there are a lot of routing failures in this scenario. This scenario is pessimistic in order to illustrate the technique. This preliminary structure maybe too tedious to analyze now as it requires the system designer to anticipate what can happen in the routing scenario. The subject to predict such behaviors of the network is beyond the scope of this paper.

Scenario2 Attack scenarios:

We consider two attacks, namely an attack at S_2 and an attack at S_3 (one attack at a time). We assume the same delays and timeouts as used in the normal scenario.

In the first case, suppose S_2 is compromised. Consequently, attacker at S_2 will not forward the packet further causing the timeout. Thus, the routing manager will use an alternative route and start over. This is different from the normal case where the first 10 packets are successfully transmitted. In order to compare the normal scenario with the attack scenario at S_2, after the destination S_6 receives 10 packets, we continue on with the next 10 packets (so that a total of 50 packets are sent from S_1 in both scenarios), in which case, the shortest path will be applied. The summary of the estimated likelihood of the number of packets received is shown in Table IV. The last 10 packets sent from S_1 is shown in the last row of the table.

TABLE IV: NUMBER OF PACKETS BEING FORWARDED- S_2 ATTACK

Path	S_1	S_2	S_3	S_4	S_5	S_6
(S_1, S_2, S_6)	10	½ •10				
(S_1, S_2, S_3, S_6)	10	½ •10				
(S_1, S_2, S_5, S_6)	10	½ •10				
(S_1, S_4, S_5, S_6)	10			½ •10	10	10
(S_1, S_2, S_6)	10	½ •10				
Total impacts	50	20	0	5	10	10

As shown in Table IV, the number of packets received at the destination switch is at a slightly decreased rate of 10/50 = 20% as opposed to the rate of 13.3/50 = 26.6% in the normal case when no attack on S_2. If our scenario in the normal case did not have a timeout at S_2, the resulting impacts would have been greater.

Next, we consider when an attack occurs at S_3. The resulting impact analysis is summarized in Table V.

TABLE V. NUMBER OF PACKETS BEING FORWARDED- S_3 ATTACK

Path	S_1	S_2	S_3	S_4	S_5	S_6
(S_1, S_2, S_6)	10	½ •10				⅓ •10
(S_1, S_2, S_6)	10	½ •10				
(S_1, S_2, S_3, S_6)	10	½ •10	⅓ •10			
(S_1, S_2, S_5, S_6)	10	½ •10			⅓ •10	
(S_1, S_4, S_5, S_6)	10			½ •10	10	10
Total impacts	50	20	3.3	5	13.3	13.3

As seen in Table V, attack at S_3 in this scenario has no impact on the number of packets received at S_6. If the scenario did not have timeout at S3 for path route (S_1, S_2, S_3, S_6), we may see more impacts of the attack since S_3 will not forward the packet (as opposed to timeout).

However, we can see that the impact of the attack on the network routing becomes more severe if an attack occurs on a more connected switch (i.e., S_2) than less connected switch (i.e., S_3) that are on the path to the destination.

The results of the analysis are meant to illustrate the key concept and contribution of the proposed mechanism and not on the actual results themselves. The proposed mechanism gives a framework that allows a systematic analytical analysis to estimate impacts of SDN attacks.

B. Empirical Approach

Most SDN's research relies on a simulation of SDN architecture as a tool to verify their studies. This section shows how we analyze the impacts of link discovery attacks empirically.

SDN Simulation and Experimental Setup:

Here we used Mininet [16] for emulating virtual SDN/OpenFlow networks, and POX [17], a Python-based controller for software-defined networking simulation, which we used to perform the link discovery attack. In this paper, our simulation considers only cases when the network has a single attack at a time.

For the Link Discovery Attack, we simulated a "fake" link between two nodes by injecting a false message to the controller notifying a packet being sent from either a

non-existing sender's ID or from a legitimate sender's ID/address but with a non-existing/unused port. The simulation was run on HP v7x machine having Intel(R) Core(TM) i7-5600U CPU at 2.60 GHz, and 8GB of RAM, running 64 bit Windows 10 Pro.

