Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 62 (2015) 228 – 235

Procedia Computer Science

The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

# Impact on SDN Powered Network Services under Adversarial Attacks

Quamar Niyaz[a], Weiqing Sun[b], Mansoor Alam[a]

[a]*Department of Electrical Engineering & Computer Science*
[b]*Department of Engineering Technology*
*The University of Toledo, Toledo, OH-43606, USA*
*{quamar.niyaz, weiqing.sun, mansoor.alam2}@utoledo.edu*

**Abstract**

Logically centralized nature of the controller in Software-defined Networking (SDN) makes it vulnerable to various adversarial attacks. These attacks have capabilities to degrade the performance of the managed network or bring it down in the worst case. For this reason, a large-scale deployment of SDN needs evaluation of the impact of adversarial attacks on network services. In this work, we implement various attacks in SDN and analyze their impact on the performance of web-based services running over it. Prior to that, we briefly discuss different kinds of vulnerabilities and threats in SDN. We consider the connection set-up latency and loss, for web-client requests, as metrics for the evaluation. We observe a significant degradation in the performances of web services, in terms of response time and availability, in the presence of implemented attacks.



Peer-review under responsibility of organizing committee of The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

*Keywords:* Software-defined networking; security; adversarial attacks; performance evaluation.

## 1. Introduction

Software-Defined Networking (**SDN**), originated from research in academia, now became a significant topic of discussion among the network service providers, operators, and equipment vendors. In the SDN architecture, the control logic and the data logic are decoupled from network devices, unlike the traditional network architecture. The control plane is consolidated to a *logically* centralized controller. It keeps the global view of the entire network and makes it *programmable* using software applications running on top of it. The network devices termed as *switches*, become flexible in functionality. They can be programmed to function as switches, routers, or firewalls using different network applications implemented on the controller platform[7].

SDN offers several advantages to the network operators, administrators, and researchers over traditional networks. It makes distributed network decision problems, e.g. routing, to the logically centralized one. In addition, it provides

* Weiqing Sun. Tel.: +1-419-530-3273; Fax: +1-419-530-3068
*E-mail address:* weiqing.sun@utoledo.edu



Peer-review under responsibility of organizing committee of The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)
doi:10.1016/j.procs.2015.08.444

an abstraction of the managed network to the controller applications. These two features make the network management easier in an SDN environment. A centralized network policy, implemented without any low-level network devices configurations, can manage the network efficiently. SDN brings an advancement towards network innovation. A researcher can implement a *prototype* application on the controller and observe its impact on a large portion of the network without affecting the production traffic. It aims to reduce the timespan of transition of a prototype to the real-world deployment. This is in contrast to the traditional *ossified* network architecture where it takes a decade in this transition[12]. Within a couple of years from its inception, SDN has found many use cases in the networking world including traffic engineering, mobility and wireless, network monitoring, and data-center networking[7].

SDN is envisaged as a great promise for the future Internet. Nevertheless, it has to address several issues to make it widely accepted for the real-world deployment. Security is one of the most important issues that has to be taken into consideration at first hand. This overemphasis on the security comes from the fact that in the last few decades, we have witnessed an unprecedented increase in the system vulnerabilities and threats. Various network security applications and frameworks are proposed using SDN. However, the security of the SDN itself has never been paid attention in much detail as compared to other research work in this area. A few literatures have discussed various threat vectors and their mitigation in a limited domain. We discuss them briefly in Section 5.

In contrast to the previous work, we believe that it is also important to evaluate the impact on the network services running over the SDN managed networks under adversarial attacks. Are the network providers/operators able to meet the customers Service Level Agreement (**SLA**) in the presence of the attacks? Availability of the network services to the customers and their response time are the critical components in any network service SLAs. With this motivation, we aim to model various attacks specific to the SDN and evaluate the impact on the performance of the network services.

Towards this end, our paper is organized as follows. We present an overview of the SDN architecture in Section 2. Section 3 discusses various vulnerabilities in SDN. In Section 4, we discuss the attacks that we choose, methodologies to implement them, and evaluate the performance of network services under those attacks. Finally, we conclude our paper with an insight for the future work in Section 6.

