Aalto University
School of Science
Master's Degree Programme in Security and Mobile Computing

Tien Thanh Bui

Analysis of Topology Poisoning Attacks in Software-Defined Networking

Master's Thesis
Espoo, June 30, 2015

Supervisors: Professor Tuomas Aura, Aalto University
Professor Markus Hidell, KTH Royal Institute of Technology

Advisor: Markku Antikainen M.Sc. (Tech.)

Aalto University
School of Science
Degree Programme in Security and Mobile Computing

ABSTRACT OF
MASTER'S THESIS

Author: Tien Thanh Bui

Title:
Analysis of Topology Poisoning Attacks in Software-Defined Networking

Date: June 30, 2015 **Pages:** 79

Major: Data Communication Software **Code:** T-110

Supervisors: Professor Tuomas Aura
Professor Markus Hidell

Advisor: Markku Antikainen M.Sc. (Tech.)

Software-defined networking (SDN) is an emerging architecture with a great potential to foster the development of modern networks. By separating the control plane from the network devices and centralizing it at a software-based controller, SDN provides network-wide visibility and flexible programmability to network administrators. However, the security aspects of SDN are not yet fully understood. For example, while SDN is resistant to some topology poisoning attacks in which the attacker misleads the routing algorithm about the network structure, similar attacks by compromised hosts and switches are still known to be possible.

The goal of this thesis is to thoroughly analyze the topology poisoning attacks initiated by compromised switches and to identify whether they are a threat to SDN. We identify three base cases of the topology poisoning attack, in which the attack that requires a single compromised switch is a new variant of topology poisoning. We develop proof-of-concept implementations for these attacks in emulated networks based on OpenFlow, the most popular framework for SDN. We also evaluate the attacks in simulated networks by measuring how much additional traffic the attacker can divert to the compromised switches. A wide range of network topologies and routing algorithms are used in the simulations.

The simulation results show that the discovered attacks are severe in many cases. Furthermore, the seriousness of the attacks increases according to the number of tunnels that the attacker can fabricate and also depends on the distance between the tunnel endpoints. The simulations indicate that network design can help to mitigate the attacks by, for example, shortening the paths between switches in the network, randomizing regular network structure, or increasing the load-balancing capability of the routing strategy.

Keywords: Software-defined networking, OpenFlow, Topology poisoning attack

Language: English

Aalto-universitetet
Högskolan för teknikvetenskaper
Examensprogram för datateknik

SAMMANDRAG AV
DIPLOMARBETET

Utfört av: Tien Thanh Bui

Arbetets namn:
Analys av Topologiförgiftningsattacker i Mjukvarudefinierade Nätverk

Datum:	Den 30 Juni 2015	**Sidantal:**	79
Huvudämne:	Datakommunikationsprogram	**Kod:**	T-110
Övervakare:	Professor Tuomas Aura Professor Markus Hidell		
Handledare:	Diplomingenjör Markku Antikainen		

Mjukvarudefinierade nätverk (eng. Software-defined networking) är en framväxande arkitektur med stor potential att styra utvecklingen av moderna nätverk. Genom att separera kontrollplanet från nätverksenheter och centralisera det till en mjukvarubaserad styrenhet, kan SDN ge nätverksadministratörer en översikt över nätverket samt flexibla möjligheter att programmera nätverket. Likväl är datasäkerheten i SDN inte ännu fullständigt förstådd. Även om SDN kan motstå vissa topologiförgiftningsattacker, i vilka attackeraren vilseleder routingalgoritmen angående nätverksstrukturen, så är man medveten om att liknande attacker utförda av äventyrade värddatorer och switchar är möjliga.

Målet med detta diplomarbete är att grundligt analysera topologiförgiftningsattacker initierade av äventyrade switchar och att identifiera om de är ett hot mot SDN. Vi identifierar tre basfall av topologiförgiftningsattacker, där attacken som kräver en enskild äventyrad switch är en ny variant av topologiförgiftning. Vi utvecklar en konceptuell implementation för dessa attacker i emulerade nätverk baserade på OpenFlow, det populäraste ramverket för SDN. Vi evaluerar också attacker i simulerade nätverk genom att mäta mängden av extra trafik som attackeraren kan styra om till äventyrade switchar. Ett brett utbud av nätverkstopologier och routingalgoritmer används i simuleringarna.

Resultaten från simuleringarna visar att de upptäckta attackerna i många fall är allvarliga. Vidare ökar attackernas gravhet i relation till antalet tunnlar som attackeraren kan fabricera och beror även på avståndet mellan tunnlarnas ändpunkter. Simuleringarna indikerar att nätverksdesign kan bidra till att motverka attacker, till exempel genom att förkorta avståndet mellan switcharna i nätverket, göra en regelbunden nätverksstruktur mer slumpmässig eller förbättra routingstrategins förmåga att balansera last.

Nyckelord:	Mjukvarudefinierade Nätverk, SDN, OpenFlow, Topologiförgiftningsattack
Språk:	Engelska

Acknowledgements

First and foremost, I am thankful to my supervisor, Tuomas Aura. I deeply appreciate the time he has spent on discussing my topic and on guiding my work. His mentorship has been invaluable not only for this thesis, but it is also a great inspiration for my future work.

I am grateful to my instructor, Markku Antikainen, for all the time, advice, and feedback he gave me.

I would also like to thank my supervisor, Markus Hidell, for co-supervising this thesis and providing me remote support.

Last but not least, I would like to express my deepest gratitude to my friends and my family, who sincerely supported me throughout my master study.

Espoo, June 30, 2015

Tien Thanh Bui

Abbreviations and Acronyms

BC	Benign Controller
BGP	Border Gateway Protocol
BPDU	Bridge Protocol Data Unit
CDPI	Control Data Plane Interface
ForCES	Forwarding and Control Element Separation
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
LLDP	Link Layer Discovery Protocol
LSA	Link State Advertisement
MC	Malicious Controller
NBI	Northbound Interface
OFDP	OpenFlow Discovery Protocol
OLSR	Optimized Link State Routing
OSPF	Open Shortest Path First
QoS	Quality of Service
RIP	Routing Information Protocol
SDN	Software-Defined Networking
STP	Spanning Tree Protocol
TCAM	Ternary Content-Addressable Memory

Contents

Chapter 1

Introduction

The growth in the volume of data exchanged, stored, and processed in data centers as well as the trend of deploying applications and services on the cloud are driving the demand for a new networking paradigm which is more flexible and automatically responsive than the traditional ones. *Software-Defined Networking* (SDN) [29] is an emerging architecture with a great potential to foster the development of modern networks. The architecture separates the control plane from the network devices and centralizes it at a software-based controller. With this architecture, SDN provides network-wide visibility and flexible programmability to network administrators, allowing them to design and control the network with their own applications and responding quickly to changing business needs.

However, SDN also comes with new security concerns. Attacks can be initiated from malicious management applications, controller, or from compromised network entities, namely hosts and switches. On the other hand, by centralizing the control plane, SDN also offers a considerable simplification to the way we integrate security mechanisms into networks. Some approaches to SDN security have been proposed. Most of them focus on security analysis of SDN applications [11, 37, 41] or real time verification of network policies [21, 26, 27, 38]. However, there has been lots of attention to the vulnerabilities inside SDN controllers which can be exploited by compromised hosts or switches to affect the network [7, 15, 30], if not many solutions.

Among the key innovations provided by SDN controllers, the network topology service is one of the most crucial ones. The service manages up-to-date topological information and provides the information for the management applications, such as packet routing and mobility tracking. Therefore, if this fundamental service does not work properly, all dependent applications will be affected, potentially causing security problems.

Topology poisoning is not a new concept. In traditional networks, an

attacker can issue false routing advertisements, usually from compromised routers, to manipulate how other routers view the network topology [33]. SDN is known to be resistant to some topology poisoning attacks. The reason is that, in SDN, each network device usually updates its state to the controller via a secure channel, which makes it difficult for a compromised entity to spoof the routing information about the other devices. However, it has been discovered that the network topology service of SDN controllers can still be poisoned by either compromised hosts [22] or switches [7, 15]. By poisoning the service, an attacker can, for example, inject bogus links into the network so that some traffic will get diverted away from its intended route and pass through compromised entities instead. After that, the attacker can not only eavesdrop on the traffic but also facilitate man-in-the-middle or impersonation attacks. Some work has been done to address this kind of attacks [15, 22], but none of them provides a thorough solution to the attacks, especially the ones initiated by compromised switches.

Research problem: Our goal is *to thoroughly analyze the topology poisoning attacks initiated by compromised switches to identify whether they are threats to SDN*. We aim to achieve the following:

1. Identify existing and possibly new attacks to poison the network topology from compromised switches.
2. Provide proof-of-concept for all attacks that we discover.
3. Evaluate the significance of each attack in different types of networks.
4. Analyze the impact of network design on the seriousness of the attacks.

Research methods: We first theoretically analyze the specification of OpenFlow [32], the most popular framework for SDNs, to find loopholes in its network topology service and then attempt to discover possible ways to poison the service. Subsequently, we implement the attacks in an emulated network environment to prove that they actually work. Finally, for each attack, we conduct simulations on different kinds of networks and quantitatively measure the significance of the attack in terms of the number of traffic flows it can compromise. Various network topologies and routing algorithms are deployed in the simulations. Note that even though OpenFlow is the main focus of this work, our findings are not specific to OpenFlow. The same results are achievable in other SDN frameworks.

Impact and sustainable development: Topology poisoning is a relatively new kind of attacks in SDN, and, to the best of our knowledge, our work is the first study that focuses on such attacks. By identifying and evaluating

a potential threat in SDN, we believe that this research can foster its development. Once SDN has become mature and been widely deployed, it will, with no doubt, greatly contribute to the global sustainable development. Specifically, it helps to reduce the cost needed to initialize and operate networks. By separating the control plane from network devices, there is no need for intelligent devices in the network, which reduces the appliance expenses. SDN devices are also not required to perform computationally expensive tasks, thereby resulting in substantial energy savings. Furthermore, our research brings ethical impacts to the society since it brings light to a security threat that may be exploited by criminals or may lead to the invasion of privacy. We hope that through this research the community can see the significance of the threat and spend a reasonable amount of effort to mitigate it.

Structure of the thesis: The rest of this thesis is structured as follows. Chapter 2 presents an overview of SDN and the OpenFlow framework as well as a literature review of major SDN security problems. Chapter 3 describes our threat model and all the network poisoning attacks that we have discovered. Thereafter, we test and illustrate the attacks with emulation in chapter 4. Chapter 5 presents the simulations which we have conducted to evaluate the seriousness of the attacks. Chapter 6 presents the simulation results and our analysis on the results. Chapter 7 summarizes our findings and discusses the implications of the findings to network design and SDN deployment. Finally, chapter 8 provides a summary of this thesis.

Chapter 2

Background and Related work

In this chapter, we give an overview of SDN and the fundamentals of the OpenFlow protocol with focus on what is relevant to the context of this work. We also describe major security challenges existing in SDN and what has been done to mitigate these problems, including topology poisoning attacks in SDN and traditional networks.

2.1 Software-defined networking

Software-defined networking [29] is an emerging networking architecture which is manageable, adaptable, and cost-effective, making it ideal for changing business requirements. Its principal idea is to decouple the control plane and the data plane of networking devices. Therefore, the control plane is no longer managed by distributed protocols, like OSPF [34] and BGP [39], but by a logically centralized software controller instead. As a result, network administrators can dynamically shape the traffic in the network from a single place without reconfiguring individual devices. Moreover, the controller has information about the whole network, including the network topology and the state of network resources (e.g. links and switches), hence giving the administrators more flexibility in applying routing policies compared to in legacy networks. For example, the controller can dynamically adjust the routing to avoid congested links or provide different routing algorithms to different types of traffic.

Figure 2.1 presents a high-level overview of the SDN architecture, which comprises the application, control and infrastructure tiers. The SDN applications exist in the application tier. They make their internal decisions by using the abstracted view of the network provided by the controller and specify their network requirements towards the controller via the northbound inter-

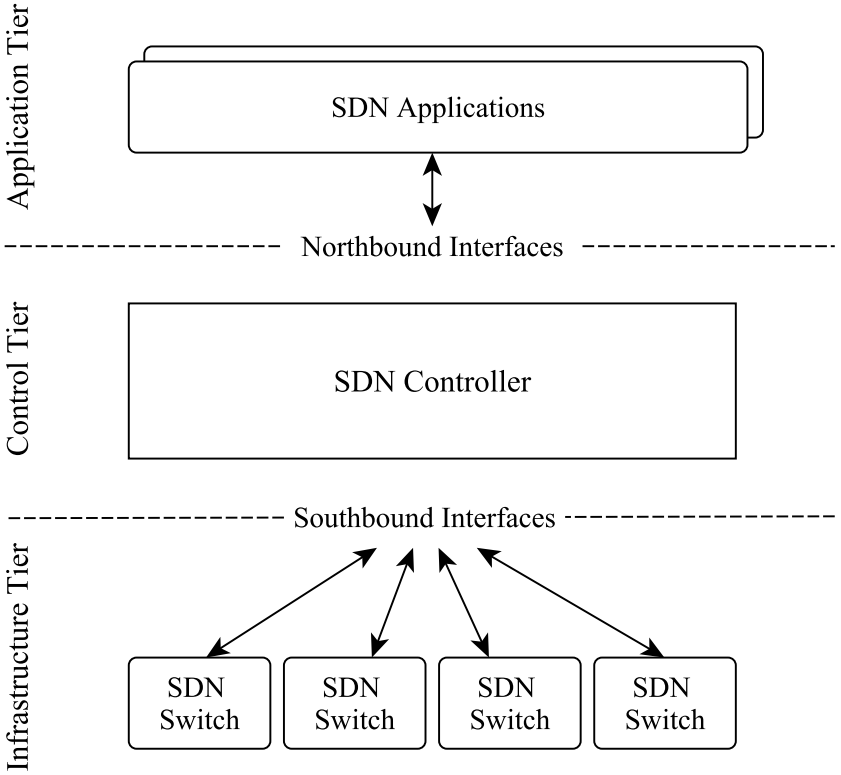


Figure 2.1: An overview of the Software-Defined Networking architecture

faces (NBI). Some examples of SDN applications are network monitors, load balancers, and intrusion detection systems (IDS). In the control tier, besides providing relevant information up to SDN applications, the SDN controller translates the network requirements to configuration commands and sends them to the physical network. Even though the SDN controller is defined as a single logically centralized entity, it can be physically distributed to increase computing power. NOX [20], POX[1], Floodlight[2], OpenDayLight[3], and Maestro [19] are some popular open source controllers at the time of writing. The infrastructure tier includes all of the SDN network devices and cabling for the network. Instead of routers and switches, all SDN network devices are called *switches*, although they forward packets based on both layer-2 and layer-3 headers. The SDN switches exchange control messages with the controller via the southbound interfaces, or control data plane interfaces (CDPI), using

[1]http://www.noxrepo.org/pox/about-pox/
[2]http://www.projectfloodlight.org/floodlight/
[3]http://www.opendaylight.org/

standardized protocols [28]. Some examples of the southbound protocols are Forwarding and Control Element Separation (ForCES) [17, 48], SoftRouter [31], and OpenFlow [32]. Among these, OpenFlow is the leading candidate, and Open vSwitch[4] is one of the most popular switch implementation of OpenFlow.

2.2 OpenFlow

OpenFlow is the most common southbound protocol for the interaction between the switches and controllers, which reside in the infrastructure tier and control tier respectively. The architecture of OpenFlow comprises three main components [10]: *the data plane*, which is composed of OpenFlow switches, *the control plane*, which consists of one or more OpenFlow controllers, and *the control channel*, which connects the data plane and the control plane. The OpenFlow controller manages the data plane by using the control channel to install flow entries to the flow tables in the switches so that the switches can forward data packets according to these entries. In this way, the SDN applications can freely alter the flow tables of switches to apply new routing rules or security models to the network.

This section gives an overview of the three components as described in the OpenFlow protocol specification version 1.0 [1], which we use in this research. Also, this section describes some actual processes that we observe in practice and is not described in the specification. Note that our work is not specific to this version of OpenFlow. The same results can be obtained with other OpenFlow versions or southbound protocols.

2.2.1 Control channel

An OpenFlow switch has to initiate a control channel to the controller before it can exchange any messages with the controller. The connection for the control channel could use plain TCP or be encrypted with TLS. When the switch has successfully established the connection, both endpoints send an OFPT_HELLO message to the other to negotiate the protocol version, which will consequently be used to configure the channel. If the recipients agree on the protocol version, then the process advances. Otherwise, the recipients must reply with an OFPT_ERROR indicating the failure. After the switch and the controller have configured the channel successfully, OpenFlow messages can be exchanged over the channel.

[4] http://openvswitch.org/

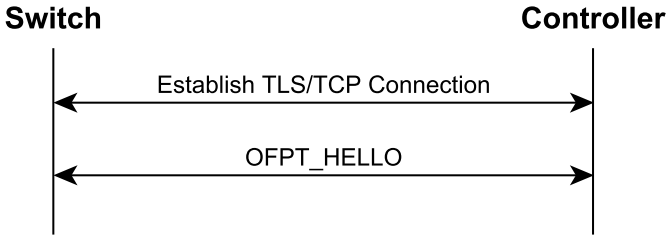
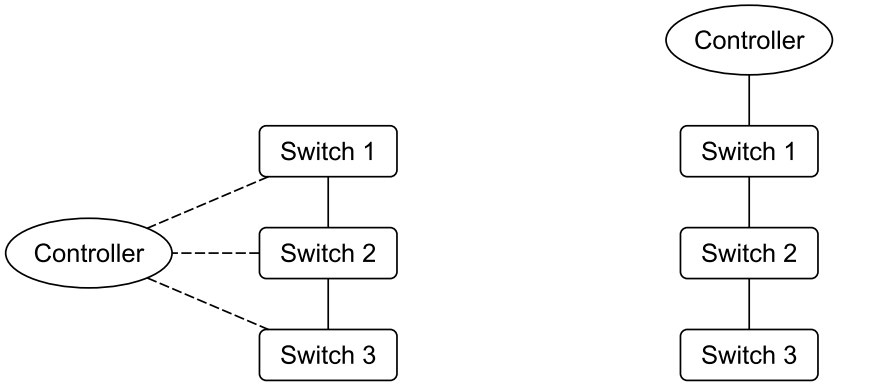


Figure 2.2: Control channel setup process

There are two types of control channel: *out-of-band* and *in-band* [7], illustrated in Figure 2.3. Out-of-band control transmits control traffic via a dedicated network, which can be either physical or logical. In the former case, the OpenFlow switches are directly connected to the controller or to a traditional network that connects to the controller while, in the latter case, the control channel is only logically separate from the data plane, e.g. in a VLAN. Although out-of-band control ensures the security and high availability of the control traffic, it is not always feasible, especially in wireless networks. On the other hand, in-band control uses the same network for both control traffic and data traffic. Apparently, this approach can be deployed more easily but has more security concerns compared to the out-of-band approach.