We ran experiments on the network as shown in Fig. 7, where the packets are sent from host *X* to host *Y*. For the attack scenarios we want to further verify our analytical results that attack at most connected switch (i.e., S_2) yields more severe consequences than the attack at the less connected one (e.g., S_3). To obtain realistic results, we ran experiments for both UDP and TCP protocols as in practice. Link discovery attacks from the compromised switch leads to the fabrication of the links following the switch (e.g., links (S_2, S_3), (S_2, S_6), (S_3, S_6)) Recall that only one single attack occurs at a time. In each scenario, we set hard timeouts of 10 seconds. This means that a flow table entry is removed after 10 seconds and a new flow rule or path has to be recomputed and identified by the controller. The total time for running each session of the experiment was set for 10 minutes.

Experimental Results:

We compare the number of packets received at each switch in the three scenarios: normal, link discovery attack at S_2 and link discovery attack at S_3. Here the critical switch is the destination switch S_6, which connects to host *Y*. For the UDP protocol (no acknowledgement), an estimated of 520 packets are sent for each scenario. Fig. 8 shows the comparison results obtained for the UDP protocol.

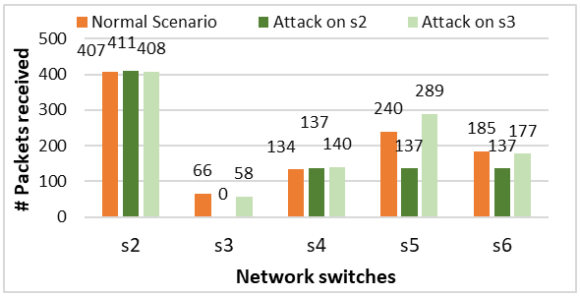


Fig. 8. Comparison of packets received using UDP protocol.

As shown in Fig. 8, while the numbers of packets received at S_2 are comparable for the three scenarios (i.e., 407, 411, and 408 for normal, attack at S_2 and attack at S_3, respectively), they are not at the destination switch S_6. As expected, the number of packets received when attacks occur are 137 and 177, which are lower than that of the normal case of 185 packets.

Here attack at S_2 yields less number of packets received than attack at S_3. This is consistent with results obtained from our mathematical analysis in that both confirm our hypothesis that the link discovery attack on the most connected switch (S_2) is more severe than attack on less connected switch (S_3).

Fig. 8 also shows significant reduction of the number of packets received on its other connecting switches (e.g., reduction from 66 to zero on S_3, and from 240 to 137 on S_5). The reduction of the number of packets received as a result of attack at S_3 goes directly to S_6, whose packets are received from multiple paths. Thus, we cannot isolate the immediate impact of attack at S_3 alone. On the other hand, the numbers of packet received at S_2 and S_4 in the three scenarios are comparable. Based on the network topology in Fig. 7, it is clear why the attacks have no impacts on these switches (as they are not on the path following the attack switches).

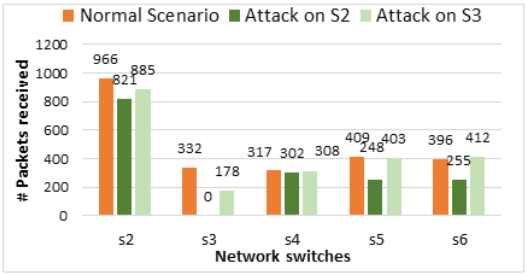


Fig. 9. Comparison of packets received using TCP protocol.

Fig. 9 shows the comparison results obtained for the TCP protocol. Here an estimated of 1535 packets are used for each scenario. The results are similar to the UDP case. The destination switch S_6 received the least number of packets of 255 on the S_2 attack compared to that in the normal case of 396, and that of 412 in the S_3 attack. Because of the nature of TCP communication that requires acknowledgement, which causes extra delays along with the fact that the attack at S_3 barely effects the number of packets received at S_6 (since it is one out of three routes), the resulting packets received at S_6 when attack occurs on S_3 is slightly higher than that when no attack occurs.