## 2. Overview of SDN Architecture

The controller is the core of the system in SDN managed networks. It runs on a dedicated or distributed server(s), however, logically centralized. Every switch is connected to the controller through a TCP connection. The connection is either TLS/SSL encrypted or plain and forms the control channel of SDN. The controller communicates with switches using an open API. Currently, OpenFlow is a *de facto* standard protocol which provides an interface for the controller-switch communication[19].

Flow tables reside inside every switch for handling network flows. A flow in SDN is defined as a set of packets with similar values for certain header fields. These flow tables consist of flow rules that are installed by the controller based on the network applications running over it. Flow rules can be installed by the controller either in reactive mode or proactive mode[20]. In the reactive flow installation, whenever a packet arrives for an upcoming flow at a switch, a match is performed against the flow rules inside the flow tables. If there is a match found inside the tables then an action is taken accordingly as specified in the tables corresponding to the matched rule. These actions can either be *forward, drop or set*. If a miss happens for the flow, a `packet_in` event message is forwarded to the controller from the switch. This message encapsulates the packet headers for the missed flow. The controller examines the packet headers for the flow in the message. Based on the network applications policies, the controller either sends a `packet_out` or a `flow_mod` message to the switch for the flow.

In response to `flow_mod` message, flow rules are installed on switches for the upcoming flow. The subsequent packets for the same flow can then be directly processed inside the switches instead of forwarding them again to the controller. Figure 1 shows the traffic flow in the SDN architecture in reactive mode. In order to manage the memory size of flow tables, flow rules expire after certain timeout events. For a `packet_out` message, no flow rule is installed inside the flow tables. The packet is forwarded to a single port or broadcast to all ports of the switch depends on the network policies.

On the other hand, in case of proactive flow installation, the controller pre-installs flow rules for all kinds of flows that could come into the switches instead of reacting to every packet.

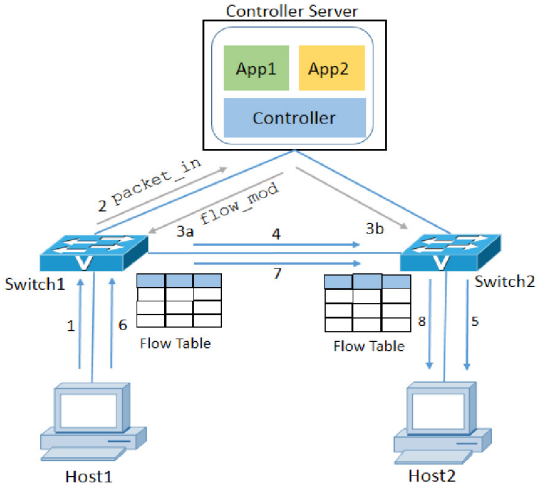


Fig. 1: Basic SDN architecture and traffic flow in reactive mode flow installation [13].

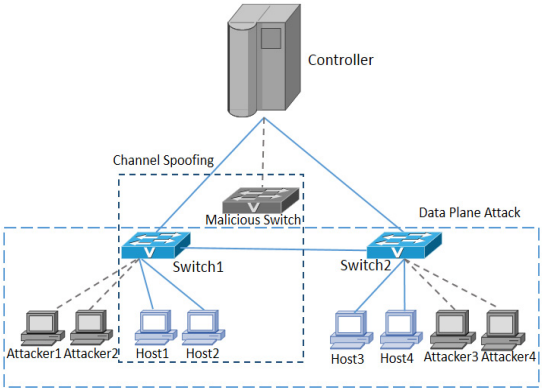


Fig. 2: Various attack scenarios implemented for web service performance analysis. The data plane attack includes controller, switch1, switch2, web hosts, and attacker hosts. The channel spoofing attack includes controller, switch1, malicious switch, and web hosts of switch1.

## 3. Security Challenges in SDN

Security challenges in SDN are more threatening for the managed networks as compared to the traditional networks. In a traditional network, targets are only a few servers or a portion of the network. On the contrary, if an SDN is compromised by some adversaries the whole managed network will be endangered. Various vulnerabilities and threats have been discussed in [1,8,15]. However, for the sake of completeness, we provide a brief discussion of vulnerabilities and threats in SDN from the following aspects:

### *3.1. SDN Management Threats*

SDN management includes set-up of switches and controllers, installation of network applications, administering controllers, managing credentials between various entities of the SDN. Albeit, the complexity of compromising SDN management is high as it needs authentication. However, If it gets compromised, the effect on the network is severe. Various security threats can occur in the management. A malicious administrator can misconfigure network policies which might degrade the performance of the managed network or can bring it down in the worst case. An unauthorized network application, comes from a third-party, could contain a malicious code. The restriction on applications to access the controller server or the underlying managed network is very less in SDN. Therefore, a malicious code can manipulate various configurations on the controller server, shut it down or crash it down once the malware gets installed on it. Installation of faulty or malicious switches can deter the SDN to perform intended tasks in accordance to the network policies.