(a) Out-of-band control (b) In-band control

Figure 2.3: In-band vs Out-of-band control

2.2.2 Data plane

The data plane operates in the infrastructure tier and comprises OpenFlow switches. Its main responsibilities are data forwarding and collecting data statistics. This section will describe how an OpenFlow switch bootstraps its data plane and how the data forwarding process works.

Bootstrapping process

Upon the control channel establishment, the controller and the OpenFlow switch must exchange a set of messages in order for the controller to identify and configure the switch. The process is not described in the OpenFlow specification since it is implementation-specific. In this section, we present the process as we observed while using the Floodlight controller 1.0[5] and Open vSwitch 2.3.1[6]. The process consists of four steps: handshake, switch configuration, switch's description collection, and controller-role configuration, as shown in Figure 2.4. Details of each step are as follows.

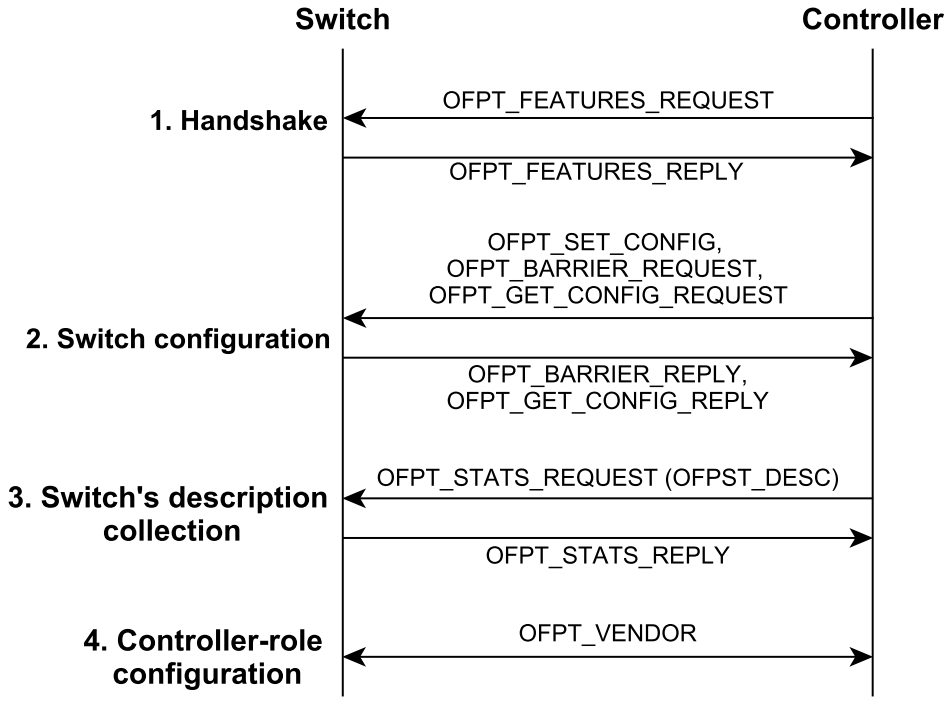


Figure 2.4: Bootstrapping process of the data plane

[5]http://www.projectfloodlight.org/download/
[6]http://openvswitch.org/pipermail/announce/2014-December/000071.html

First, the controller initiates a handshake with the switch in order to identify the switch's features. It sends an OFPT_FEATURES_REQUEST message to the switch via the control channel, and the switch consequently responds with an OFPT_FEATURES_REPLY message containing its basic features, such as the datapath identifier (i.e., a 64-bit number identifying the OpenFlow instance on the switch), the number of ports, and the MAC address of each port.

After the handshake, the controller configures some parameters in the switch with the OFPT_SET_CONFIG message, forces the switch to apply the configuration with the OFPT_BARRIER_REQUEST message, and then verifies that the switch has actually applied the configuration by querying its configuration with the OFPT_GET_CONFIG_REQUEST message. All of these messages are placed in a single packet and sent to the switch. When the switch receives the messages, it applies the specified configuration and subsequently responds with an OFPT_BARRIER_REPLY and an OFPT_GET_CONFIG_REPLY message containing its new configuration.

In the next step, the controller queries the description of the switch with an OFPT_STATS_REQUEST message. The message includes a *type* field which tells what kind of statistics that the controller wants to query. It is set to the OFPST_DESC constant in this case. The switch replies with an OFPT_STAT_REPLY containing its manufacturer description, hardware description, software description, serial number, and a human readable description of its datapath.

Finally, the controller configures its role on the switch by encapsulating the role in an OFPT_VENDOR message and then sending it to the switch. The role can be *equal, master, or slave*. The controller with the equal-role or master-role has full permission to the switch. The difference between these roles is that at most one controller with the master-role is allowed while there can be multiple controllers with the equal-role. The controller with the slave-role only has read-only access to the switch. Upon receive the message, the switch confirms the role by replying with another OFPT_VENDOR message containing the specified role. Note that since the OpenFlow protocol version 1.0 does not support controller roles, this is not a standard way but an extension defined solely for Open vSwitch to configure the controller role.

Forwarding

An OpenFlow switch is a simple forwarding device that processes incoming data packets based on its *flow table*. The flow table contains a set of *flow entries*, each of which includes match fields, priority, counters, instructions, timeouts, cookie, and flags, illustrated in Figure 2.5. The match fields de-

scribe with which packets this entry is associated. They include the ingress port and some specific header fields of packets, such as IP address and MAC address. The priority helps the switch to choose which entry should be executed when multiple entries match a packet. The entry with higher priority is preferred. The counters are used for calculating statistics about flows. The instructions specify how packets matching the entry are processed. Two example instructions are forwarding the packets to a specific port, and dropping the packets. The timeouts consist of two values: an idle timeout and a hard timeout. The hard timeout is triggered at the flow entry's arrival time, while the idle timeout is triggered when the flow is inactive. The expiry of either of these timers causes the flow entry to be removed. The cookie is an opaque data value chosen by the controller, indicating which flow entries are affected by flow statistics, flow modification and flow removal requests. The flags specify how the flow entry are managed; for example, the OFPFF_SEND_FLOW_REM causes the switch to notify the controller when the flow entry is removed.

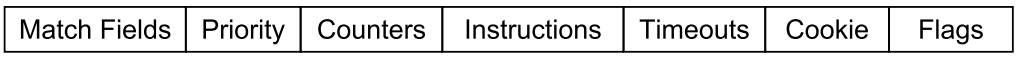


Figure 2.5: Components of a flow entry [1]

When the switch receives an incoming data packet, it first checks whether the packet matches any installed flow entries. If the switch discovers any matching entries, it will select the entry with the highest priority and execute instructions contained in the entry. Otherwise, as the default action, the switch will send an OFPT_PACKET_IN message to the controller to request for a rule or a specific action to be applied to the packet. The controller subsequently issues either an OFPT_FLOW_MOD or an OFPT_PACKET_OUT message containing the instructions on how the switch should process the data packet. The main difference between these messages is that the former instructs the switch to keep the information encapsulated in the message as a flow entry in the flow table so that the switch is able to process similar upcoming data packets without querying the controller repeatedly, while the latter is for one-time packet processing only. These two scenarios are depicted in Figure 2.6.

2.2.3 Control plane

The control plane is responsible for managing the data plane. It consists of one or more software controllers which communicate with the data plane

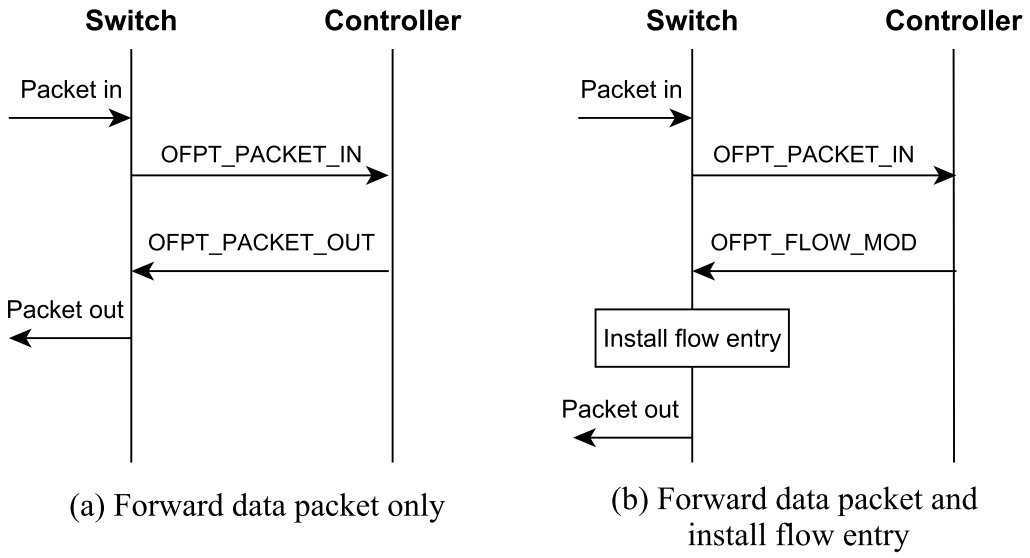


Figure 2.6: Processing a data packet without a matching flow entry

via the control channel. In addition to its role in the bootstrapping process and controlling the forwarding tables, it continuously gathers information from the data plane and provides information to the services and apps in the application tier about the network operations. In this section, we discuss two special tasks of the control plane: link discovery and network statistics monitoring.

Link discovery process

Link discovery is an important process for topology-dependent services, such as network routing. Typically, the OpenFlow controller uses the OpenFlow Discovery Protocol (OFDP), which leverages the Link Layer Discovery Protocol (LLDP) packet [4], to dynamically detect direct links between adjacent OpenFlow switches.

Figure 2.7 illustrates the link discovery process between two switches in an OpenFlow network. To begin with, the controller sends to switch 1 an OFPT_PACKET_OUT message which contains a LLDP packet and an instruction on which port the switch should forward it. Upon receiving the message, switch 1 extracts the LLDP packet and forwards it to the specified port. Subsequently, switch 2 receives the LLDP packet, and then, because it does not know how to process the packet, it encapsulates the packet in an OFPT_PACKET_IN message along with the ingress port identifier and sends the message to the controller. When the controller receives the message, it

can identify a unidirectional link between the two switches. The discovery process in the opposite direction operates similarly. As stated in [22], all popular open-source OpenFlow controllers at the moment use the procedure described above for link discovery.

It is important to point out here that if a switch somehow receives an LLDP packet containing information about another switch to which it is not directly connected, it still sends the packet to the controller, thus resulting in a false link in the network view of the controller.

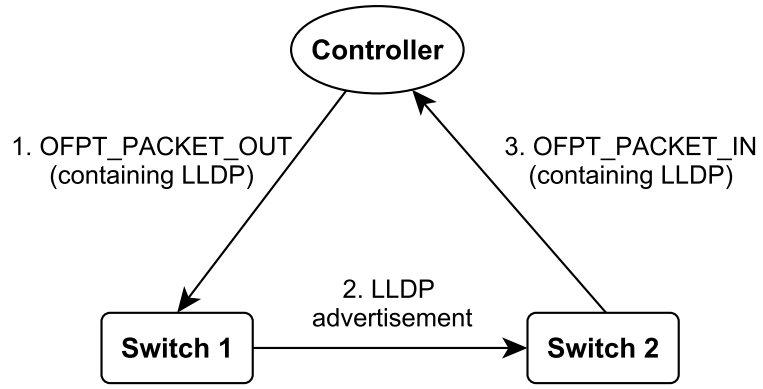


Figure 2.7: Switch-to-switch link discovery in OpenFlow networks

Network statistics monitoring

An OpenFlow switch calculates statistics of the data packets that it has processed and provides the information to the controller upon request. In OpenFlow version 1.0, the statistics consists of flow statistics, table statistics, port statistics, and queue statistics. The *flow statistics* contain information about each of the existing flows in the switch, including the duration that the flow has been alive and the number of data packets and bytes matched the flow. The *table statistics* hold information about each flow table in the switch, including the maximum number of flow entries that it supports, the number of active flow entries, and the number of packets that have been looked up in or hit the table. The *port statistics* consist of information about each physical port of the switch, including the number of received packets and bytes, the number of transmitted packets and bytes, and the number of dropped packets. The *queue statistics* contain the queue status on each port, including the number of transmitted packets and bytes and the number of packets dropped due to overruns. In the latest OpenFlow version (version 1.5) [2], the switch can also provide other types of statistics and meters to

support network monitoring, such as *group statistics, action bucket statistics, and per-flow meter*. With all of these statistics, the controller is aware of the traffic status of each switch and link and can adjust the routing policies accordingly.

2.3 SDN security

Software-defined networking comes with significant advantages over the legacy networking, but, like most new technologies, it also comes with new security threats. This section describes potential security problems that arise from different parts of the SDN architecture and the work that has been done to mitigate the problems.

2.3.1 Attacks from the controller system

Attacks on the controller would have the most severe impact on SDNs because they could affect the entire network. Such attacks could be initiated from the system hosting the controller by exploiting some software vulnerabilities in the system, like the Shellshock [3] or the GHOST vulnerability [24]. Such vulnerabilities might enable the attacker to run arbitrary code or even gain administrator access to the system. Therefore, if no security settings are configured to protect the controller in such cases, the controller would be under the control of the attacker.

As no software is entirely free from bugs, a way to mitigate risks is to update and patch software [36]. Another mitigation strategy is to provide a separate security domain to the controller with techniques such as sandboxing and virtualization. These techniques might help to isolate the controller when attacks happen on the host system.

2.3.2 Attacks from SDN applications

As described in section 2.1, the controller provides interfaces for SDN applications to manage the network. However, the lack of trust between the controller and SDN applications is a security concern since malicious SDN applications could issue any unwanted commands to the network through the northbound interfaces.

In order to prevent such problems, Kreutz et al. [30] propose that the trust between the controller and SDN applications should be established using dynamic trust models [47]. Another mitigation solution, introduced in [37], is to build a security-enforcement kernel that can constrain which

commands and requests an application can issue. However, establishing defending policies that prevent all malicious application behaviors is still not easy.

2.3.3 Attacks from hosts

In enterprise cloud networks, it is feasible for an attacker to acquire a large number of hosts, which can be used to perform various kinds of attacks. Two major attacks that malicious hosts can perform are denial-of-service and network topology poisoning. This section will describe these attacks in detail.

Denial-of-Service

The Denial-of-Service (DoS) attack in SDN involves overwhelming the network's resources so that the routing in the network does not work properly. It can be done by flooding packets to the controller or overflowing the flow tables of switches. These are referred to as *control plane resource consumption attack* and *data plane resource consumption attack*, respectively [40]. Both types of attacks can be done by flooding the network with a large number of forged packets with all fields set to random values. These packets would trigger the switches to send a large number of requests to the controller for new flow rules, thus consuming the control channel bandwidth and the controller CPU resources. As a result, the controller would respond slowly to legitimate requests. At the same time, the switches would also suffer from traffic congestion because the packets could quickly exhaust the memory for flow table storage in the switches [25].

There is some research about how the DoS attacks can be mitigated. OFGUARD [46] detects the attack by monitoring the rate of packet-in messages from the data plane. When the rate is above a certain threshold, it starts filtering all packets with a data plane cache before letting the switches send the packets to the controller. AVANT-GUARD [42] extends the OpenFlow data plane to mitigate DoS attacks. It drops all TCP connections which do not complete the handshake or provide the evidence of actual traffic, i.e. data packets. Kandoi et al. [25] discuss how the network configurations, consisting of timeout values and control plane bandwidth, could be optimized to mitigate the DoS attacks.

Topology poisoning

Hong and Xu et al. [22] propose two ways to poison the network topology in OpenFlow networks: host location hijacking and link fabrication. To the best of our knowledge, it is the only paper that discusses such attacks at the moment.

In the *host location hijacking* attack, an attacker impersonates some target hosts in order to receive the traffic destined to them. The attack is performed by exploiting the Host Tracking Service (HTS), which uses information in the OFPT_PACKET_IN messages to detect if a new host has joined the network or if an existed host has relocated. The attacker can send spoofed OFPT_PACKET_IN messages with the information of the target hosts to the controller in order to make the controller think that the target hosts have been moved to the attacker's location. Two defense strategies to this kind of attacks were proposed: cryptographically authenticating the location of the hosts whenever they relocate, and verifying conditions of a host migration, consisting of the port-down signal and the unreachability of the host in the previous location.

In the *link fabrication* attack, the attacker attempts to create bogus links in the network by manipulating the propagation of LLDP packets in the link discovery process. The attacker can either generate fake LLDP packets and send them to target switches or relay genuine LLDP packets received from one target switch to another. In both cases, the controller would think that a link exists between those switches even though it does not. As a result, all traffic that is routed through the imaginary link will go through the malicious hosts, thereby letting the attacker eavesdrop on the traffic. The solutions proposed to this type of attacks are adding an extra authenticator to LLDP packets so that they cannot be forged and ignoring all LLDP packets coming from a host.

2.3.4 Attacks from switches

Compromised switches not only have the same capabilities as the malicious hosts, but they are also capable of performing more dynamic and severe attacks. First, a compromised switch can be used for traffic eavesdropping [7]. Both data and control traffic passing through the compromised switch can be replicated and sent to the attacker for further processing. Furthermore, the attacker can interfere with the control traffic passing through the compromised switches to perform man-in-mhe-middle attacks [8]. By doing so, the attacker can act as the controller to some target switches. The attacker can also spoof control messages to the controller on behalf of the target switches

to prevent the controller from detecting the attack. TLS can help to prevent these attacks by providing confidentiality for the control traffic and mutual authentication between the controllers and switches.

The topology poisoning attack — the main focus of this work — is also initiated from compromised switches. This kind of attacks has not been studied extensively in the context of SDN. It is first introduced in [7]. Its principle is similar to that of the link fabrication attack caused by malicious hosts: compromised switches send arbitrary LLDP packets to fabricate bogus links in the network. However, compared with the attack originating from hosts, this can be performed in a larger scale. The reason is that a switch is usually connected to more neighbors than a host is. Therefore, a compromised switch can deceive more targets, thus potentially increasing the number of bogus links in the network. It is also more difficult to detect this kind of attacks since it causes less suspicious behavior in the network. For example, it does not require the participation of a host in its manipulation of the link discovery process, hence the solutions in section 2.3.3 for the link fabrication attack are ineffective. Another solution provided by SPHINX [15] is to build and continuously update flow graphs for each traffic flow observed in the network in order to detect anomalies in the network topology. However, the anomaly-detection approach does not cover all possible attack scenarios and can also be circumvented by spoofing additional traffic between compromised switches.