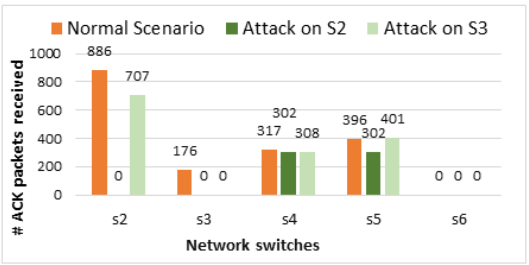


Fig. 10. Comparison of ACK packets received in TCP protocol.

The TCP communication protocol requires a recipient switch to send an acknowledgement (ACK packets). Fig. 10 shows the number of ACK packets received at each switch in all of the three scenarios. As shown in Fig. 10, on S_2 attack case, there is zero ACK packets received at S_2 and the following switches, S_3 and S_6 on the routing path. Since S_6 is the end of the route, there is no need to send ACK packets to it. Thus, no ACK packet received at S_6 in every scenario.

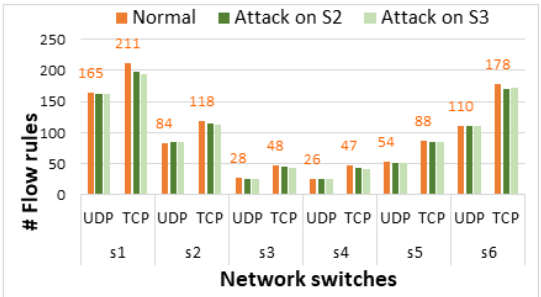


Fig. 11. Comparison of flow rules installation.

Fig. 11 shows the number of flow rules installed. As shown in Fig. 11, the number of flow rules at each switch fluctuates. However, the numbers compared in all of the three scenarios are about the same, whether there is attack or not. This is because the attacks do not impact the flow rule activities more than the normal case. On the other hand, the numbers of flow rules installed at each switch in the TCP case are always higher than those of the UDP case (e.g., at S_2 the numbers of installed flow rules are 84 in UDP vs. 118 in TCP in the normal scenario) in all three scenarios. The reason is due to the installation of additional flow rules of ACK packets.

In this paper we use the packet drop count to signify link discovery attacks. Although we have not illustrated here, it should be relatively easy to see that attacks can also be detected by finding the active ports, which are already in use in a particular switch (as explained in the algorithm in Fig. 6).

V. Related Work

There have been a large number of studies that address various security issues of SDN [7], [8], [14], [18]. However, most of them do not address the fundamental vulnerabilities of the OpenFlow-based controllers [10].

Recent work on security of topology discovery has been researched [2]-[5], [9] as it is an important service of SDN's controller. Topology discovery mechanisms are based on the Open Flow Discovery Protocol (OFDP), which has been shown to be vulnerable in that an attacker can poison the topology view of the SDN and create spoofed links by injecting fake control packets into the network via one or more compromised hosts [4]-[6]. Hong *et al* [11] introduced a link fabrication attack through a compromised host, while Showyra *et al* [19] introduced two new attacks called Port Amnesia and Port Probing. The former enables an attacker to reset the port type while the latter enables an attacker to send a fake message on port configuration, both with the aim for the following link fabrication attacks to escape the detection mechanism. Unlike the above work, we identify a link discovery attack through switches, which occurs when a switch is compromised (using tools e.g., [20]).