### *3.2. Control Plane Threats*

Control plane includes network applications policies and signalling traffic between switches and the controller for administering the managed network. In SDN, network applications policies can contradict each other. Lack of their prioritization leads to an unexpected behaviour of the managed network. For example, the *set* action modifies the flow rules inside the flow tables. This action can be used for *honeypot, quarantine a host, or captive portals* application. However, a malicious application can exploit this action to bypass the firewall policies implemented by other applications through modification of flow headers.

According to the OpenFlow specification, the communication channel between the controller and the switch may be TLS/SSL encrypted or plain. It is observed that many switch vendors and controllers lack in the implementation of encrypted channels [1]. They adopt a plain TCP-based communication channel. It is not a concern for secure networks.

However, if the signalling traffic is carried out via an insecure channel, a man-in-the-middle attack can be launched easily. It can happen in the case of Software-Defined Mobile Networks (**SDMN**), campus Wi-Fi networks, or traffic passes through adversarial networks.

Furthermore, a malicious switch with the datapath identity of a genuine switch can cause disconnection of the genuine switch from the network and can lead to the disruption of the traffic from the switch[4]. Apart from that, a faulty malicious switch can generate too many fake `packet_in` requests to the controller to consume the controller resources. This makes the controller less available to handle the genuine flow requests.

### *3.3. Data Plane Threats*

End user devices, switch flow tables, and the traffic passing through them form the data plane. As mentioned in Section 2, for a flow table miss event, the packet is forwarded to the controller for flow rule installation in the switch. Therefore, the response time of the first packet in a flow is, in general, longer as compared to the subsequent packets of the flow in SDN. This characteristic of SDN helps an adversary from the data plane to fingerprint it[17]. In addition, it helps adversaries to find out flows for which there is no flow rule installation in the flow tables and the controller sends `packet_out` messages to handle them. The adversaries can send these flows to overwhelm the controller. These fake flows keep the controller busy in handling them and occupy the memory of a switch in buffering them as well until they get responses from the controller. This leads in the degradation of the network performance.

## 4. Attacks Implementation and Impact Analysis

In our attack analysis experiments, we implemented those attacks for which there is no requirement of any kind of authentication to access an SDN system. From this perspective, we choose the data plane attack and the control channel spoofing. In the first attack, we send fake requests, from adversaries in the data plane, to keep the controller and switches busy to handle them and cause delay and loss in flow set-up of genuine traffic. In the second attack, a malicious switch spoofs the identity of a genuine switch and causes it to be disconnected from the controller. Figure 2 shows our overall attack scenarios for both cases. We evaluate the impact of these attacks on connection set-up latency and loss of web clients requests to servers. With the growth of web-based businesses, the network operators must confirm various service-level agreements (**SLA**) to their web customers. *Service availability* and *response-time* are two of the critical metrics in any SLA. We account these metrics by measuring the connection set-up latency and loss. In the following subsections, we discuss these attacks implementation and their impact on web services.

### *4.1. Data Plane Attack*

We use `mininet` emulator[9] and `POX` controller[2] to implement the data plane attack. Mininet is an SDN emulator, implemented using network namespace feature of Linux and Open vSwitch. We can create as many virtual hosts and OpenFlow based switches as we need on a single host using `mininet`. The virtual hosts can emulate all network functionalities and run various network services independent of other virtual hosts. Switches can be connected to a controller running either on the same host or a remote host. Although `mininet` has some performance fidelity issues while emulating a large number of virtual hosts and switches and developed mainly for rapid-prototyping of SDN application, we can get an insight about the attack impacts on the services. `POX` is a Python-based controller that provides a platform for rapid deployment of network applications for SDN. We modified the `l2_learning` application, a self-learning ethernet switch module available in `POX`, for our experiments. The modified application installs flow rules inside the flow tables of switches for web-based traffic only. It generates `packet_out` event for other traffic. The flow timeout is set to $15s$. If a flow set-up request does not arrive at switches within this timeout period, the flow rules evict from flow tables. As a genuine traffic, a client sends a web request to the server after a regular interval. We set the interval of request less than the timeout of flow rules in switches in one case and greater than it in another case. We use `curl` utility at clients side to measure the response time of the server[3]. We set maximum connection timeout to $60s$ for a client request to a server. After this timeout, the request is considered as lost.