2.4 Topology poisoning in traditional networks

Topology poisoning is not a new concept in traditional networks. It has received a fair share of attention in the research community because of its potential impact. As long as an attacker is able to issue false routing advertisements, usually from compromised routers, it can manipulate how other routers view the network topology [33].

In networks that use distance vector routing protocols like RIP, a router periodically sends link updates to its neighbors. Each update includes a destination identifier and the cost to the destination. When a router receives an update, it re-calculates its distance to the destination using the distributed Bellman-Ford algorithm [9]. The problem is that a router cannot verify the link updates. Therefore, a compromised router can claim it has the least-cost path to a particular address, thus causing it to receive some or all traffic to that address. This kind of attack is referred to as the *black hole attack* [45].

The same idea can be applied in networks that use link-state routing protocols. A popular link-state protocol in wired network is OSPF, in which

each router advertises its links to neighboring routers and networks. These advertisements are called Link State Advertisements (LSA). A compromised router in OSPF networks can either falsify its own LSA [45] or send out false LSAs on behalf of other routers [23]. The result of both cases is that traffic destined to the target addresses will be attracted to the compromised routers. This way the attacker can eavesdrop on the traffic, route it through a longer path, or completely drop it. Besides wired networks, link-state protocols in mobile wireless networks also have to deal with similar security problems. For example, in the Optimized Link State Routing (OLSR) protocol, each node injects topological information into the network by generating HELLO (or TC) messages, which can also be falsified [13].

Manipulating the STP protocol in local-area networks might also divert traffic to compromised switches. This attack is called STP mangling [35]. In the attack, an attacker falsifies BPDUs with the smallest bridge ID in order to cause a compromised switch to become the root of the network's spanning tree. Once it succeeds, the compromised switch becomes the focal point of the network and sees most of the traffic on the network.

Chapter 3

Topology Poisoning Attacks

This thesis focuses on the topology poisoning attacks caused by compromised switches. The idea of poisoning the network topology is not entirely new in SDN. As described in chapter 2, the topology poisoning attacks can be initiated from either compromised hosts or switches, but those originating from switches are potentially more severe and difficult to be detected. Also, they are more interesting because they have not been studied widely. However, the large number of possible attack scenarios makes the security analysis challenging. To tackle this, instead of analyzing all possible attack cases, we focus on the most atomic ones, which can be combined to represent more complex attacks.

This chapter starts by defining our threat model. Thereafter, we describe in detail the three attack cases that we consider in our analysis: two-switch tunnel, extended two-switch tunnel, and single-switch tunnel. The two-switch tunnel attack has been presented in the literature [7, 15], but the other attacks have not.

3.1 Threat model

We consider a multi-homed software-defined network with the following assumptions:

1. The network uses OpenFlow as the southbound protocol.
2. The attacker has compromised one or more OpenFlow switches.
3. Non-compromised switches operate correctly and according to the benign controller's instructions.

4. The control channel is properly protected with TLS protocol, meaning that it provides confidentiality for the control traffic as well as mutual authentication between the controller and switches.

3.2 Attacker hierarchy

Two levels of attackers exist in our threat model. The attacker of the first level has access to the control interface of the compromised switches. Thus, it is able to modify the flow tables of the switches (i.e., add, remove, and modify flow entries in the flow tables) and even reconfigure the switches to communicate with a malicious controller rather than the benign one. However, accessing sensitive information in the compromised switches' firmware, such as the private key used with TLS, is out of the attacker's capabilities. Without the ability to extract the TLS key, the attacker is unable to communicate with the benign controller on behalf of the compromised switches, which makes it easier to detect the attack.

On the other hand, the attacker of the second level has full access to the compromised switches. It means that, besides the ability to issue commands to the switches via their control interface it can extract all kinds of information from the switches, including the TLS key pair. As a result, the attacker can spoof control traffic to the benign controller in order to, for example, make it think that the compromised switches are still working as expected. In brief, the attackers of both levels can implement the same attacks, but the attacker who has full access to the compromised switches can conceal its attacks better.

3.3 Attack scenarios

With the assumptions described above, we present three different attack scenarios: two-switch tunnel, extended two-switch tunnel, and single-switch tunnel. Even though their requirements and goals are different, they are all based on the same principle, which is to manipulate the propagation of the LLDP packets to fabricate non-existing links in the networks. We refer to these imaginary links as *tunnels*. The next sections describe these attack scenarios in detail, including the prerequisites, goals, and attack method.

3.3.1 Two-switch tunnel attack

Prerequisites: Two non-adjacent switches and a third node are under the control of the attacker. The third node can be either a host or a switch.

Goal: To lead the controller to believe that the compromised switches are directly connected.

Method: First, each compromised switch needs to fool the controller into thinking that it has one more port in use than it actually has. The fake active port will be used for the virtual link between the two switches. In order to create a fake port, the attacker can add spoofed port information to the OFPT_FEATURES_REPLY message during the bootstrapping process (see section 2.2.2). After that, whenever one of the compromised switches receives an LLDP packet that is for the fake port, it forwards the packet to the other switch via the third node (Figure 3.1). The other switch consequently encapsulates the LLDP packet into an OFPT_PACKET_IN message and then sends it to the controller. As a result, the controller thinks that there is a link between these switches.

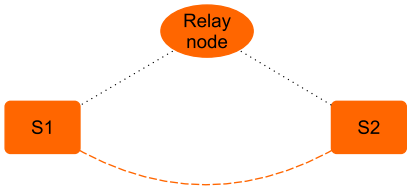


Figure 3.1: Illustration of the two-switch tunnel attack

We refer to the third node as the *relay node*. The use of relay nodes was proposed in [22]. Without the relay node, the switches would not be able to send any packet to each other after the tunnel has been established. The reason for this is that when one of the switches wants to send a packet to the other, it has to forward the packet to a non-compromised adjacent switch first. This switch possibly sends the packet back to the sender since the controller thinks that the shortest route is through the virtual link.

In practice, the packet relay process from a compromised switch is not as simple as changing the destination address of the packets and sending them to the relay node and then from the relay node to the other compromised switch. This is because by doing so it is difficult for the compromised switches to restore the original source and destination addresses of the packets. One way to make relaying packets more feasible is to reconfigure the compromised

switches to communicate with a malicious controller rather than the benign one. The relay node can act as the malicious controller or the medium to forward control traffic between the malicious controller and the compromised switches. In either case, the compromised switches can send any packets to each other through their control channel to the malicious controller.

3.3.2 Extended two-switch tunnel attack

Prerequisites: As above, two non-adjacent switches and a relay node are under the control of the attacker. The relay node can be either a host or a switch.

Goal: To lead the controller to believe that the compromised switches are directly connected and so are their neighbors.

Method: The tunnel between the two compromised switches can be fabricated using the method of the two-switch tunnel attack described in the previous section. In order to fool the controller to think that there are links connecting the neighbors of the two compromised switches, the attacker can configure each compromised switch to forward the LLDP packets that it receives from one neighbor to one of the other switch's neighbors via the relay node. By doing so, if each compromised switch has n neighbors, apart from the tunnel connecting the compromised switches, the attacker can create at most $(n - 1)$ additional tunnels. The attacker cannot create n additional tunnels because it has to keep one neighbor of each compromised switch out of the attack so that it still can forward traffic between the compromised switches and the relay node.

In fact, it is also possible to create tunnels from each compromised switch to the neighbors of the other compromised switch. However, we do not consider such tunnels since less distance can be reduced with such tunnels than with the tunnels connecting the neighbors of the compromised switches.

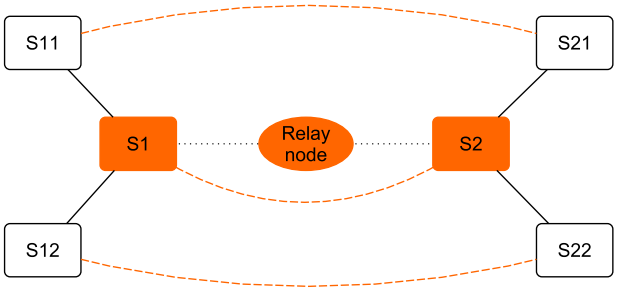


Figure 3.2: Illustration of the extended two-switch tunnel attack

Figure 3.2 illustrates an example of this attack scenario. $S1$ and $S2$ are the two compromised switches, $S1$ is adjacent to $S11$ and $S12$, and $S2$ is adjacent to $S21$ and $S22$. By relaying LLDP packets between $S11$ and $S21$, the attacker can cause the controller to think these switches are directly connected. However, since the attacker cannot manipulate the number of ports of the non-compromised switches, it has to sacrifice the real links between the compromised switches and their neighbors that are used for the attack. Therefore, in the example, if the attacker has established a tunnel between $S11$ and $S21$, it has to logically remove the links $S1-S11$ and $S2-S21$ from the controller's view. This can be done by $S1$ dropping all LLDP packets that the controller instructs it to send to $S11$ and $S2$ dropping all LLDP packets that the controller instructs it to send to $S21$. The attacker can also apply a similar process to fabricate a tunnel between $S12$ and $S22$, between $S12$ and $S21$, or between $S11$ and $S22$.

3.3.3 Single-switch tunnel attack

Prerequisites: A switch is under control of the attacker.

Goal: To lead the controller to believe that the neighbors of the compromised switch are directly connected to each other.

Method: The attacker needs to configure the compromised switch to forward the LLDP packets received from one port to another port instead of sending the packets to the controller. If the compromised switch directly connects to n switches, the attacker can fabricate a maximum of $\lfloor \frac{n}{2} \rfloor$ tunnels between these switches.

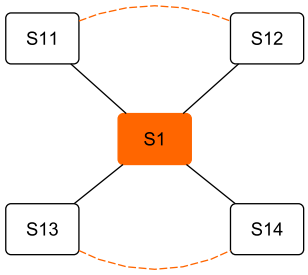


Figure 3.3: Illustration of the single-switch tunnel attack

For example, in Figure 3.3, $S1$ is the compromised switch, and $S11$, $S12$, $S13$, $S14$ are its neighboring switches. By forwarding the LLDP packets received from $S11$ to $S12$, the attack can cause the controller to think that there is a link connecting these switches. However, for the same reason as described in the extended two-switch tunnel attack, the attacker has to sacrifice the links $S1 - S11$ and $S1 - S12$ for the attack. Tunnels can also be created between other switch pairs, such as $S13$ and $S14$.

Chapter 4

Attack Implementation

In chapter 3, we discussed three topology poisoning attack scenarios: two-switch tunnel, extended two-switch tunnel, and single-switch tunnel. In order to demonstrate the feasibility of the attacks, we implement them in an emulated SDN network environment. This chapter first describes the emulation methods, including the tools that were used. Thereafter, we present our controller implementation that we used to control the compromised switches. The last section of the chapter describes the detail of the emulations for each attack.

4.1 Emulation environment

Table 4.1 summarizes the software and methods used in the emulations. We created the emulated networks with *Mininet 2.2.0*[1] — a widely used emulation tool for software-defined networks. Mininet leverages lightweight virtualization to make a single system look like a complete network, in which the hosts run in separate network namespaces with their own network interfaces and addresses while the switches usually work in the same network namespace as the underlying system.

In-band control was configured in the emulated network. This was chosen because it provides more general results than out-of-band control. If the attacks work with in-band control, they also work with out-of-band control. However, in Mininet, the switch implementation does not work well with in-band control. Therefore, instead of using the Mininet switches, we deployed Mininet hosts and run Open vSwitch 2.3.1[2] on them so that they acted as real OpenFlow switches. *OpenFlow 1.0* was selected for the control channel

[1] http://mininet.org/blog/2014/12/09/announcing-mininet-2-2-0/

[2] http://openvswitch.org/pipermail/announce/2014-December/000071.html

Network emulator	Mininet 2.2.0
Switch software	Open vSwitch 2.3.1
Benign controller	Floodlight 1.0
Malicious controller	POX
Routing algorithm	Shortest-path routing
Control communication	In-band
Southbound protocol	OpenFlow version 1.0

Table 4.1: Summary of the emulation setup

since it is widely supported by various controller implementations. We did not enable TLS in the control channel as TLS does not affect the attacks. It would have made the emulations unnecessarily complicated and made it difficult for us to monitor the packet routing.

There are two kinds of switches in the network: benign and compromised. The switches of each kind connect to a different controller. The benign switches are controlled by the *benign controller (BC)*, for which we used *Floodlight 1.0*[3] with its default settings. The benign controller routes data packets in the network using the *shortest-path routing* algorithm [16] and uses the link discovery process as described in section 2.2.3 to discover the network topology. It also provides northbound web interfaces[4], which we used to query information about the benign controller's view of the network topology. On the other hand, the compromised switches are controlled by the *malicious controller (MC)*, which was implemented with POX[5]. This controller acts as the attacker in the network.

4.2 Malicious controller

The malicious controller has two main tasks: spoofing the state of the compromised switches to the benign controller and processing routing requests from the compromised switches. This section explains how we implemented these tasks.

4.2.1 Switch state spoofing

The malicious controller has to send spoofed traffic to the benign controller on behalf of the compromised switches in order to make it think that the

[3]http://www.projectfloodlight.org/download/
[4]https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+REST+API
[5]http://www.noxrepo.org/pox/about-pox/

switches still work as expected. The spoofing process for each compromised switch is done as follows. The malicious controller first establishes a control channel to the listening port of the benign controller (i.e., port 6653) via the compromised switch. In order to do so, it installs two forwarding rules on the compromised switch:

- **Towards the benign controller**: for any packet that is destined to port 6653 on the switch, the switch replaces the source and destination addresses of the packet with its own address and the benign controller's address, respectively, and then sends the modified packet to the benign controller.
- **From the benign controller**: for any packet that is received from port 6653 of the benign controller and is destined to the switch's address, the switch replaces the source and destination addresses of the packet with its own address and the malicious controller's address, respectively, and sends the modified packet to the malicious controller.

With these rules, the malicious controller only needs to establish a connection to the port 6653 of the compromised switch, and the switch then forwards the connection to the benign controller as if it were the connection endpoint. Once this control channel has been established, the malicious controller spoofs the bootstrapping process (see section 2.2.2) to the benign controller with the real information of the compromised switch. However, for the two-switch tunnel and extended two-switch tunnel attacks, additional information of the fake active port is added for the tunnel endpoints. When the bootstrapping process is done, the benign controller believes that the compromised switch is under its control.

4.2.2 Packet processing

When a compromised switch receives a packet that it does not know how to forward, it requests the malicious controller for routing instructions. The malicious controller processes this request as illustrated in Figure 4.1. Since the malicious controller has no information about the network topology, it has to query the benign controller about how to process the request. Hence, it forwards the request to the benign controller through the control channel that goes through the compromised switch, as explained above. The benign controller consequently processes the request and sends instructions back. If the instructions are to forward the packet to the fake port, the malicious controller will forward the packet to the other compromised switch via its control channel. Otherwise, it will send the instructions to the compromised

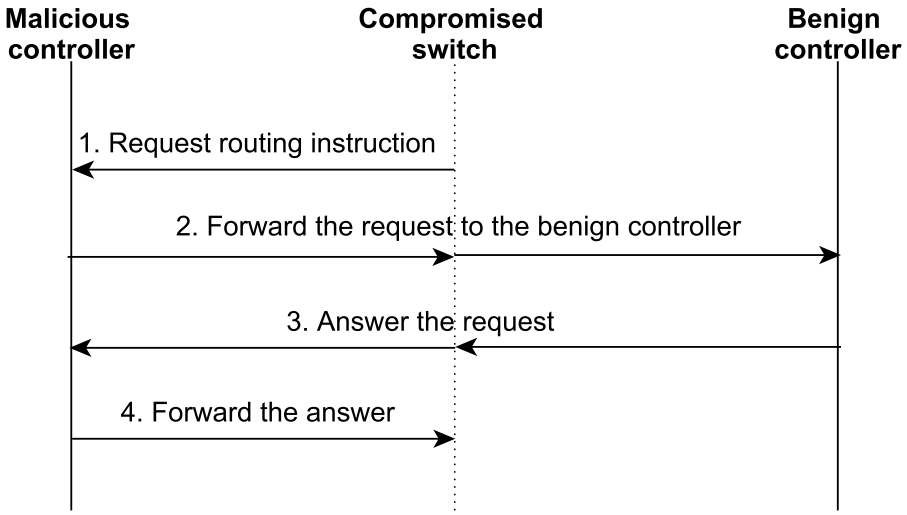


Figure 4.1: Processing a routing request by the malicious controller

switch. As a result, the compromised switch will receive normal instructions from the benign controller, except for the traffic that is routed through the tunnel and for the control channel. The malicious controller acts as the relay node, as described in section 3.3.1.

4.3 Emulation details

For each attack presented in chapter 3, we conducted a separate emulation to demonstrate it and to test its feasibility. To observe the effect of the attacks, we ran each emulation for two situations: *normal* and *attack*. In the normal situation, all switches operate normally under the control of the benign controller. On the other hand, in the attack situation, the compromised switches are controlled by the malicious controller while the other switches still work normally. The next sections describe how the emulations work.

4.3.1 Two-switch tunnel attack

Figure 4.2 shows the emulated network for the two-switch tunnel attack. $S1$, $S2$, $S6$, $S7$, and $S8$ are the normal switches, $S3$ and $S5$ are the tunnel endpoints, $S4$ is the relay node, and $h1$ and $h2$ are the hosts.

Under the *normal situation*, all switches were controlled by the benign controller, which ran on the host connected to $S1$. There are two paths of equal length between $h1$ and $h2$: $[S1 - S6 - S7 - S8 - S2]$ and $[S1 - S3 -$

$S4 - S5 - S2$]. Therefore, when $h1$ pinged $h2$, the ICMP requests went through the former route, while the ICMP replies went through the latter route, resulting in 50% of the data packets going through the compromised switches.

Under the *attack situation*, we ran the malicious controller on the host connected to $S4$ and configured it to be the controller of $S4$. We also configured $S4$ to be the controller of $S3$ and $S5$. $S4$ was configured to forward control traffic between the malicious controller and the tunnel endpoints. We set the controller of $S3$ and $S5$ to be $S4$ and not the malicious controller so that the malicious controller can be at any location in the network.

