An Experimental Software Defined Security Controller for Software Defined Network

Malek Al-Zewairi*, Dima Suleiman*†, and Sufyan Almajali*
* Computer Science Department,
King Hussein Faculty of Computing Sciences,
Princess Sumaya University for Technology,
Amman 11941, Jordan
† Teacher at the University of Jordan
*m.alzewairi@psut.edu.jo, *†dimah_1999@yahoo.com, *s.almajali@psut.edu.jo,

Abstract—**Software Defined Networking is an emerging technology that permits computer network infrastructure to be scaled dynamically as needed while enhancing the manageability of the various network devices in heterogeneous environment as opposed to classical networking. These capabilities emerge from the separation of the data plane from the control plane; thus, allowing the network devices to be programmatically managed and controlled. Similarly, the concept of Software Defined Security allows security solutions such as Firewalls and Intrusion Detection Systems to be dynamically implemented, controlled and managed using programmable interfaces. In this research, the authors propose an experimental software defined security controller based on the Open vSwitch Controller to detect and prevent IP and MAC spoofing attacks on the network. The proposed controller is simulated using Mininet. The simulation results confirm that the proposed controller is capable to detect and prevent the aforementioned attacks with high precision.**

Index Terms—**Software Defined Security; Software Defined Networking; SDSec; SDS; SDN;**

I. Introduction

On contrast to computing power and computing storage, in classical computer networking, the cost of network infrastructure does not scale down as the network scales up. Additionally, the manageability of network devices in large-scale networks is quite difficult and becomes more complicated in heterogeneous environments where network devices are adopted from different manufacturers [1]. Software Defined Networking (SDN) is an emerging technology that redefined the architecture of traditional computer networking. As the name suggests, in SDN the network is defined, managed and controlled via programmable interfaces much like a computer software; thus, allowing the network to dynamically change its topology, characteristics and scale up or down as needed [2]. Another distinctive feature of SDN is the segregation of the control plane from the data plane. In network, the Control Plane is where the network topology is defined and exchanged. Moreover, it is responsible for making routing decisions; in addition to, handling system configurations and network management functions. On the other hand, the Data Plane (i.e. Forwarding Plane) is responsible for forwarding packets to the next hop as instructed by the control plane. On contrary to traditional networking, the functionalities of the control plane are separated from the data plane and are programmatically managed by a centralized entity that is referred to as the Controller. Moreover, on top of the control plane resides the Application Plane (also referred to as SDN Plane), whereas several applications and services such as security services, traffic monitoring, access control and many other services can be implemented to facilitate network management operations and the mapping between the entire network and the applications is performed by the Network Operating System (NOS). In SDN, the communications between the controller and the forwarding elements (e.g. switches) are performed using the OpenFlow (OF) protocol. Evidently, these distinctive features have significantly helped in decreasing the network operating cost in addition to increasing the creativity of network applications [1].

However, the introduction of the application plane coupled with the separation between the control plane and the data plane have resulted in spawning new challenges for network security. Moreover, previously known attacks such as Denial of Service (DoS), Distributed Denial of Service (DDoS), Man-in-the-Middle (MitM), and IP Spoofing to name a few have to be treated differently in SDN since they have the ability to propagate alongside the data flow path through the controller and take down the entire network. Additionally, attacks against the application plane constitutes a new attack vector [3].

Nonetheless, the centralized approach for managing the network provides the controller with global visibility of the forwarded packets; which permits it to monitor, reroute, block or restrict the traffic using custom security applications. This unique ability to provide multiple security services using programmable interfaces with centralized overview of the network has led to the emergence of a new security model often referred to as Software Defined Security (SDS or SDSec). SDSec helps in providing the necessary security services for virtualized environments by employing the same segregation technique between the control plane and data plane used in SDN. It also consists of three main plane (i.e. data, control and application planes) as in SDN. The main features that make SDSec different and more powerful than the traditional security approaches can be summarized as abstraction, automation, elasticity, concurrency control, visibility and portability

[4]. Recently, several research works have contributed to the SDSec field [1], [3–12].

IP and MAC address spoofing attacks are used to mask the attacker identity by impersonating another machine on the network and are often employed at the preliminary stages of several security attacks such as DoS/DDoS, MitM and by-passing network-based filtering controls. In order to recognize the active links in the network, the controller in SDN utilizes the OpenFlow Discovery Protocol (OFDP), which in its turn leverages a modified version of the Link Layer Discovery Protocol (LLDP) to perform the topology discovery. However, the OFDP has been proven vulnerable to spoofing attacks [9]. In this paper, the authors propose an experimental SDSec controller based on the Open vSwitch (OVS) controller to detect and prevent IP and MAC spoofing attacks in SDN. The proposed SDSec controller is implemented using python on the Mininet platform.

The rest of this paper is organized as follows: in Section II, a collection of the recent related works on SDN and SDS are reviewed. Then, the security issues facing SDN is discussed in Section III. Followed by introducing the proposed SDSec controller, its design, implementation and evaluation in Section IV. Finally, Section V concludes this paper and presents the future work.

II. Literature Review

In this section, the related works on SDN and SDSec are presented and reviewed.

Several survey studies that discuss the security in SDN have been published recently [1], [3], [10], [13]. The centralized architecture of SDN has introduced many advantages to help in improving the network security as the controller can monitor and analyze the network traffic generated from all the network nodes. On the other hand, this centralized architecture has brought various security challenges to new level such as DoS attacks against the controller that could bring the entire network down [1].