We carried out our experiments on virtual machines (VMs) created over VMWare ESXi host. This host runs on Dell PowerEdge 2950 server with Intel Xeon CPU E5420 @ 2.50 GHz and 16GB RAM. There are 8 CPU cores

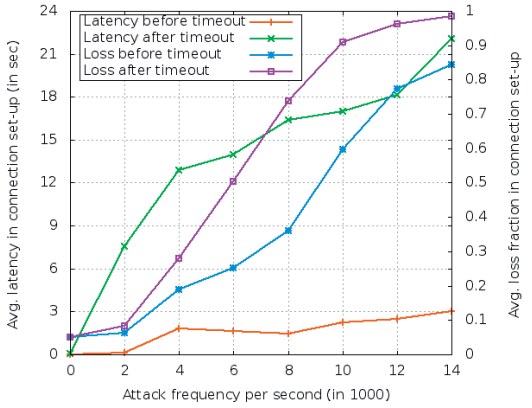


Fig. 3: Average connection set-up latency and loss when all hosts reside in the same network.

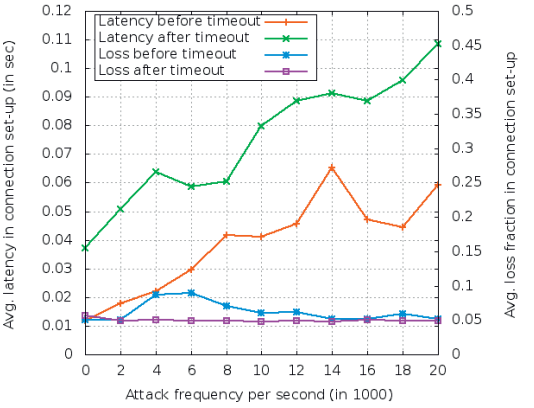


Fig. 4: Average connection set-up latency and loss when web hosts and adversaries reside in different networks.

with 2.493 GHz. We run the same set of experiments on four VMs, each with configuration of 4GB RAM and 4 core CPUs. For our final result, we take the average of all runs. On each VM, we run the `POX` controller with the modified `l2_learning` module. We created 2 switches and virtual hosts for web hosts and adversaries using `mininet`, which are in turn connected with the controller. The link-speed and delay between controller-to-switch and switch-to-switch are $10Mbps$ and $1ms$. However, we keep the link-speed and delay between host-to-switch $1Mbps$ and $1ms$, respectively. In order to have a good estimation of the attack, we connected 10 web client-server pairs with switches. These clients send 20 requests to their corresponding server after a regular interval as mentioned above. We consider `ping` flooding for the adversarial attacks. The adversarial hosts are part of the network and work in a distributed manner as in the case of botnets. An adversary sends `ping` traffic to its peer adversary. We varied the `ping` frequency interval during our experiments and studied the impact on web services. We consider four attack scenarios based on the locations of the client-server (web hosts) and adversaries. These scenarios are as follows:

#### *4.1.1. Intra-switch communication on the same network*

Web hosts and adversaries reside on the same network. As an example in Figure 2, all hosts are connected to *switch1*. We observe from Figure 3 that connection set-up latency is quite high when client requests are sent after the timeout period of flow rules in the switch as compared to when they are sent before the timeout period. In the former case, the connection set-up latency increased upto $22s$ with the attack frequency of $14kps$, where in the later case it went upto $3s$. The loss fraction in both cases increases with increased attack frequencies. However, in the first case, it is higher as compared to the second case, reached upto 98%. The reason for the high loss rate, still in the case when requests are generated before flow rules timeout period in the switch, is due to the large number of `ping` flooding packets from adversaries cause memory buffer overflow in the switch for queuing the incoming packets from the hosts.