Subsequently, we triggered the malicious controller to start the spoofing process described in section 4.2.1 for each compromised switch and then apply the method presented in section 3.3.1 to create a tunnel between $S3$ and $S5$. When this had been done, we pinged from $h1$ to $h2$ again. This time, all ICMP packets went through the compromised switches. The reason for this is that the tunnel had shortened the path that go through the compromised switches.

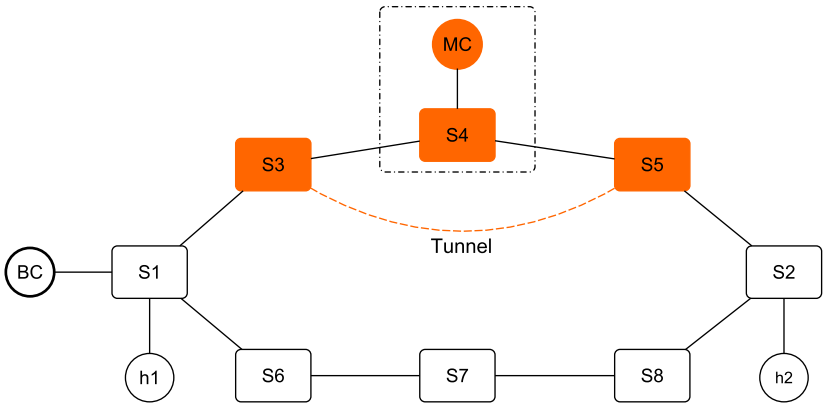


Figure 4.2: Emulated network for the two-switch tunnel attack

4.3.2 Extended two-switch tunnel attack

Figure 4.3 illustrates the emulated network for the extended two-switch tunnel attack. $S1$, $S2$, and $S6$ are the normal switches, $S3$ and $S5$ are the tunnel endpoints, $S4$ is the relay node, and $h1$ and $h2$ are the hosts.

Under the *normal situation*, all switches were controlled by the benign controller, which ran on the host connected to $S1$. There are two paths

between $h1$ and $h2$: $[S1 - S3 - S4 - S5 - S2]$ and $[S1 - S6 - S2]$, which have length of 4 and 2, respectively. Therefore, when $h1$ pinged $h2$, all ICMP packets went through the latter path, meaning that no traffic went through the compromised switches.

Under the *attack situation*, we first configured the malicious controller and the compromised nodes as described in section 4.3.1 to create the tunnel between $S3$ and $S5$. At this moment, the ICMP traffic still did not go through the path that contains the compromised switches since the length of the path was reduced to 3, which was still greater than that of the other path. We then applied the method illustrated in section 3.3.2 to create the tunnel between $S1$ and $S2$. When this had been done, the benign controller thought that there was a direct link between $S1$ and $S2$. As a result, when $h1$ pinged $h2$ again, all ICMP packets were routed through this tunnel, which was actually the path containing the compromised switches.

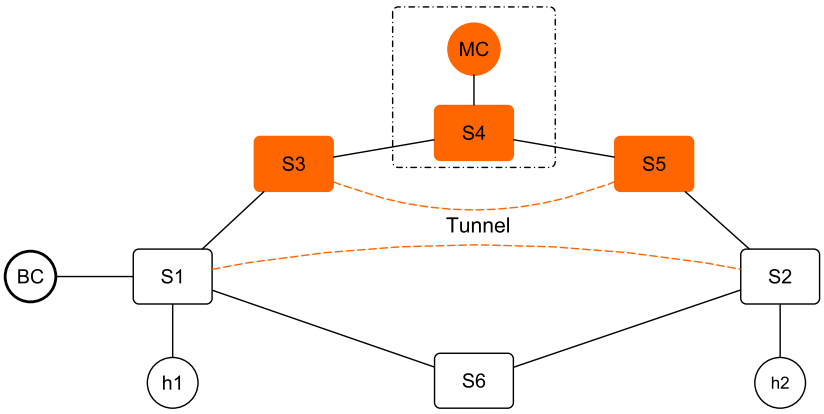


Figure 4.3: Emulated network for the extended two-switch tunnel attack

4.3.3 Single-switch tunnel attack

Figure 4.4 shows the emulated network for the single-switch tunnel attack. $S1$, $S2$, and $S4$ are the normal switches, $S3$ is the compromised switch, and $h1$ and $h2$ are the hosts.

Under the *normal situation*, all switches were controlled by the benign controller, which ran on the host connected to $S1$. There are two paths of equal length between $h1$ and $h2$: $[S1-S4-S2]$ and $[S1-S3-S2]$. Therefore, when $h1$ pinged $h2$, the ICMP requests went through the former, while the ICMP replies went through the latter, meaning that 50% of the data packets went through the compromised switch.

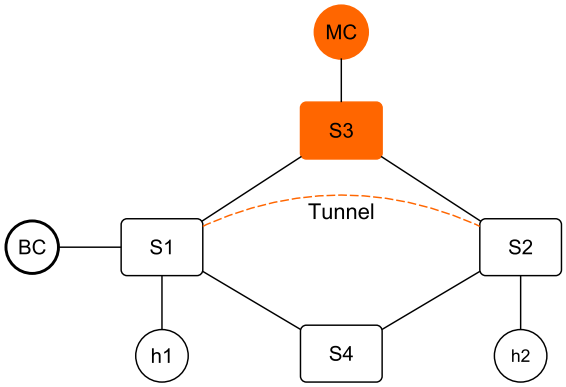


Figure 4.4: Emulated network for the single-switch tunnel attack

Under the *attack situation*, we ran the malicious controller on the host connected to $S3$ and then configured it to be the controller of $S3$. We implemented the malicious controller so that it forwarded all LLDP packets that S3 received from S1 to S2 and vice versa, resulting in a tunnel between $S1$ and $S2$. When this had been done, we pinged from $h1$ to $h2$ again. We observed that all ICMP packets were routed through the tunnel and were routed via the compromised switch $S3$.

Chapter 5

Attack Evaluation Methods

SDN will be deployed in many different types of networks, and the software implementation of routing algorithms in the controller means that a wide variety of routing algorithms may be used. In this thesis, the goal is to analyze topology spoofing attacks on a general level. Thus, rather than picking one specific network and routing algorithm, we assess the threats against a wide variety of network topologies and routing strategies. By doing so, the results will be more general and also teach us about resilient network design.

To assess the threat caused by the discovered attacks, we performed the attacks on large simulated networks and measured how much additional traffic they can divert to compromised switches. The more traffic the compromised switches can attract, the more severe the attacks are. In the simulations, we used three different shortest-path routing algorithms, which are hop-count, fully-deterministic, and load-balance. Seven types of topologies were chosen for the simulations, which are grid, 2D torus, 3D torus, hypercube, triangulated planar, fat tree, and k-ary tree. These routing algorithms and topologies are explained in sections 5.1 and 5.2, respectively. Section 5.3 describes how we ran each simulation. The simulation results are shown in the next chapter.

5.1 Routing algorithms

Data center networks provide environments for increasingly sophisticated applications and services. Each of them comes with different demands on the underlying network. For example, delay-sensitive services, such as web servers, require low communication latency, while big data applications, such as MapReduce, prioritize high throughput and congestion avoidance.

In the simulations, we implemented three shortest-path routing algorithms: hop-count, fully-deterministic, and load-balance. The first two algorithms are for latency-sensitive applications and services while the third one is for ones that do not require low latency but consume a great amount of bandwidth. They leverage the Dijkstra algorithm [16] to perform the path search and use the criteria presented in Table 5.1 to prioritize paths. The next sections describe the routing policies in detail.

	Main criterion	**Secondary criterion**
Hop-count	Smallest number of hops	Limiting flow-table size
Fully-deterministic	Smallest number of hops	Deterministic routing
Load-balance	Optimal bandwidth	Minimum latency

Table 5.1: Criteria used in the routing algorithms

5.1.1 Hop-count routing

The hop-count routing primarily minimizes the number of hops in the path between the source and the destination. For paths with the same number of hops, the maximum number of entries in the flow tables of the switches along each path acts as the tie breaker. The reason why we choose this tie-breaking rule is that latency-sensitive data flows are usually small in size, e.g. HTTP requests to a web server. As a result, they will quickly fill up the flow tables, thus exhausting the switch's TCAM memory. The tie breaker helps to avoid the problem to some extent.

In order to simulate the flow tables in the switches, we keep track of the flows that are currently passing through each switch. The flows are removed from all switches along their route when they finish.

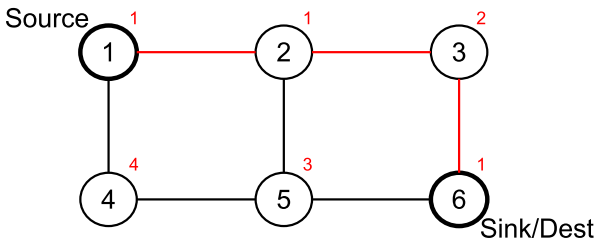


Figure 5.1: An example of how the hop-count routing works

An example of how the routing works is in Figure 5.1. Switch 1 and switch 6 are the source and the destination, respectively. The number of entries in the flow table of each switch is shown on top of it. There are three shortest paths with the same number of hops between the source and the destination: $[1-2-3-6]$, $[1-2-5-6]$, and $[1-4-5-6]$. The maximum numbers of flow entries in the switches along the paths are 2, 3, and 4, respectively. The routing algorithm chooses the path $[1-2-3-6]$ because, among the three paths, it has the lowest maximum number of flows entries in the switches.

5.1.2 Fully-deterministic routing

The routing algorithm explained above is non-deterministic in the sense that the route for a new flow depends on the other flows in the network and, thus, cannot be predicted. Non-deterministic routing is not always desirable, especially for Quality of Service (QoS). Our fully-deterministic routing is to deal with this problem. It forwards data packets on a pre-determined shortest path between the source and destination. This is done as follows. We first generate a weight for each link in the network. The weight is generated uniformly randomly from $[1, 1+\epsilon]$ so that ϵ is small enough for the total weight of any path to be always less than the path's length plus 1. Therefore, the decimals of the link weight act as the tie breaker between paths with equal hop counts.

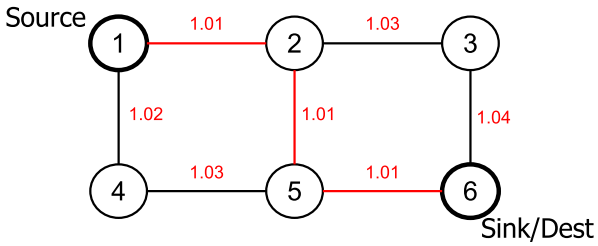


Figure 5.2: An example of how the fully-deterministic routing works

Figure 5.2 illustrates an example of this routing method. Data is sent from switch 1 to switch 6. The weight of each link is shown on top of them. Among the three paths between the source and the destination, the routing would select the path $[1-2-5-6]$ because it has the smallest total weight. The way in which the tie breaker works is not important here. The important fact is that, for the same source-destination pair, the same route will always be chosen.

5.1.3 Load-balance routing

The load-balance routing algorithm uses the number of data flows that are passing each link as the primary cost for routing through that link. By minimizing their sum, the algorithm favors the paths with the greatest amount of free bandwidth. To calculate the total number of data flows on each path, we keep track of the data flows that are currently passing through each link. The flows are removed from all links along their route when they finish. For the paths with the same total number of data flows, the algorithm uses the length of the paths as the tie breaker. This tie breaker is calculated in the same way as in the hop-count routing.

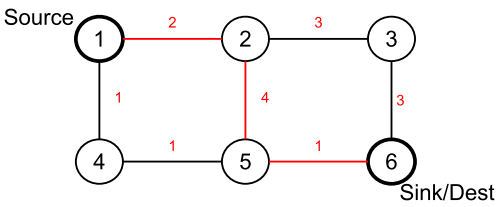


Figure 5.3: An example of how the load-balance routing works

Figure 5.3 illustrates an example of the load-balance routing method. The source and the destination are nodes 1 and 6, respectively. The number of data flows that are currently passing through each link is shown on top of them. It is obvious that the path containing the minimum total number of flows is $[1-4-5-6]$. Therefore, it would be selected for routing the flow.

5.2 Network topologies

We considered *five mesh topologies, which are grid, 2D torus, 3D torus, hypercube, and triangulated planar graph, and two tree topologies, which are fat tree, and k-ary tree.* The first four mesh topologies have been widely used for various high-performance computing applications [12, 18, 43, 44]. The triangulated planar topology can illustrate, for example, a wireless mesh network where each node represents a wireless station and each link represents a connection between two adjacent stations. The fat tree topology is popular in data centers because of its high fault tolerance and good performance. The k-ary tree topology is a simple topology that can be deployed in small-scale networks, such as a university campus network. The next sections present an overview of these topologies and how we used them in the simulations. The information is summarized in Table 5.2.

Topology	**Number of nodes**	**Communication endpoints**
Grid	729 (27×27)	Any switches
2D torus	729 (27×27)	Any switches
3D torus	729 ($9 \times 9 \times 9$)	Any switches
Hypercube	512 (9-dimension)	Any switches
Triangulated planar	729	Any switches
K-ary tree	512 (binary tree of height 8)	Root and leaf switches
Fat tree	720 (24 pods)	Core and edge switches

Table 5.2: Summary of the topologies used in the simulations

5.2.1 Mesh topologies

Grid

Grid is a 2-dimensional topology in which each node is connected to at most two immediate neighbors along each dimension (Figure 5.4a). The properties of an $n \times n$ grid topology are shown in Table 5.3.

In the simulations, we used an 27×27 grid topology with 729 nodes. All switches were considered to have hosts connected to them, and an arbitrary switch was chosen as the gateway to external networks (i.e., the Internet).

Number of nodes	n^2
Node degree	≤ 4
Length of the shortest path between nodes	$\leq 2 \cdot (n-1)$

Table 5.3: Properties of an $n \times n$ grid topology

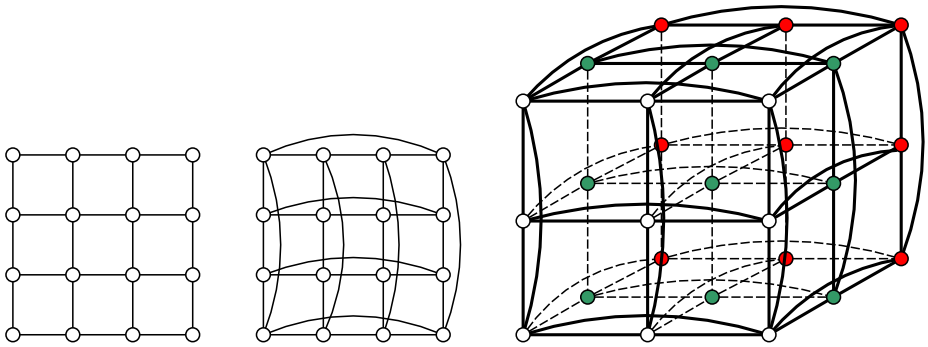
2D torus

2D torus is basically the grid topology with wraparound connections at the edges (Figure 5.4b). An $n \times n$ 2D torus has the properties as shown in Figure 5.4.

In the simulations, we used a 2D torus topology of size 27×27. All switches were considered to have hosts connected to them, and an arbitrary switch was chosen as the gateway to external networks (i.e., the Internet).

Number of nodes	n^2
Node degree	4
Length of the shortest path between nodes	$\leq 2 \cdot \lfloor \frac{n}{2} \rfloor$

Table 5.4: Properties of an $n \times n$ 2D torus topology



(a) Grid topology (b) 2D torus topology (c) 3D torus topology

Figure 5.4: Examples of grid, 2D torus, and 3D torus topologies

3D torus

3D torus is a 3-dimensional topology type in which each node is connected to two neighbors along each of the three dimensions (Figure 5.4c). The nodes at the edges have wraparound connections to the opposite edges. The properties of an $n \times n \times n$ 3D torus are shown in Table 5.5.

In the simulations, a 3D torus topology of size $9 \times 9 \times 9$ was deployed. All switches were considered to have hosts connected to them, and an arbitrary switch was chosen as the gateway to external networks (i.e., the Internet).

Number of nodes	n^3
Node degree	6
Length of the shortest path between nodes	$\leq 3 \cdot \lfloor \frac{n}{2} \rfloor$

Table 5.5: Properties of an $n \times n \times n$ 3D torus topology

Hypercube

Hypercube is an n-dimensional topology that is constructed by creating two identical $(n-1)$-dimensional hypercubes and then adding a link from each node in the first hypercube to the corresponding node in the second one. For

example, a line connecting two nodes is an 1-dimensional hypercube, a square with four nodes is an 2-dimensional hypercube, and a cube with eight nodes is an 3-dimensional hypercube, illustrated in Figure 5.5. The properties of an n-dimensional hypercube are described in Table 5.6.

An 9-dimension hypercube topology was used in the simulations. All switches were considered to have hosts connected to them, and an arbitrary switch was chosen as the gateway to external networks (i.e., the Internet).

Number of nodes	2^n
Node degree	n
Length of the shortest path between nodes	$\leq n$

Table 5.6: Properties of an n-dimensional hypercube topology

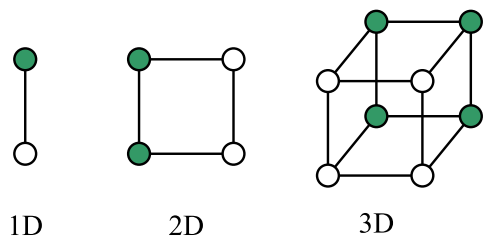


Figure 5.5: Examples of 1D, 2D, and 3D hypercube

Triangulated planar

Triangulated planar topology represents a Delaunay triangulation [14] of a set of points in a plane, in which no point is inside the circumcircle of any triangle in the triangulation. Since we do not consider the switches' geographic location in the simulation, we generated the topology as a mesh topology with each node randomly connecting to 6-8 other nodes, except for the edge nodes. An example of the topology is shown in Figure 5.6. The properties of a triangulated planar topology with n nodes are described in Table 5.7.

In the simulations, a triangulated planar topology with 729 nodes was generated. All switches were considered to have hosts connected to them, and an arbitrary switch was chosen as the gateway to external networks (i.e., the Internet).