To improve the OFDP topology discovery, work in [10] aims to improve both efficiency and security while majority focuses on defense mechanisms and counter measures [2], [4]-[6], [12]. Two approaches, one of which implements a real-time system that automatically detects an attack when it occurs (e.g., SPHINX[15]). SPHINX compares network behaviors with "normal" behavior to detect anomaly and sends alert to the controller. The other approach is by adding an extra authentication to the LLDP packets and ignoring all LLDP packets originating from the host port as used in TopoGuard [11], which is later extended to TopoGuard+ [12] to handle the port attacks. TopoGuard and Alharbi *et al* approach are similar in that both use HMAC, a keyed-hash authentication mechanism [5]. However, TopoGuard uses a static secret key, for computing HMAC and therefore is vulnerable to replay attacks. In addition, Alharbi [5] also discussed LLDP spoofing with technical details of the attack and provided empirical analysis to verify the feasibility of these attacks.

More recent work on link discovery attacks have been studied [6], [8], [10], [21]. In [21] the authors have shown that the OFDP is vulnerable to attacks that can cause a serious impact on the network. Alimohammadifar et. al [10] has shown that one of the most vulnerable attack is link discovery attack which can poison the topology view of the SDN and create spoofed links by injecting fake control packets into the network via one or more compromised hosts. Similarly, Nehra *et al* [8] has shown the other similar attacks that can poison the network. Authors in [21] also have shown that these topology poisoning attacks can lead to other attacks like MiTM which can also eventually cause a threat to the controller. Azzouni *et al*. [7] have tried to improve the OFDP by introducing an improved protocol sOFTDP while others have tried to propose some defense mechanisms and counter measures. Some have used probing mechanism by sending probing packets in order to verify legitimate links and identify fake links independent of how a fake link is fabricated as in [8], [10] while others present a technique to detect Link Fabrication Attack by observing if the packet traffic exceeds normal threshold [21].

Our work is similar to the above in that we aim to automatically detect link discovery attacks to alert the controller. However, unlike any of the above, our detection technique does not use authentication mechanisms or comparison of network behavior but an active port status to help detect malicious behaviors. Furthermore, our detection mechanism can detect both host-based and switch-based link discovery attacks. Finally, unlike [4], [5] that provide empirical analysis, we provide an impact analysis framework to estimate consequences of the attacks.

VI. Conclusion

This paper addresses security challenges of topology discovery, an essential service of SDN controller. We show how topology discovery attacks can occur in OpenFlow discovery protocols via compromised hosts and switches and present a simple detection technique for both cases as a defense mechanism. The paper also describes an analytical technique to analyze impacts of these attacks on network routing and verify some hypotheses with empirical analysis. Future work on additional security can enhance current limitations including securing controller's tracking table, and HMAC or LLDP authentication for the LLDP packet fields to prevent it from forging.

Conflict of Interest

The authors declare no conflicts of interest.

AUTHOR CONTRIBUTIONS

Sonali Sen Baidya conducted the research including formulating idea, performance evaluation to the final manuscript. Rattikorn Hewett supervised this work by investing a full guidance to conduct this research. However, both authors had approved the final version.

REFERENCES

[1] D. Kreutz, F. Ramos, P. Verissimo, *et al.*, "Software-defined networking: A comprehensive survey," *Proc. of the IEEE*, vol. 103, no. 1, pp. 14-76, 2015.

[2] A. Dawoud, S. Shahrestani, and C. Ruan, "Software-defined network controller security: Empirical study," in *Proc. International Conference on Information Technology and Applications (ICITA)*, Sydney, Australia, 2017.

[3] A. Abdou, P. V. Oorschot, and T. Wan, "A framework and comparative analysis of control plane security of SDN and conventional networks," arXiv preprint arXiv: 1703.06992, *2017*.

[4] S. Khan, A. Gani, A. Wahab, M. Guizani, and M. Khan, "Topology discovery in software defined networks: Threats, taxonomy, and State-of-the-Art," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 303–324, 2017.

[5] T. Alharbi, M. Portmann and F. Pakzad, "The (in) security of topology discovery in software defined networks," in *Proc. IEEE 40th Conf. Local Computer Network (LCN)*, 2015, pp. 502–505.