#### *4.1.2. Intra-switch communication on different networks*

Web hosts and adversaries reside on different networks. As an example in Figure 2, web hosts and adversaries are associated with *switch1* and *switch2*, respectively. As is evident from Figure 4 that the attack impact is insignificant. We observed that the loss remains around 5% most of the time and the latency is low even with increased attack frequencies in both cases. It went upto $0.11s$ and $0.06s$ for requests generated after and before the timeout, respectively. The reason for less impact is that only the controller and the adversaries side switch is affected by the attack. Flow tables and memory buffers for queuing the incoming packets in the switch and the control link from the switch to the controller on the web hosts side did not get overwhelmed with unwanted `packet_in` requests of the `ping` floods.

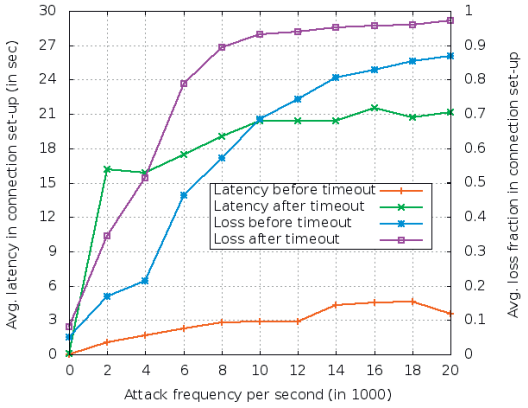


Fig. 5: Average connection set-up latency and loss when web hosts reside on the same network and adversaries located on different networks.

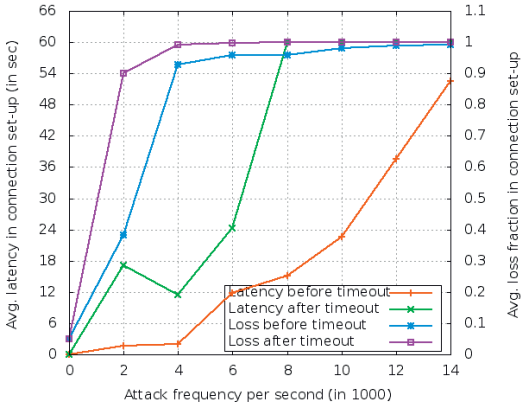


Fig. 6: Average connection set-up latency and loss when web hosts and adversaries span on different networks.

#### *4.1.3. Intra-switch and inter-switch communication*

In this scenario, web hosts are associated with *switch1* while adversaries are associated with both switches as shown in Figure 2. We observed from Figure 5 that the result for this scenario is similar to the first attack scenario with less severity. An explanation for this is as adversaries are distributed, flow tables and memory buffers in web hosts side switch and the control link of it with the controller are less occupied with flow requests as compared to the first attack scenario.

#### *4.1.4. Inter-switch communication*

Web hosts and adversaries both have inter-switch communication. As an example in Figure 2, clients are associated with *switch1* and servers are associated with *switch2*. Adversaries are associated with both switches. As is evident from Figure 6, the attack is the most severe of all. This attack consumes the controller resources, overflows switches memory buffers and flow tables, and overwhelms all control links. Connection set-up latencies increased upto $52s$ even in the case when requests are sent before the flow rule timeout of switches which gives a notion that flow rules are evicted due to large `packet_in` requests for adversarial DoS flows. Connection loss saturation occurred at only $6kps$ when the requests are sent after the flow rule time out in switches.

### *4.2. Control Channel Spoofing*

We exploit the switch datapath identity vulnerability in the attack[4] as mentioned in Section 4. With the lack of implementation of TLS/SSL encryption in the control channel, an adversary can easily find out information about switches in the network and use this information to spoof switches in the SDN managed network. To implement this attack, we ran an OpenFlow-based open vSwitch on a host and connected this host with the `POX` controller on a remote host. We used the same modified `l2_learning` module with the timeout interval of $30s$. We created 20 virtual hosts using network namespace and connected them with the switch. These virtual hosts form client-server pairs for web application. We performed the same connection set-up latency and loss experiments as we did previously. We ran another OpenFlow-based switch to function it like a malicious switch on another host. When the switch is connected with the controller, we establish a connection for the malicious switch, with the same IP address of the genuine switch, with the controller. This causes disconnection of the genuine switch and connection of the malicious switch with the controller, alternatively. Therefore, end devices that were associated with genuine switch suffer a connection disruption in the network. We automated the connection-disconnection set-up of the malicious switch with the controller after a specified interval. We varied the interval from $2s$ to $22s$. The genuine switch automatically attempts to connect with the controller upon disconnection. Clients generate requests at an interval greater than and less than the flow-rule timeout period in the switch. Figure 7 and 8 show the result for connection set-up latency