Number of nodes	n
Node degree	< 8
Length of the shortest path between nodes	arbitrary

Table 5.7: Properties of a triangulated planar topology with n nodes

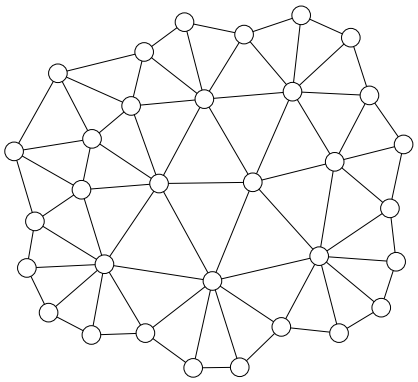


Figure 5.6: An example of the triangulated planar topology

5.2.2 Tree topologies

Fat tree

A fat tree topology [5] typically has three layers: core, aggregation, and edge. The switches in the aggregation and edge layer are divided into k pods. Each pod contains k switches, half of which belong to each layer. Each edge switch connects to all aggregation switches in the same pod. There are $\left(\frac{k}{2}\right)^2$ core switches. Each core switch has one port connected to each of the k pods. The i^{th} port of a core switch is connected to an aggregation switch of the pod i so that each aggregation switch is connected to exactly $\frac{k}{2}$ core switches. The properties of a fat tree with k pods are presented in Table 5.8. Figure 5.7 shows an example of a fat tree topology with 4 pods.

We deployed a fat tree with 24 pods in the simulations. Hosts were considered to connect only to the edge switches of the tree. All core switches of the tree acted as the gateways of the network.

Number of nodes	$1.25 \cdot k^2$
Node degree	k
Length of the shortest path between nodes	≤ 4

Table 5.8: Properties of a fat tree topology with k pods

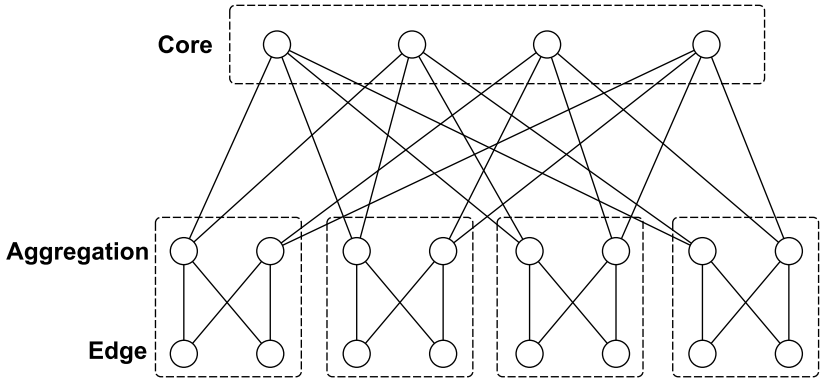


Figure 5.7: Example a three-level fat tree topology with four pods

K-ary tree

K-ary tree is a rooted tree in which each node has no more than k children. When $k = 2$, it becomes a binary tree. A k-ary tree is said to have height h if the distance from its root to its furthest leaf is h. A perfect k-ary tree is a k-ary tree where all leaf nodes are at the same depth and each node has either 0 or k children. A perfect k-ary tree with height h has the properties shown in Table 5.9. Figure 5.8 illustrates an example of a perfect 3-ary tree with height 2.

In the simulations, we used a binary tree with height 8. While real networks rarely have binary-tree topology, binary was chosen for the simulation because it emphasizes the tree-like qualities of the network e.g. in comparison to the fat tree, which is much shallower. Hosts were connected only to the leaf nodes of the tree. The root node of the tree acted as the gateway to external networks.

Number of nodes	$\lfloor (k^{h+1} - 1)/(k - 1) \rfloor$
Node degree	$k + 1$
Length of the shortest path between nodes	$\leq 2 \cdot h$

Table 5.9: Properties of a perfect k-ary tree with height h

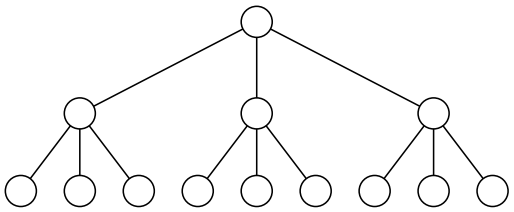


Figure 5.8: Example a 3-ary tree with height 2

5.3 Simulation method

For each simulation, we chose one of the network topologies described above with a specific routing method and measured the number of data flows that passed through the compromised switches in each of the following scenarios:

1. **Single-switch baseline**: one switch is compromised and the attacker does not conduct any topology poisoning attack.

2. **Single-switch tunnel attack**: one switch is compromised and the attacker conducts the single-switch tunnel attack. The tunnels between the compromised switch's neighbors were created differently in each topology. For the hypercube, triangulated planar and k-ary tree topologies, the tunnels were fabricated randomly among the neighbors of the compromised switch. For the grid, 2D torus and 3D torus topologies, we chose the endpoints of each tunnel to be the switches that are on the same dimension. For example, in the grid topology shown in Figure 5.9, where $S5$ is the compromised switch, the tunnels would be created between $S4$ and $S6$ and between $S2$ and $S8$. Such tunnels would attract more traffic flows than the diagonal tunnels like $S2$-$S4$, because the switches in more directions can take advantages of them. Regarding the fat tree topology, if the compromised switch was in the aggregation layer, the tunnels were created between the core switches and the edge switches rather than between the switches of the same core layer or of the same edge layer. This is because the core and edge switches are on the edges of the topology, which makes the tunnels connecting these switches less effective. If the compromised switch was in the core layer or in the edge layer, the tunnels were created randomly among the switch's neighbors.

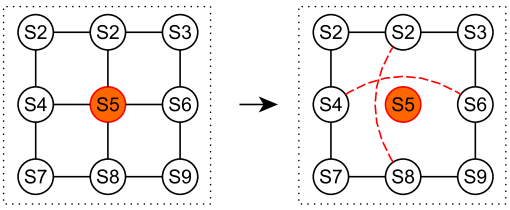


Figure 5.9: Example of how tunnels are created with the single-switch tunnel attack in a grid topology

3. **Two-switch baseline**: two switches are compromised and the attacker does not conduct any topology poisoning attack.

4. **Two-switch tunnel attack**: two switches are compromised and the attacker conducts the two-switch tunnel attack. We simulated the tunnel between the compromised switches by adding a link between them.

5. **Extended two-switch tunnel attack**: two switches are compromised and the attacker conducts the extended two-switch tunnel attack. The tunnels were simulated by randomly adding direct links between the neighbors of the two compromised switches so that the total distance between the tunnel endpoints is maximum. The links between the compromised switches and the neighbors that were used for the attack were also removed from the network.

The first and the third scenario provided the baseline results to evaluate the significance of the attacks. In the last two scenarios, in order to make the simulations simple, we assumed that the relay node exists somewhere in the network but did not consider it when calculating the number of compromised flows. Each simulation was repeated 100 times with the compromised switches randomly selected.

Simulation duration	10s
Total number of data flows	50 000
Flow duration	<1s

Table 5.10: Summary of how traffic was generated in the simulations

The duration of each simulation is 10 seconds. During this interval, we randomly generated a set of 50 000 data flows, each of which contained the following information: the source, destination, starting time, and duration. The source and the destination were randomly selected among the switches

which were either gateways or were directly connected to hosts. The starting time was uniformly randomly chosen within the simulation time. The duration was also uniformly randomly chosen but always less than 1 second. These information are summarized in Table 5.10.

Chapter 6

Results

This chapter presents our observations about the simulations. We noticed that the mesh topologies with similar characteristics produce similar results. Therefore, to make this chapter easier to follow, besides the two tree topologies, we only demonstrate our observations with the results obtained from two representative mesh topologies, which are the 3D torus and triangulated planar topologies. The 3D torus represents the mesh topologies in which all switches are equal (i.e. 2D torus, 3D torus, and hypercube), while the triangulated planar topology illustrates the mesh topologies in which there are center and edges (i.e. grid and triangulated planar). The results of the other topologies can be found in Appendix A.

Figures 6.2, 6.3, 6.4 and 6.5 depict the CDF of the percentage of compromised flows in the 3D torus, triangulated planar, fat tree and k-ary tree respectively. These figures show that the attacks in many cases are much more serious compared to the baselines. However, the number of compromised flows vary significantly in each case. The next sections analyze the factors that bring this variation to the attacks, which are the routing algorithms, network topology, and location of the compromised switches.

6.1 Impact of the routing algorithm

It can be seen from Figures 6.2-6.5 that *the more load-balancing the routing algorithm is, the less harmful the topology poisoning attacks are.* Among the routing methods used in the simulations, the load-balance routing has the most load-balancing capability. The figures show that this routing algorithm is the least favorable one to the attacker. This can be explained by the fact that the tunnels are able to divert more traffic to the compromised switches by shortening the path between switches, but this routing distributes the

traffic over the entire network instead of prioritizing the paths with smallest number of hops. However, *the load-balance routing can only limit the effect of the extended two-switch tunnel attack to some extent.* This is because, by fabricating multiple tunnels, the attack increases the apparent bandwidth between the compromised switches, which increases the share of traffic routed through them.

The figures also demonstrate that, the fully-deterministic routing often diverts more traffic to the attacker than the hop-count routing. The reason for this is that the hop-count routing balances switch load to some extent since its tie breaker rule chooses the path that minimizes the maximum number of flows passing through switches. On the other hand, the fully-deterministic routing always uses the same path between the same endpoints, and, thus, provides no load balancing.

6.2 Impact of the network topology

An observation about the impact of the topologies is that *the effect of the attacks is significant on the mesh topologies, but it is not so big on the tree topologies.* This observation can be explained as follows.

In the k-ary tree topology, there is only one route between any pair of switches. Thus, the tunnels do not affect whether a data flow passes through the compromised switch or not in most cases. It can be seen from Figure 6.5 that the single-switch tunnel attack and the extended two-switch tunnel attack even divert slightly fewer data flows to the attacker than there are in baseline cases. The reason is that the tunnels are not only unable to divert more data flows to the attacker but they also block some data flows going through the compromised switches. This is because the attacker has to sacrifice for the tunnels some legitimate links connecting the compromised switches and their neighbors. For example, in Figure 6.1 where $S2$ is the compromised switch, after creating the tunnel between $S1$ and $S3$, the connection between $S3$ and $S4$ no longer exists.

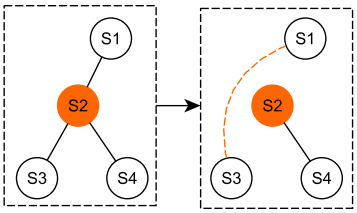


Figure 6.1: An example of how the tunnel is created with the single-switch tunnel attack in a k-ary tree

The fat tree topology, on the contrary, provides multiple paths between switches. However, the tree is so shallow that the length of the longest path in the tree is only 4. As a result, the number of switch pairs whose distance from each other the tunnels can shorten is not big.

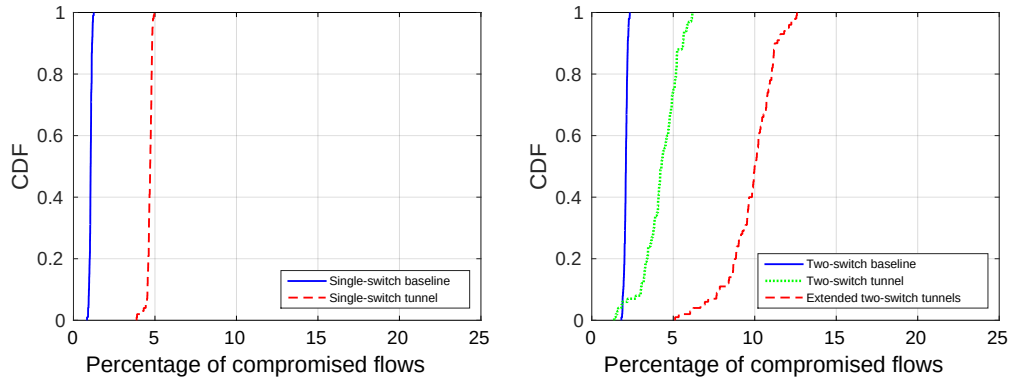
The second observation about the impact of the topologies is that *an arbitrary network structure (i.e. in the triangulated planar topology) is more resistant to the single-switch tunnel attack than a regular one (i.e. in the grid, 2D torus, 3D torus and hypercube topologies).* For example, Figures 6.2 and 6.3 illustrate that the single-switch tunnel attack can divert a large amount of additional traffic to the compromised switch in the 3D torus, but it is not so effective in the triangulated planar topology. This can be explained by the fact that the regular network structure usually provides more shortest paths between switches than the arbitrary network structure. Therefore, the tunnels are likely to affect the shortest paths between more pairs of switches in the regular network structure.

6.3 Impact of the location of the compromised switches

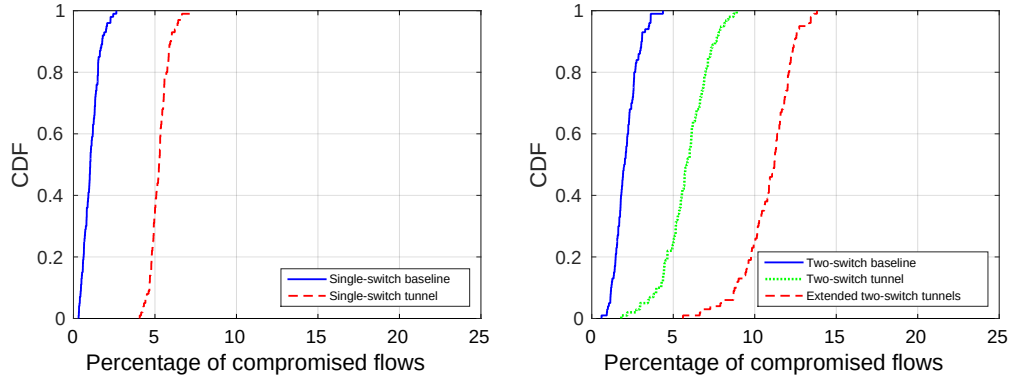
The location of the compromised switches is more important to the attacker in the topologies where there are center and edges (i.e. grid, triangulated planar, and k-ary tree topologies) than in the topologies where all nodes are equal (i.e. 2D torus, 3D torus, and hypercube topologies). This observation is true especially when the routing method does not prioritize some switches over the others, like in the fully-deterministic routing. The switches near the center of the grid or triangulated planar topology, or the switches close to the root of the k-ary tree topology, are likely to get more traffic than the other switches. This is demonstrated most clearly in the scenarios where there is only one compromised switch and in the two-switch baseline scenario. For example, as can be seen in Figures 6.2 and 6.3, the CDFs of the percentage of compromised flows in these scenarios in the 3D torus topology are mostly uniform, while those in the triangulated planar topology vary remarkably depending on where the compromised nodes were placed.

Regarding the two-switch tunnel and the extended two-switch tunnel attacks, *the attacks divert more traffic to the compromised switches when the tunnel endpoints are more distant from each other, especially when the routing strategy prioritizes the paths with the smallest number of hops.* This is because the longer the distance between the two compromised switches is, the more the tunnels can shorten the paths through them. Consequently,

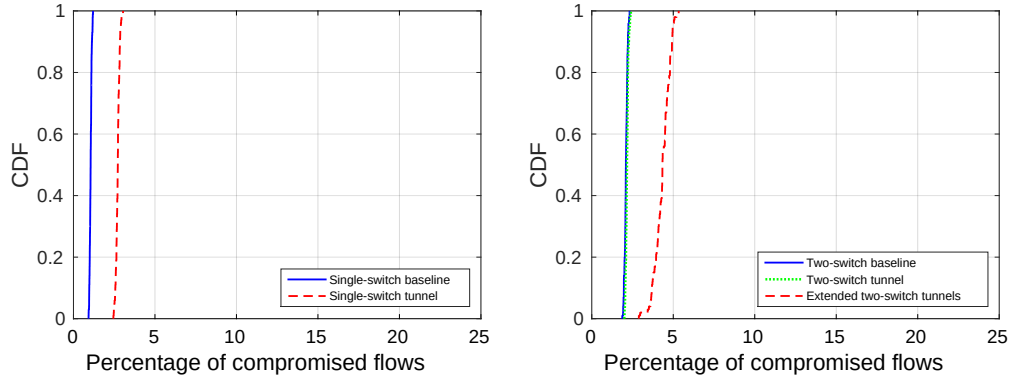
more pairs of switches the tunnel can bring closer, meaning that more data flows get compromised. Figure 6.6 and 6.7 depict the relation between the distance of the two compromised switches and the average increase in the number of compromised flows caused by the two attacks in the 3D torus and triangulated planar topologies, respectively. As shown in the figures, the percentage of compromised flows increases with the distance between the compromised switches. However, when the load-balance routing algorithm is deployed in the 3D torus topology, the increase is appreciable only if the extended two-switch tunnel attack is conducted.