[6] A. Azzouni, R. Boutaba, N. Trang and G. Pujolle, "Limitations of OpenFlow topology discovery protocol," in *Proc. 16th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, 2017.

[7] A. Azzouni, R. Boutaba, N. Trang, and G. Pujolle, "sOFTDP: Secure and efficient topology discovery protocol for SDN," arXiv preprint arXiv:1705.04527, 2017.

[8] A. Nehra, M. Tripathi, M. Singh, R. Gaur, B. Battula, and C. Lal, "SLDP: A secure and lightweight link discovery protocol for software defined networking," *Journal of Computer Networks*, vol. 50, no. 1, pp. 102-116, 2019.

[9] K. Z. Hajji, S. El, and G. Orhanou, "Design and implementation of a new security plane for hybrid distributed SDNs," *Journal of Communications*, vol. 14, no. 1, pp. 26-32, 2019.

[10] A. Alimohammadifar, S. Majumdar, T. Madi, Y. Jarraya, M. Pourzandi, L. Wang, and M. Debbabi, "Stealthy probing-based verification (spv): An active approach to defending software defined networks against topology poisoning attacks," in *Proc. European Symposium on Research in Computer Security*, Springer, 2018, pp. 463–484.

[11] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," *Proceedings of NDSS*, vol. 15, pp. 8-11, 2015.

[12] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Software Defined Network*, 2013, pp. 165–166.

[13] Q. Wander, A. M. C. Miriam, and D. Mario, "An approach for SDN traffic monitoring based on big data techniques," *Journal of Network and Computer Applications*, vol. 131, no. 1, pp. 28-39, 2019.

[14] Z. Shu, J. Wan, D. Li, J. Lin, *et al.*, "Security in software-defined networking: Threats and countermeasures," *Mobile Network Appl.*, pp. 1–13, 2016.

[15] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting security attacks in software-defined networks," in *Proc. 22th Annual Network & Distributed System Security Conference (NDSS'15)*, 2015.

[16] R. Fontes, S. Afzal, S. Brito, M. Santos, and C. Rothenberg, "Mininet-WiFi: Emulating software-defined wireless networks," in *Proc. 11th International Conference on Network and Service Management (CNSM)*, 2015, pp. 384-389.

[17] OpenFlow Networking Foundation, OpenFlow Switch Specification, Version 1.5.0, December 19, 2014.

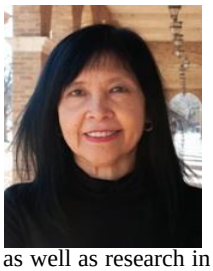
[18] T. Nguyen and M. Yoo, "Analysis of link discovery service attacks in SDN controller," in *Proc. IEEE International Conference on Information Networking (ICOIN)*, 2017, pp. 259-261.

[19] R. Skowyra, L. Xu, G. Gu, *et al.*, "Effective topology tampering attacks and defenses in software-defined networks," in *Proc. 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 374-385.

[20] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," in *Proc. 20th ACM Conference on Computer and Communications Security (CCS)*, 2013.

[21] D. Smyth, S. McSweeney, D. O'Shea, and V. Cionca, "Detecting link fabrication attacks in software-defined networks," in *Proc. 26th International Conference on Computer Communication and Networks (ICCCN)*, 2017.

Sonali Sen Baidya received the B.Eng. degree in Computer Science from Utkal University, India and the M.Tech degree from IIEST, India. She is currently pursuing the Ph.D. degree in the department of Computer Science of Texas Tech University, USA. Her research area is Cybersecurity in Software-Defined Networks.



Rattikorn Hewett is a Whitacre Chair in Computer Science and a department Chair at Texas Tech University. She received a Ph.D. from Iowa State University and a postdoctoral fellow from Stanford University. Her research in *Cyber Security* includes network security, attack models and vulnerability analysis, as well as research in *Data Science*, *Automation*, and *Intelligent systems*. She has published over 100 peer-reviewed technical articles, and has served on several journal editorial boards, and numerous conference program committees.