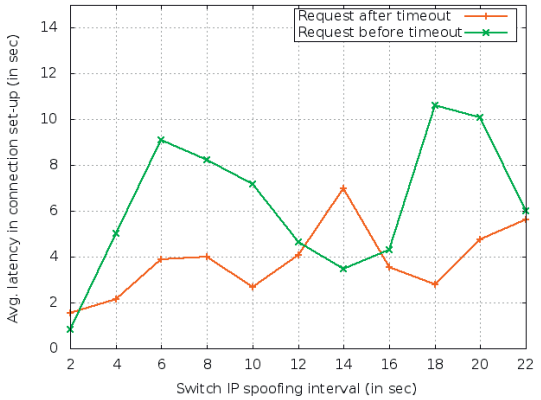


Fig. 7: Average connection set-up latency with varying spoofing interval of the malicious switch

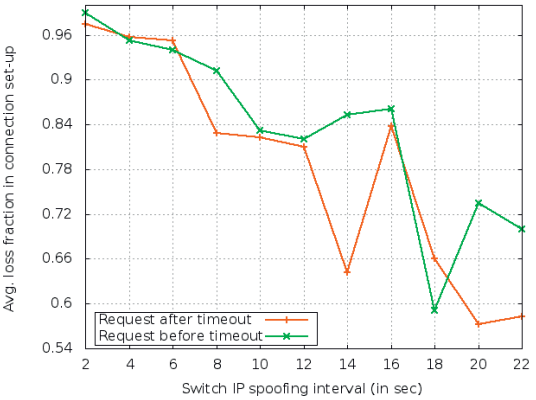


Fig. 8: Average loss with varying spoofing interval of the malicious switch

and loss fraction for all the clients. We did not observe any pattern for the connection set-up latency; however, the latency is high as compared to the case when there is no switch spoofing. The effect on the loss fraction is significant in this attack. We observed that the loss rate for the client connection is high when the interval for malicious switch connection is small. The loss rate went upto 98%. It is due to the fact that connection-disconnection of a malicious switch with small interval restricts the genuine switch to connect with the controller most of the time. With the increase of spoofing interval, the loss fraction decreased.

With a comparison to the data plane attack, launching this attack would be easier. Nonetheless, the assumption is the control channel is insecure. In the data plane attack, the adversaries have to find out the flows for which the controller does not set-up flow-rules in the flow tables of switches. On the other hand, in this attack adversaries need only information about the IP addresses of switches and the controller. With the spoofing of few switches, the performance of network services would be highly degraded.

## 5. Related Work

Various challenges were discussed for large-scale deployment of SDN including security, performance, scalability, and inter-operability of SDN with traditional networks in [15]. Threats in SDN have been categorized into seven vectors in [8]. Out of these, three vectors are specific to SDN including controller vulnerabilities, lack of trust between the controller and applications, and attack on the administrator host. Security analysis and modelling methodologies were presented for SDN, using the STRIDE and attack tree approach, uncover vulnerabilities in SDN in [7]. A survey on security using SDN and security of SDN, itself, were presented in [14].

A few work have been started to address the security challenges of SDN. A framework for developing various security applications over SDN is proposed in [16]. In addition, it has a provision for resolving conflicts among flow rules installation between various network applications. This restricts bypassing of firewall rules of a security application by a non-security application. A robust firewall application has been proposed through checking flow space and firewall authorization space in [5,21]. It accounts the intra-table flow rules dependencies in switches for rule-conflict detection. An extension to OpenFlow data plane has been proposed in order to reduce the control-data plane interaction during adversarial attacks in [18]. This reduces the load on the controller for handling large flow requests from switches. However, the extension works around only for TCP-based flow requests. A secure control channel architecture has been proposed for SDMN using Host Identity Protocol (HIP) in [10]. A controller architecture to defend the SDN from malicious administrator problems has been proposed in [11]. To restrict the access of network applications to the controller system and the managed network, a fine grained permission system has been proposed for applications in [22].