(a) Hop-count routing

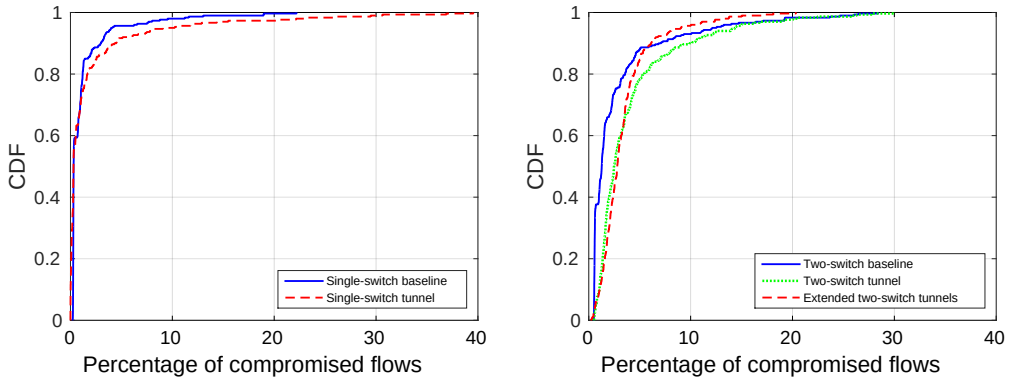


(b) Fully-deterministic routing

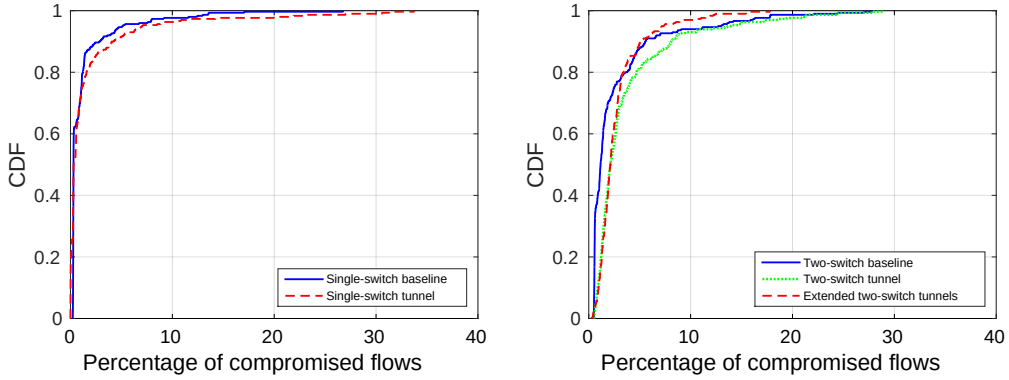


(c) Load-balance routing

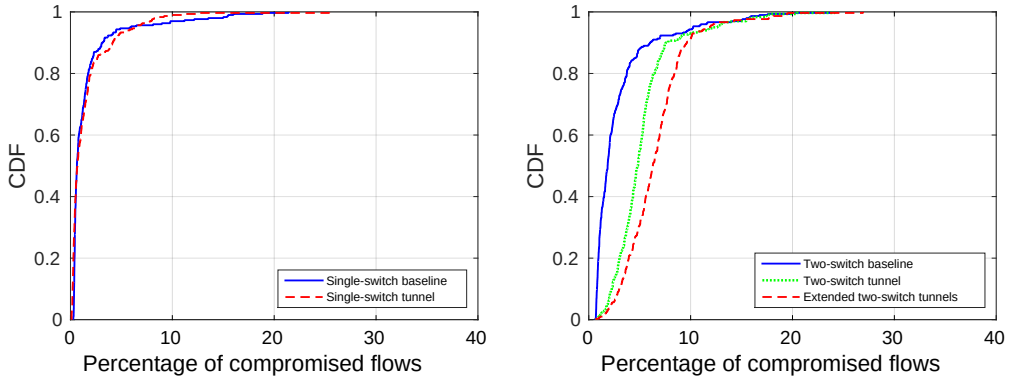
Figure 6.2: CDF of the percentage of compromised flows in the 3D torus topology with different routing algorithms



(a) Hop-count routing

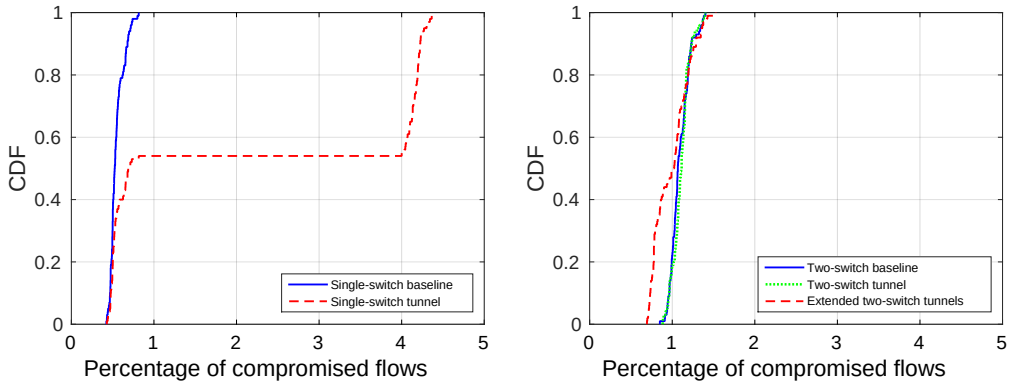


(b) Fully-deterministic routing

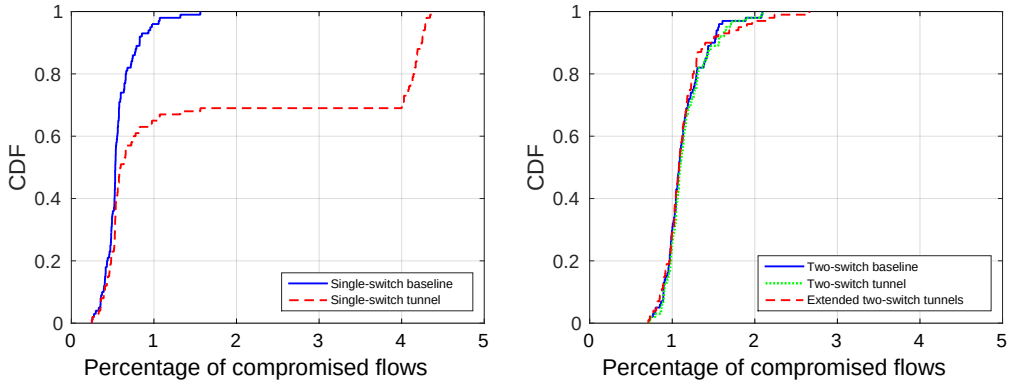


(c) Load-balance routing

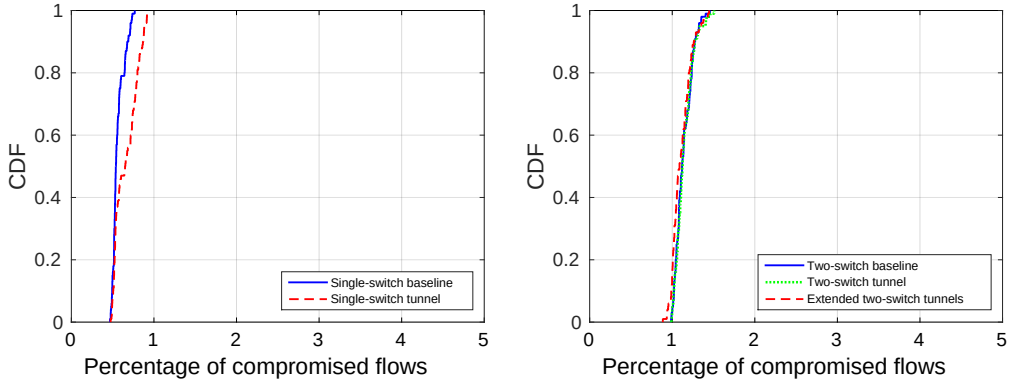
Figure 6.3: CDF of the percentage of compromised flows in the triangulated planar topology with different routing algorithms



(a) Hop-count routing

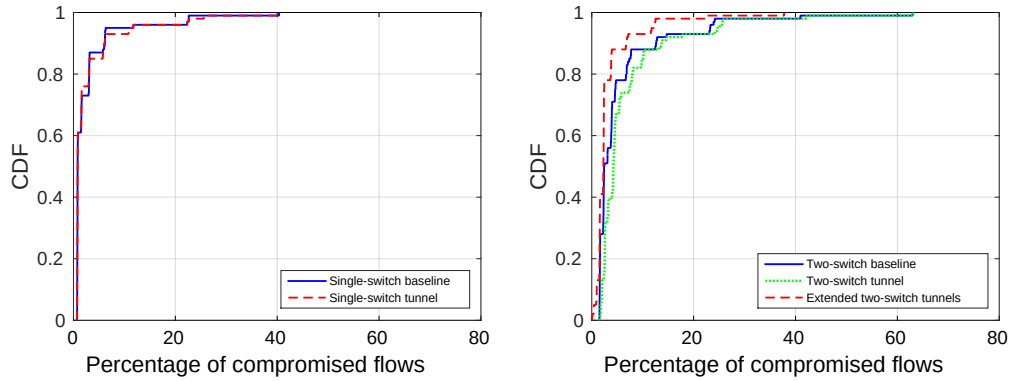


(b) Fully-deterministic routing

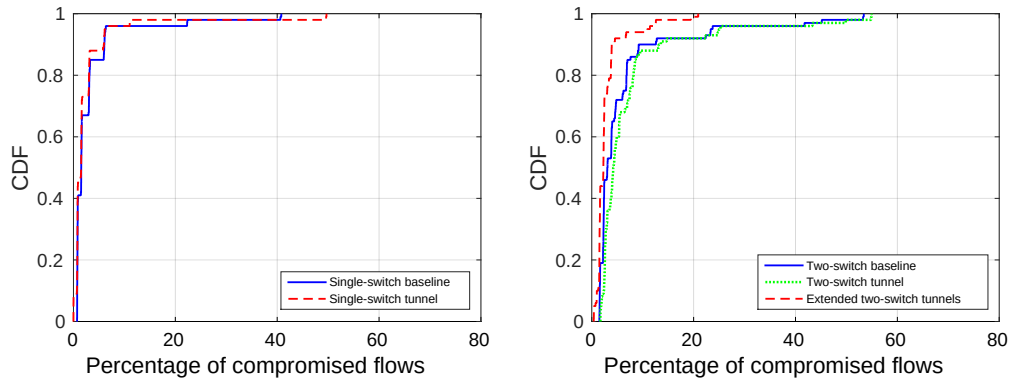


(c) Load-balance routing

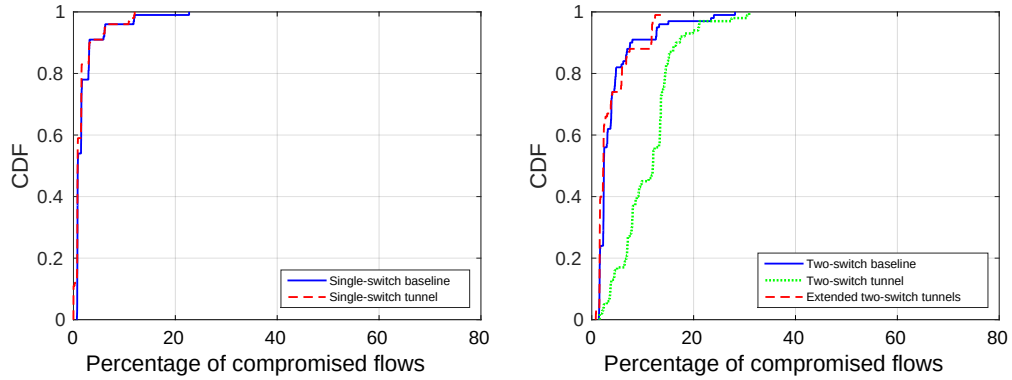
Figure 6.4: CDF of the percentage of compromised flows in the fat tree topology with different routing algorithms



(a) Hop-count routing

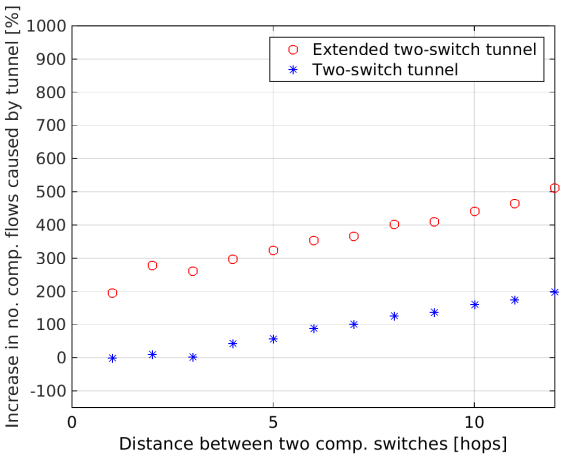


(b) Fully-deterministic routing

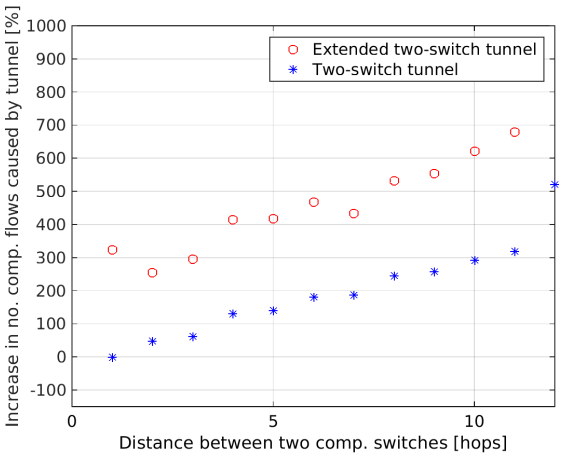


(c) Load-balance routing

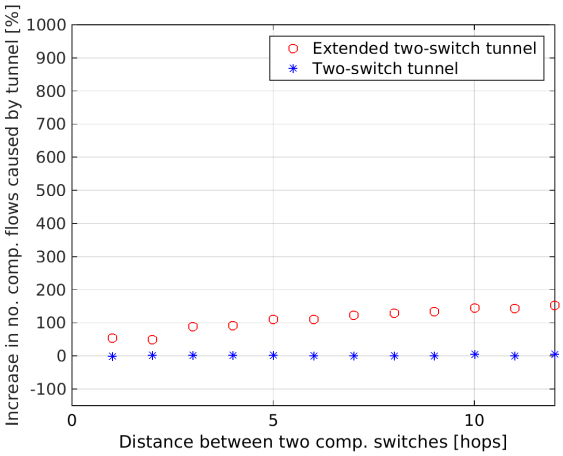
Figure 6.5: CDF of the percentage of compromised flows in the k-ary tree topology with different routing algorithms



(a) Hop-count routing

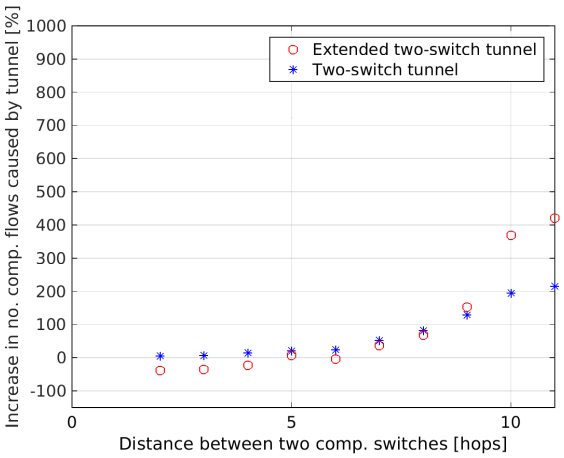


(b) Fully-deterministic routing

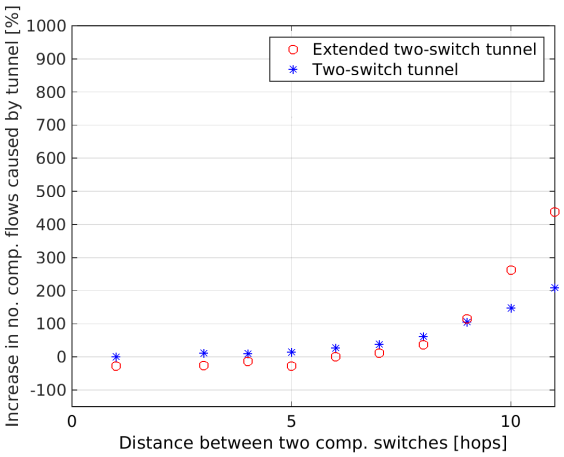


(c) Load-balance routing

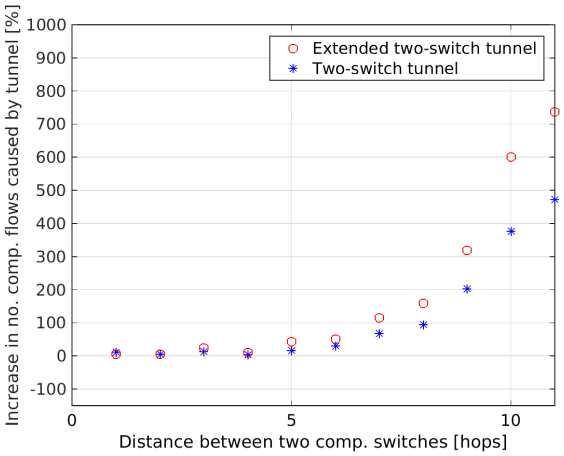
Figure 6.6: Relation between the distance between two compromised switches and the average increase in the number of compromised flows caused by the two-switch tunnel and the extended two-switch tunnel attacks in the 3D torus topology with different routing algorithms



(a) Hop-count routing



(b) Fully-deterministic routing



(c) Load-balance routing

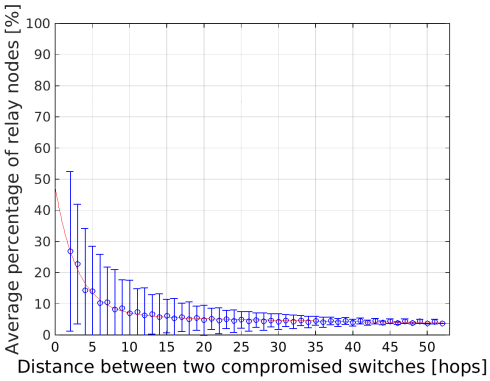
Figure 6.7: Relation between the distance between two compromised switches and the average increase in the number of compromised flows caused by the two-switch tunnel and the extended two-switch tunnel attacks in the triangulated planar topology with different routing algorithms

6.4 Finding relay node

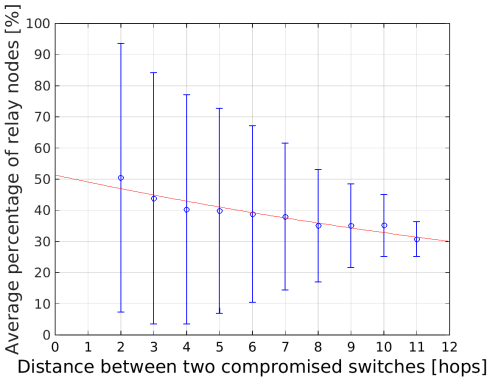
It can be seen from Figures 6.2-6.5 that the two-switch tunnel and extended two-switch tunnel attacks divert the largest amount of traffic to the compromised switches, especially in the mesh topologies. However, finding a relay node that can relay traffic between the compromised switches is crucial for these attacks to succeed. Therefore, our desire is to discover whether the attacks are always feasible in the mesh topologies by examining how many switches in the network can act as the relay node for the attacks. For this purpose, we simulated the five mesh topologies and counted the number of switches from which the route to neither compromised switch goes through a tunnel. Such switches can relay traffic between the compromised switches. The fully-deterministic routing was chosen for the simulations because, out of the three routing algorithms, it provides the most benefit to the attacker. Each simulation was repeated ten times with different link weights. The average number of potential relay nodes for each topology was the final result of the simulation.

The simulations demonstrate that *when the compromised switches are distant from each other, the number of switches that can act as the relay node considerably decreases*. Figure 6.8 demonstrates this conclusion by showing the relation between the average percentage of potential relay nodes and the distance between the two compromised switches in the five mesh topologies.

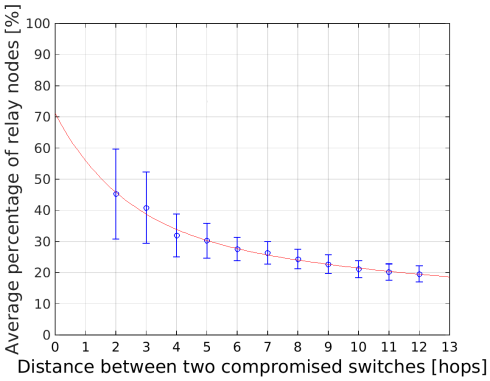
Figure 6.8 also shows that, *among the pairs of switches with the same distance, the number of relay nodes varies remarkably*. This variation is caused by the position of the compromised switches. Figures 6.9 and 6.10 illustrate such variation in the grid and triangulated planar topology, respectively. The red nodes represent the compromised switches, while the green nodes represents the switches that can relay traffic between the red nodes. In both figures, the number of relay nodes in the example on the left is considerably less than that in the example on the right even though the distances between the compromised switches are the same.



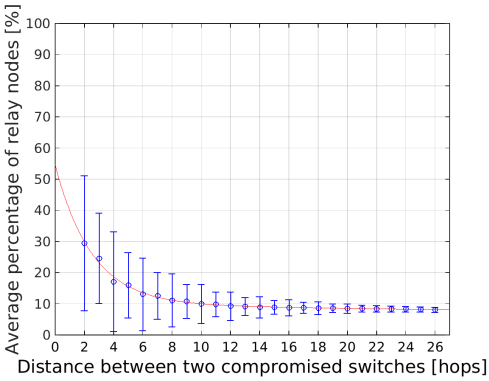
(a) Grid



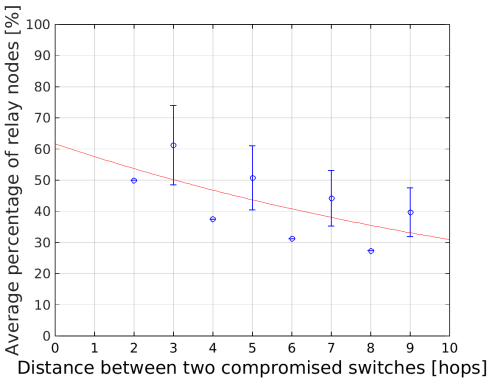
(b) Triangulated planar



(c) 3D torus



(d) 2D torus



(e) Hypercube

Figure 6.8: Average percentage of potential relay nodes vs the distance between the two compromised switches in the mesh topologies with the fully-deterministic routing

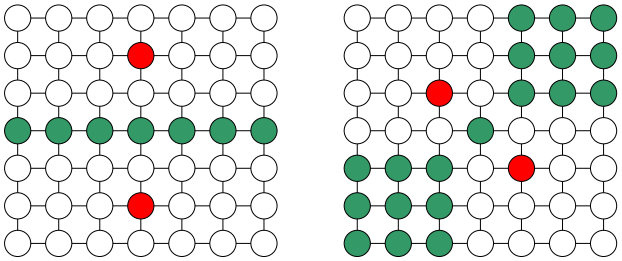


Figure 6.9: Two examples of possible relay nodes in the grid topology with the fully-deterministic routing

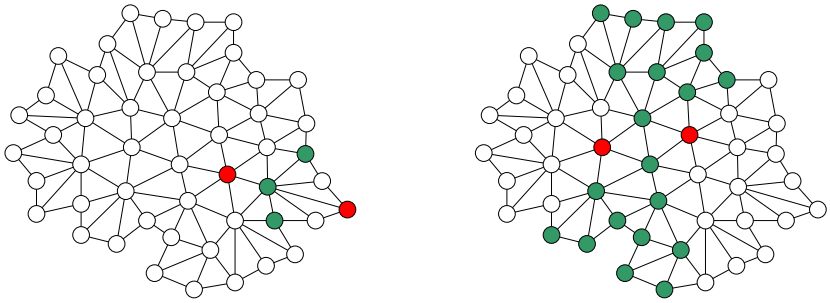


Figure 6.10: Two examples of relay nodes in the triangulated planar topology with the fully-deterministic routing

Chapter 7

Discussion

This chapter first summarizes the results obtained from the simulations. The significance of the results is then evaluated to see whether the topology poisoning attack is serious in SDN. Subsequently, we present some possible ways to detect and prevent the attack. After that, we discuss the advantages and the limitation of the methodology that we used in this research and how we can make it better. In the last section, we present some error sources and open problems in the simulations.

7.1 Summary of the simulation results

In the previous chapters, we presented the three topology poisoning attacks that are initiated from compromised switches and evaluated them in large simulated networks. The simulation results show that the attacks in many cases can divert a large amount of additional traffic to the compromised switches. For example, it can be seen from Figures 6.2 and 6.6 that, in a 3D torus topology with 729 switches, the extended two-switch tunnel attack can increase the number of compromised flows by as much as 600% compared to the baseline, and the percentage of data flows that the attacker can compromise with only two switches can be as high as 24%.

However, the effect of the attacks varies significantly according to the network topology, location of the compromised switches, and routing strategy. The attacks prove to be more serious in mesh topologies than in tree topologies. Furthermore, if the topologies have a boundary and edges, like the grid, planar and tree topologies, the impact of the attacks can depend considerably on the location of the compromised switches. Specifically, the switches near the center of the grid or planar topologies, or the switches that are close to the root of the tree topologies, are likely to get more traffic than

the other switches.

In addition, if there are two compromised switches, then the more distant the compromised switches are from each other, the more traffic the attacker can attract. Nevertheless, the number of possible relay nodes, which the attacker needs to implement tunnels, decreases when the compromised switches are more distant from each other, thus making the attacks less likely to succeed.

The routing also has a big impact on how severe the attacks can be. The amount of traffic that can be diverted to the compromised switches is lower when the routing algorithm used in the network has more load-balancing capability. In fact, among the three routing algorithms used in the simulations, the load-balance routing provides the most limitation to the attacks, followed by the hop-count routing and then the fully-deterministic routing. However, the extended two-switch tunnel attack, which proved to be the most severe attack in most cases, is quite resistant to the mitigating effects of load balancing.

7.2 Significance of the results

As explained above, the discovered topology poisoning attacks can cause serious problems in SDN. Furthermore, the attacks are not specific to the OpenFlow protocol used in the simulations. Similar attacks are feasible in other types of SDN networks. The reason is that they are based on the exploitation of one of the major innovations of SDN — the network-wide view in the controller. Unless the network topology is fixed in the controller, any SDN controller needs to build the view of the entire network by running some link discovery protocol. The protocol would involve sending out discovery packets around the network. Therefore, an attacker would somehow find a way to manipulate the propagation of such packets to perform topology poisoning attacks. For example, the networks that use ForCES [17, 48] or SoftRouter [31], two other southbound protocols for SDN, run a similar protocol as the OpenFlow discovery protocol to discover the links between their forwarding elements (FE) [6]. Instead of sending the LLDP packets, they send the Hello/Probe messages between the FEs to discover links. This protocol would enable an attacker to perform the same topology poisoning attacks as in the OpenFlow networks.

In addition, our simulations were done on a high level of abstraction because the simulations were independent from how the attacker fabricates the tunnels. We assumed that the attacker had already created the tunnels and quantitatively measured the number of data flows passing the compromised

switches accordingly. This means that the simulation results do not depend on a specific link discovery mechanism.

7.3 Detection and prevention of the attacks

It is difficult to detect the discovered attacks, especially the single-switch tunnel attack. One obvious way to detect bogus links is that, for each pair of adjacent switches that the controller detects, the controller sends an OFPT_PACKET_OUT message with an unpredictable data packet inside to one of the switches and observes how the data packet is forwarded. At the same time, the controller also disallows all other switches in the network to forward any packet besides the data packet. If the controller does not receive the data packet back, it knows that the link connecting these switches is a malicious fabrication. This solution, however, is clearly not so practical because the controller has to disable forwarding in the other switches for a period of time. It is also unable to detect the single-switch tunnel attack since no additional switches are required to create the tunnels in that attack.

Another solution is to observe end-to-end delay when an LLDP packet is forwarded on each link. The relay process between two compromised switches might add noticeable latency to the packet forwarding, especially when the compromised switches are distant from each other. The limitation of this solution is that it is hard to monitor the forwarding time accurately, especially when in-band control is used. Moreover, this solution is still not effective against the single-switch tunnel attack.

An observation that can help to detect the attacks is that the attacker has to sacrifice legitimate links from the compromised switches for creating the tunnels. Therefore, if the attacker uses too many neighbors of the compromised switches to perform the attacks and the controller is aware of the lower bound of the switches' degree (i.e. the number of connected ports) in the network, the attacks can be detected. This solution works against all of the discovered attacks, but it requires the controller to know about the minimum number of neighbors for each switch. Furthermore, if the attacker knows this minimum, it can adjust the attack strategy accordingly so that the attacks cannot be detected.

In order to prevent the attack, one possible way is to use Trusted Execution Environment (TEE) to protect the TLS private key of the switches. By doing so, the attacker cannot extract the key from the compromised switches and, thus, cannot spoof connection to the benign controller on behalf of them. The TEE could also be used to implement other trusted functionalities on the switches.

7.4 Evaluation of the methodology

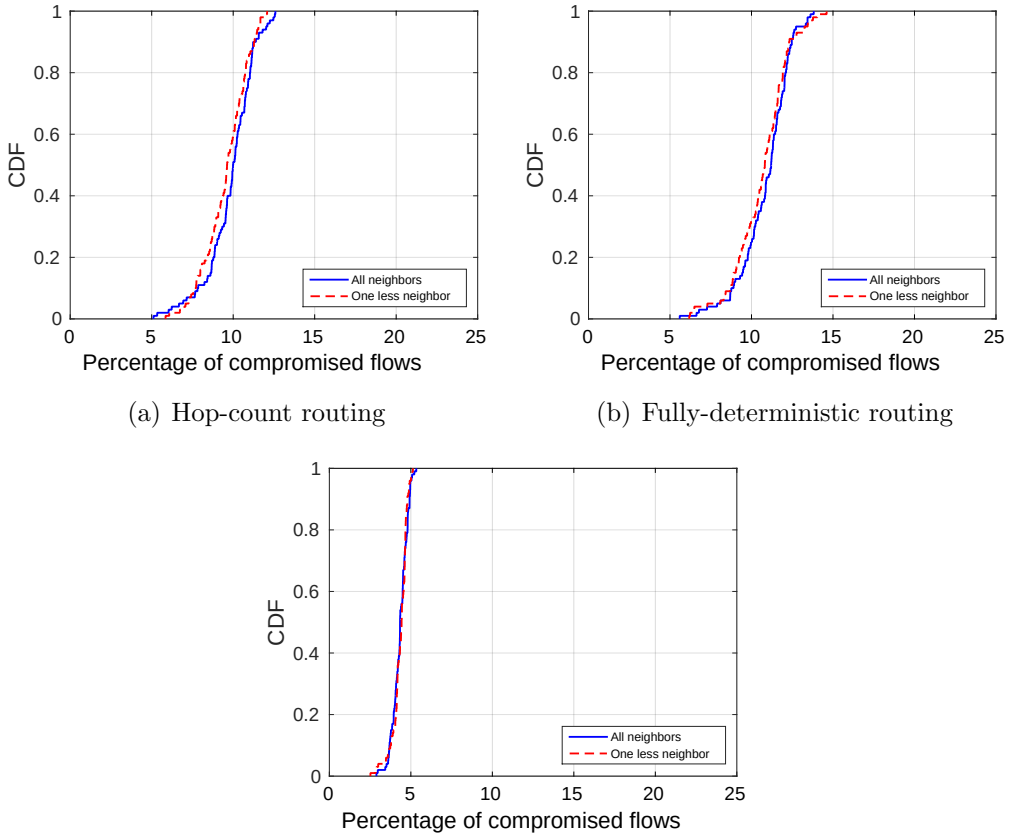
In this thesis, the threat caused by the discovered attacks were empirically assessed with simulations on different kinds of networks. The advantage of this methodology is that it enables us to evaluate the attacks on arbitrary network topologies and routing algorithms. However, the tremendous amount of time that a simulation takes and the high variety of possible positions of the compromised switches in a big network allow us to evaluate the attacks in only a limited number of cases. Furthermore, we cannot demonstrate the reliability of the simulation results with this methodology.

Another methodology we could use is to analytically evaluate the attacks with a mathematical model. The number of compromised data flows can be calculated given the routing strategy and topology information of the network, such as the size of the network and node degree. Although the analytical results are more reliable, it is difficult to evaluate the attacks in complicated networks with this methodology.

Therefore, in the future, we could extend the simulations to evaluate more complex cases of the topology poisoning attack, such as when there are more than two compromised switches. Also, we could conduct analytical analysis on the discovered attacks. By doing so we could verify some of the results that we have already had by comparing them with analytical results in the same environment.

7.5 Error sources and open problems

We have to acknowledge that there are error sources in the simulations. We may have overestimated the effect of the extended two-switch tunnel attack. This is because we did not consider the relay node in the simulations and used all neighbors of the compromised switches for the attack. Thus, we created one tunnel more than what the attacker can on physical switches. However, the difference between the simulation results and what the attacker can actually achieve with the attack should not be significant. Figure 7.1 shows the CDFs of the percentage of compromised flows in the 3D torus topology when all neighbors of the compromised switches are used in the extended two-switch tunnel attack and when one random neighbor of each compromised switch is not used in the attack, respectively. It can be seen from the figure that the difference between these two cases is small.



(a) Hop-count routing

(b) Fully-deterministic routing

(c) Load-balance routing

Figure 7.1: CDF of the percentage of compromised flows in the 3D torus topology when all neighbors of the compromised switches are used in the extended two-switch tunnel attack vs that when one random neighbor of each switch is not used in the attack

Furthermore, our analysis of the simulation results, which is presented in the previous chapter, does not explain some observations. First, the load-balance routing algorithm did not bring the least amount of traffic to the compromised switches in the triangulated planar and k-ary tree topologies. It can be seen in Figure 6.3 that, in the triangulated planar topology, the extended-two switch tunnel attack has the most significant effect when the load-balance routing is used. The reason for this is likely to be that a large number of switches in the topology are on the edges or near the edges. These switches receive only a small portion of the data flows in the network if the routing strategy favors the paths with the least number of hops. The load-balance routing, however, does not prioritize the hop count but the links with

more available bandwidth. Furthermore, the tunnels increase the apparent bandwidth between the compromised switches. Thus, data flows from larger parts of the network will be attracted by the apparent additional bandwidth of the tunnel.

Also, it can be seen from Figure 6.5 that, in the k-ary tree topology, the load-balance routing diverts the most considerable amount of traffic to the attacker, especially in the two-switch tunnel attack. The reason could be as follows. The tunnel created by the attack adds a second path between some switches. The path might not be the shortest one between the switches, so the other routing algorithms might not consider it. However, the load-balance routing strategy distributes traffic over the entire network, making it more likely that the tunnel is selected for routing.

Regarding the hypercube topology, Figure 6.8 shows that the number of possible relay nodes when the distance between the two compromised switches is even is not as high as their number when the distance is odd. There is also no variation in the number of possible relay nodes when the distance is even. Further investigation is needed for us to explain this observation, and to confirm the explanations given to the other phenomena above.

Chapter 8

Conclusion

Network-wide view of the network at a central controller is one of the major innovations of SDN, but it can be poisoned easily. An attacker can manipulate the link discovery service of the SDN controller to fabricate imaginary tunnels in the network. Even though this kind of attacks has been addressed in some previous work, none of them provides a thorough solution to or even analysis of the attacks, especially the ones initiated by compromised switches. In this research, we analyzed topology poisoning attacks caused by the compromised switches against SDN and demonstrated that the attacks are serious in many cases.

We first studied the topology poisoning attacks presented in the literature. It is possible to fabricate an imaginary tunnel between two compromised switches [7, 15] (i.e. the two-switch tunnel attack). In this thesis, we note that a variant of this attack, the extended two-switch tunnel, can help the attacker to create multiple tunnels with only two compromised switches. The number of possible tunnels increases with the number of neighbors of the compromised switches. We also discovered that it is possible to conduct the topology poisoning attack with just one compromised switch. Tunnels can be created between the neighbors of the compromised switch. All of these attacks were shown to be feasible in an emulated network environment that used Mininet for the emulation and OpenFlow as the southbound protocol.

We also evaluated the significance of the attacks in large simulated networks. As our goal is to assess the attacks on a general level, we simulated the attacks on a wide range of network topologies and with different routing strategies. The network topologies that were used in the simulations consist of five mesh topologies, which are 2D torus, 3D torus, grid, hypercube, and triangulated planar, and two tree topologies, which are fat tree and k-ary tree. These topologies were chosen because they either have been widely used in practice or have special characteristics that aid the analysis.

We deployed three different shortest-path routing algorithms in the simulations: hop-count, fully-deterministic and load-balance routing. Each of them represents a class of routing algorithms with specific characteristics.

The simulation results show that the topology poisoning attacks in many cases can divert considerable additional traffic to the compromised switches. Furthermore, the seriousness of the attacks increases according to the number of tunnels that the attacker can fabricate and the distance between the tunnel endpoints. The results also bring insights about how network design and routing policies can help to mitigate the attacks. Specifically, shortening the paths between switches in the network, randomizing regular network structure, or increasing load-balancing capability of the routing strategy can assist in limiting the effects of the attacks.

Bibliography

[1] OpenFlow switch specification. Version 1.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf. [Accessed 09 April 2015].

[2] OpenFlow switch specification. Version 1.5.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf. [Accessed 09 April 2015].

[3] The Shellshock vulnerability. https://access.redhat.com/security/cve/CVE-2014-6271. [Accessed 02 June 2015].

[4] IEEE standard for local and metropolitan area networks– station and media access control connectivity discovery. *IEEE Std 802.1AB-2009 (Revision of IEEE Std 802.1AB-2005)* (Sept 2009), 1–204.

[5] Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center nverifetwork architecture. *ACM SIGCOMM Computer Communication Review 38*, 4 (2008), 63–74.

[6] Ansari, F., and Halpern, J. M. ForCES intra-NE topology discovery. *IETF Draft, draft-ansariforces-discovery-01.txt* (2004).

[7] Antikainen, M., Aura, T., and Särelä, M. Spook in your network: Attacking an SDN with a compromised OpenFlow switch. In *Secure IT Systems*. Springer, 2014, pp. 229–244.

[8] Benton, K., Camp, L. J., and Small, C. OpenFlow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (2013), ACM, pp. 151–152.

[9] Bertsekas, D. P., Gallager, R. G., and Humblet, P. *Data networks*, vol. 2. Prentice-Hall International New Jersey, 1992.