## 6. Conclusion

In this work, we evaluated the impact on the performance of network services running over SDN under adversarial attacks. The impact evaluation of various attacks gives an insight for the risk analysis of them and enables us to calculate the Common Vulnerability Scoring System (**CVSS**) corresponding to these attacks. We found that web service performances, in terms of response time and availability, get highly degraded in the presence of attacks. The adverse effects on the response time and availability pose great challenges for the operators to meet SLA for their customers. For the future work, more attacks and network services including VoIP, video streaming deployed in a real SDN testbed can be considered for attack analysis. An overall risk evaluation for various threats can also be presented by evaluating the impacts of them on SDN managed networks. Although our attack implementation was based on certain assumptions, many organizations lack in imposing security policies for their networks and make these assumptions valid[6]. SDN has great future and capabilities to bring a revolution in network industries as well as in research. However, for a wide acceptance, it has to devise solutions to overcome security threats.

## References


1. Kevin Benton, L. Jean Camp, and Chris Small. OpenFlow Vulnerability Assessment. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 151–152, New York, NY, USA, 2013. ACM.
2. POX Controller. http://www.noxrepo.org/pox/about-pox/.
3. Curl. http://curl.haxx.se/.
4. Jeremy M. Dover. A Denial of Service Attack against the Open Floodlight SDN Controller. Research Report, DoverNetworks, Dec 2013.
5. Hongxin Hu, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. FLOWGUARD: Building Robust Firewalls for Software-defined Networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 97–102, New York, NY, USA, 2014. ACM.
6. DDoS Attacks in the United Kingdom. http://goo.gl/qrw0hr, 2012.
7. Diego Kreutz, Fernando M. V. Ramos, Paulo Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined Networking: A Comprehensive Survey. *CoRR*, abs/1406.0440, 2014.
8. Diego Kreutz, Fernando M.V. Ramos, and Paulo Verissimo. Towards Secure and Dependable Software-defined Networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 55–60, New York, NY, USA, 2013. ACM.
9. Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.
10. Madhusanka Liyanage, Mika Ylianttila, and Andrei Gurtov. Securing the Control Channel of Software-Defined Mobile Networks.
11. Stephanos Matsumoto, Samuel Hitz, and Adrian Perrig. Fleet: Defending SDNs from Malicious Administrators. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 103–108, New York, NY, USA, 2014. ACM.
12. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.
13. Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, and Nick McKeown. Implementing an OpenFlow Switch on the NetFPGA Platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 1–9, New York, NY, USA, 2008. ACM.
14. S. Scott-Hayward, G. O'Callaghan, and S. Sezer. SDN Security: A Survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7, Nov 2013.
15. S. Sezer, S. Scott-Hayward, P.K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are we ready for SDN? Implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7):36–43, July 2013.
16. Seugwon Shin, Phillip Porras, Vinod Yegneswaran, Martin Fong, Guofei Gu, and Mabry Tyson. FRESCO: Modular Composable Security Services for Software-Defined Networks.
17. Seungwon Shin and Guofei Gu. Attacking Software-defined Networks: A First Feasibility Study. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 165–166, New York, NY, USA, 2013. ACM.
18. Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 413–424, New York, NY, USA, 2013. ACM.
19. OpenFlow Switch Specification. https://www.opennetworking.org/, Oct 2013.
20. OpenFlow: Proactive vs. Reactive Flows. http://networkstatic.net/openflow-proactive-vs-reactive-flows/.
21. Juan Wang, Yong Wang, Hongxin Hu, Qingxin Sun, He Shi, and Longjie Zeng. Towards a Security-Enhanced Firewall Application for OpenFlow Networks. In *Cyberspace Safety and Security*, volume 8300 of *Lecture Notes in Computer Science*, pages 92–103. Springer International Publishing, 2013.
22. Xitao Wen, Yan Chen, Chengchen Hu, Chao Shi, and Yi Wang. Towards a Secure Controller Platform for Openflow Applications. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 171–172, New York, NY, USA, 2013. ACM.