[10] Braun, W., and Menth, M. Software-defined networking using OpenFlow: Protocols, applications and architectural design choices. *Future Internet 6*, 2 (2014), 302–336.

[11] Canini, M., Venzano, D., Peresini, P., Kostic, D., Rexford, J., et al. A NICE way to test OpenFlow applications. In *NSDI* (2012), vol. 12, pp. 127–140.

[12] Choo, H., Yoo, S.-M., and Youn, H. Y. Processor scheduling and allocation for 3d torus multicomputer systems. *Parallel and Distributed Systems, IEEE Transactions on 11*, 5 (2000), 475–484.

[13] Clausen, T., and Jacquet, P. Optimized link state routing protocol (OLSR). RFC 3626, 2003.

[14] Delaunay, B. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk 7*, 793-800 (1934), 1–2.

[15] Dhawan, M., Poddar, R., Mahajan, K., and Mann, V. Sphinx: Detecting security attacks in software-defined networks. In *Proceedings of the 2015 Network and Distributed System Security (NDSS) Symposium* (2015), USENIX.

[16] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[17] Doria, A., Salim, J. H., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and Halpern, J. Forwarding and control element separation (ForCES) protocol specification. RFC 5810, 2010.

[18] Duncan, R. A survey of parallel computer architectures. *Computer 23*, 2 (1990), 5–16.

[19] EugeneNg, Z. A. C. T. Maestro: Balancing fairness, latency and throughput in the OpenFlow control plane. Tech. rep., Rice University, 2011.

[20] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review 38*, 3 (2008), 105–110.

[21] Guha, A., Reitblatt, M., and Foster, N. Machine-verified network controllers. In *ACM SIGPLAN Notices* (2013), vol. 48, ACM, pp. 483–494.

[22] Hong, S., Xu, L., Wang, H., and Gu, G. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *Proceedings of the 2015 Network and Distributed System Security (NDSS) Symposium* (2015), USENIX.

[23] Jou, Y., Gong, F., Sargor, C., Wu, X., Wu, S., Chang, H., and Wang, F.-y. Design and implementation of a scalable intrusion detection system for the protection of network infrastructure. In *DARPA Information Survivability Conference and Exposition* (2000), vol. 2, IEEE, pp. 69–83.

[24] Kandek, W. The GHOST nulnerability. `https://community.qualys.com/blogs/laws-of-vulnerabilities/2015/01/27/the-ghost-vulnerability`. [Accessed 02 June 2015].

[25] Kandoi, R., and Antikainen, M. Denial-of-service attacks in OpenFlow SDN networks. In *Workshop on Security for Emerging Distributed Network Technologies (DISSECT)* (2015), IEEE/IFIP.

[26] Kazemian, P., Chan, M., Zeng, H., Varghese, G., McKeown, N., and Whyte, S. Real time network policy checking using header space analysis. In *NSDI* (2013), USENIX, pp. 99–111.

[27] Khurshid, A., Zhou, W., Caesar, M., and Godfrey, P. Veriflow: verifying network-wide invariants in real time. *ACM SIGCOMM Computer Communication Review 42*, 4 (2012), 467–472.

[28] Klöti, R., Kotronis, V., and Smith, P. OpenFlow: A security analysis. *Proceedings of Workshop on Secure Network Protocols (NPSec). IEEE* (2013).

[29] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al. Onix: A distributed control platform for large-scale production networks. In *OSDI* (2010), vol. 10, USENIX, pp. 1–6.

[30] Kreutz, D., Ramos, F., and Verissimo, P. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (2013), ACM, pp. 55–60.

[31] Lakshman, T., Nandagopal, T., Ramjee, R., Sabnani, K., and Woo, T. The SoftRouter architecture. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networking* (2004), vol. 2004.

[32] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 69–74.

[33] Mizrak, A. T., Cheng, Y.-C., Marzullo, K., and Savage, S. Fatih: Detecting and isolating malicious routers. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on* (2005), IEEE, pp. 538–547.

[34] Moy, J. OSPF version 2. RFC 2328, 1997.

[35] Ornaghi, A., and Valleri, M. Man in the middle attacks. In *Blackhat Conference Europe* (2003).

[36] Perkins, J. H., Kim, S., Larsen, S., Amarasinghe, S., Bachrach, J., Carbin, M., Pacheco, C., Sherwood, F., Sidiroglou, S., Sullivan, G., et al. Automatically patching errors in deployed software. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), ACM, pp. 87–102.

[37] Porras, P., Cheung, S., Fong, M., Skinner, K., and Yegneswaran, V. Securing the software-defined network control layer. In *Proceedings of the 2015 Network and Distributed System Security (NDSS) Symposium* (2015), USENIX.

[38] Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M., and Gu, G. A security enforcement kernel for OpenFlow networks. In *Proceedings of the first workshop on Hot topics in software defined networks* (2012), ACM, pp. 121–126.

[39] Rekhter, Y., and Li, T. A border gateway protocol 4 (BGP-4). RFC 4271, 1995.

[40] Shin, S., and Gu, G. Attacking software-defined networks: A first feasibility study. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (2013), ACM, pp. 165–166.

[41] Shin, S., Porras, P. A., Yegneswaran, V., Fong, M. W., Gu, G., and Tyson, M. FRESCO: Modular composable security services for software-defined networks. In *Proceedings of the 2013 Network and Distributed System Security (NDSS) Symposium* (2013), USENIX.

[42] Shin, S., Yegneswaran, V., Porras, P., and Gu, G. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 413–424.

[43] Shu, R., and Du, D. H. Improved hypercube topology for multiprocessor computer systems, 1992. US Patent 5,170,482.

[44] Torresen, J., Mori, S.-i., Nakashima, H., Tomita, S., and Landsverk, O. Parallel back propagation training algorithm for mimd computer with 2d–torus network. In *Proceedings of 3rd Parallel Computing Workshop (PCW94)* (1994).

[45] Wang, F., Vetter, B., and Wu, S. F. Secure routing protocols: Theory and practice. Tech. rep., North Carolina State University, 1997.

[46] Wang, H., Xu, L., and Gu, G. OF-GUARD: A DoS attack prevention extension in software-defined networks.

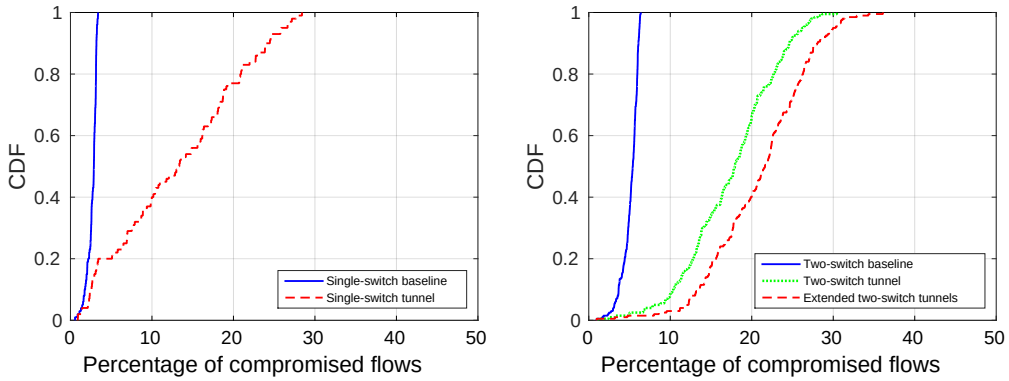
[47] Yan, Z., and Prehofer, C. Autonomic trust management for a component-based software system. *Dependable and Secure Computing, IEEE Transactions on 8*, 6 (2011), 810–823.

[48] Yang, L., Dantu, R., Anderson, T., and Gopal, R. Forwarding and control element separation (ForCES) framework. RFC 3746, 2004.

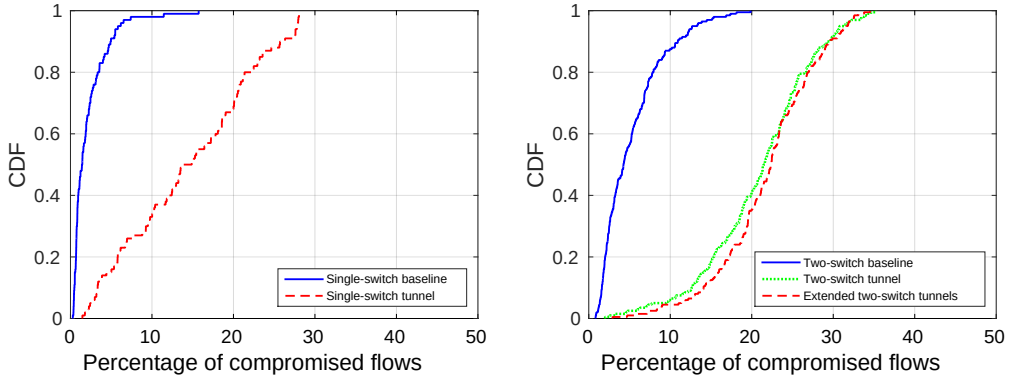
Appendix A

Simulation results

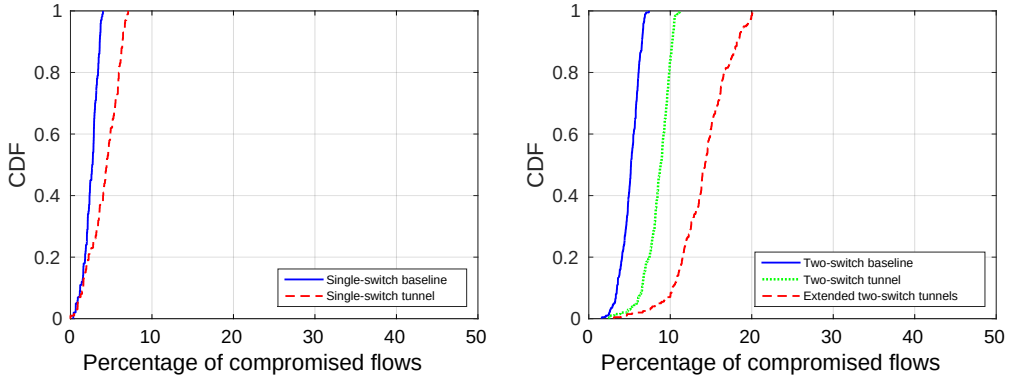
Figures A.1, A.2 and A.3 illustrate the CDF of the percentage of compromised flows that we obtained from the grid, 2D torus and hypercube topologies, respectively.



(a) Switch-optimized shortest path routing

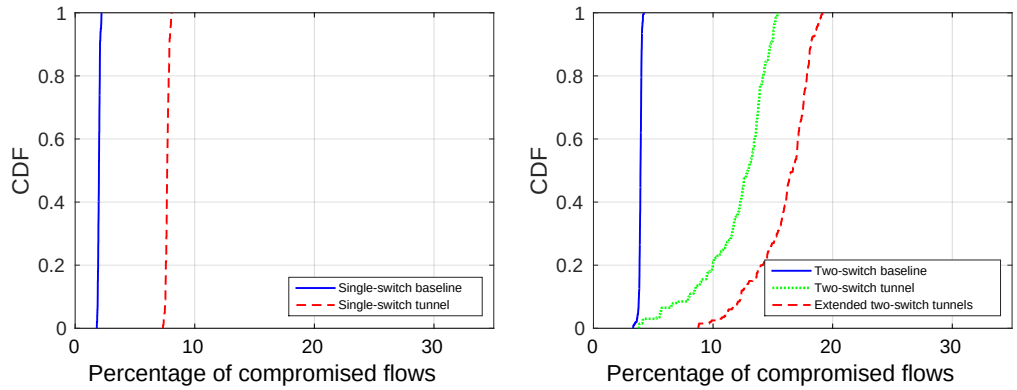


(b) Deterministic shortest path routing

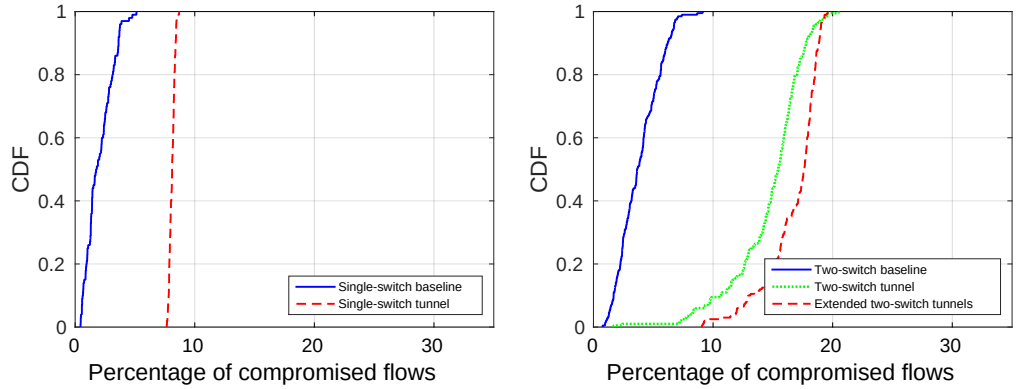


(c) Load-balancing routing

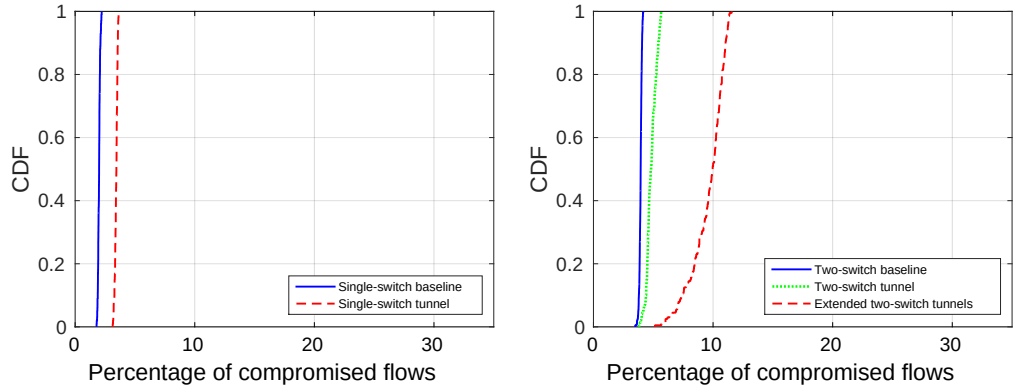
Figure A.1: CDF of the percentage of compromised flows in the grid topology with different routing algorithms



(a) Switch-optimized shortest path routing

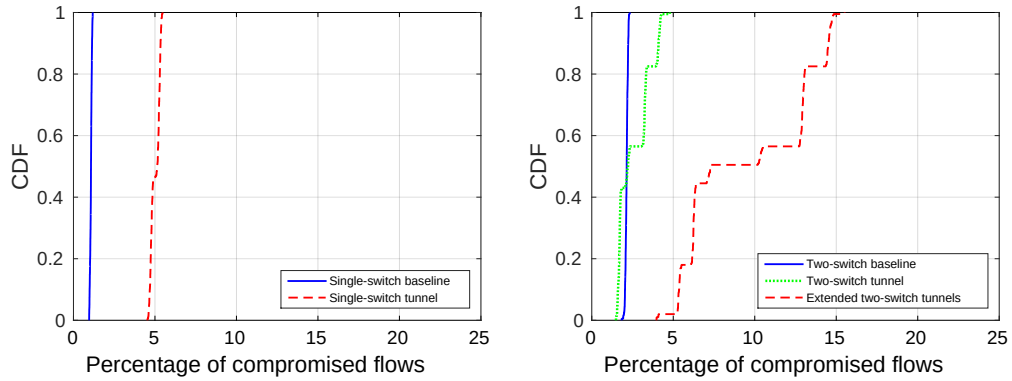


(b) Deterministic shortest path routing

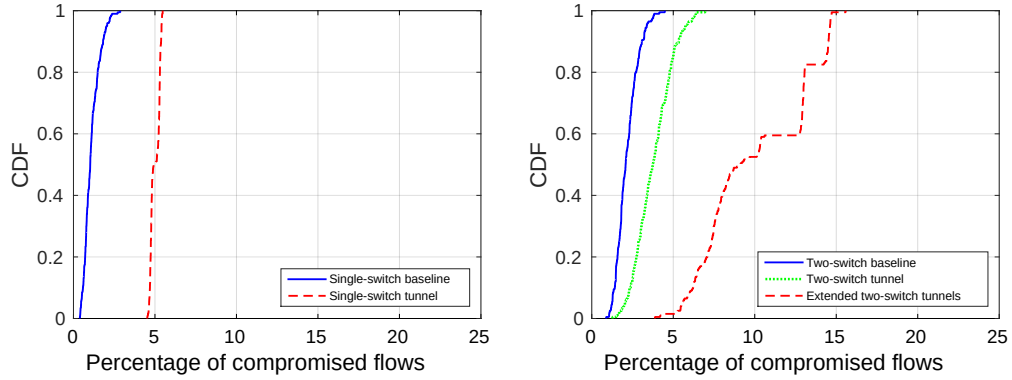


(c) Load-balancing routing

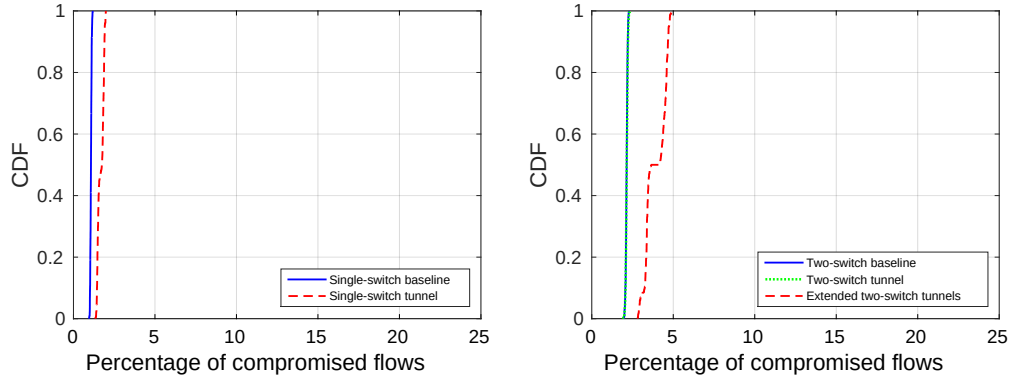
Figure A.2: CDF of the percentage of compromised flows in the 2D torus topology with different routing algorithms



(a) Switch-optimized shortest path routing



(b) Deterministic shortest path routing



(c) Load-balancing routing

Figure A.3: CDF of the percentage of compromised flows in the hypercube topology with different routing algorithms