The numerous security issues of SDN such as DoS/DDoS, elevation of privileges, repudiation, IP/MAC address spoofing, tempering, and information disclosure have been discussed in [10]. Furthermore, the survey has also covered several SDSec solutions proposed in the literature such as SDSec based Intrusion Detection System (IDS), Intrusion Prevention System (IPS), access control, firewall, policy management and auditing. While in [14], the authors proposed IDS for SDN and fuzzy logic rules were utilized to improve the proposed IDS by reducing the number of code lines by 20-30

The implications of different security attacks on SDN have been investigated in [5]. The authors identified the source of attacks in SDN to be originated from malicious hosts, switches or even controllers. They also discussed the ability of malicious switches to affect the network topological causing the controller to change the traffic direction and create black holes in the network.

DDoS attacks are prominent attacks whether in traditional networking or in SDN that has been proven difficult to prevent. In [6], the authors proposed a Software Defined Security Networking Mechanism (SDSNM) to detect and mitigate DDoS attacks in SDN. The proposed solution suggests implementing strict access control rules at the network edges in order to prevent hosts from causing the attack.

In [7], the authors proposed a Software Design Security Architecture (SDSA) that facilitates multisystem security integration, which decouples the security operations from the network control in order to reduce the size of the software in SDN and to improve the security. The authors argued that the integration between the security functionality and the network control in SDN has significantly increased the security challenges due to the fact that in SDN the security between systems is unconnected and within the system is closely coupled. Additionally, because of the flexibility of network expanding and the diversity of services, which can result in more security issues.

A new approach (i.e. OpenSec) for policy-driven network management in SDN was proposed in [8]. OpenSec is based on OF protocol and it can be used to configure multiple network devices automatically by converting human-readable security policies into OF rules to be used by the controller. While in [15], the authors proposed a novel framework (i.e. MigApp) to migrate the security policies of virtual machines into SDN environment. The proposed framework was implemented at the application plane and tested on the Floodlight controller.

The lack of packets authentication control in LLDP has left the OFDP prone to spoofing attacks; thus, allowing attackers with access to compromised hosts to inject bogus links into the network using specially crafting packets. A Keyed-Hash Message Authentication Code (HMAC) technique was proposed in [9] to address the aforementioned issue by calculating the HMAC value to each LLDP packet.

In [11], the authors proposed a Deep Packet Inspection (DPI) system for Software-Defined Infrastructure (SDI) with an IDS/IPS solution. Moreover, the proposed system was implemented on the Smart Applications on Virtual Infrastructure (SAVI) testbed. A centralized firewall and anti- DDoS solution was proposed in [12].

One of the security challenges of SDN is to deploy the security devices logically through which the network traffic will pass because unnecessary placement of security devices will result in higher resource consumption. In [16], a path decision algorithm was proposed in order to reduce the resources consumption by implementing rules translation algorithm to determine the network traffic in which to pass through specific security devices.

III. Security Issues in SDN

This section provides a brief discussion about the main security issues and the common challenges in SDN and tries to suggest possible solutions.

The security challenges in SDN can be classified into three main categories based on the SDN plane (i.e. application, control, and data), in which they affect [17], [18]. In SDN, most of the network functionality can be implemented as an application. This allows users to tailor the network application to fit their needs. Nonetheless, SDN does not define a standard or specification to authenticate the application; thus, the application layer constitutes a rich area for security enhancements. In this sense, a malicious application can take down the entire network since applications run on the controller and therefore inherits its access privileges to all network resources.

Clearly, this issue is unique to SDN in comparison to traditional networking and it could severely affect all the security pillars; for example, a malicious application may steal users credentials (confidentiality), modify sent/received data (integrity) or bring down the network (availability). A possible solution is to enforce a strict centralized model, in which both the application and its developer must be authorized, authenticated and verified prior allowing the application to run on the controller. This centralized tight seal model decreases the chances of publishing malicious applications but in the same time creates many frustrations for developers and end-users alike.

On the other hand, following a more relaxed model in which anyone can freely develop and publish their own applications that can run on SDN controllers might induce greater creativity than in a more restricted environment; however, the probability of getting fake or malicious applications is far higher. SDN manufactures can follow either of the two approaches (i.e. relaxed model vs. strict model) and the end-user can choose between a strict model with less flexibility but that provides further more security assurance or a more relaxed model with less security but much more flexibility.

The second security issue that SDN faces resides on the control plane. Due to the centralized decision-making design of SDN, it leaves it highly vulnerable for DoS/DDoS attacks. Although, flooding attacks are common across all networks, the design of SDN makes it an easy target for DoS/DDoS attacks because the controller assumes a master role in SDN; thus, being a single point of failure. In essence, this attack only affects the availability of the network and has no effect on confidentiality or integrity. Distributed Anti-DoS/DDoS solutions at tier-1, tier-2 and tier-3 Internet Service Providers (ISPs) have been proven very effective in mitigating flooding attacks in traditional networking; however, it cannot protect from local DoS attacks (originated from within the organization network itself), which can be handled by an anti-DoS software solution at the SDN application plane.

The third security challenge affects the data plane. In SDN, the rules for routing traffic (i.e. flow control) are defined at the application plane, distributed by the controller and enforced at the data plane in a physically segregated architecture. Clearly, this segregation at the physical layer might introduce a delay in setting up and updating the flow rules at the data plane, in which it will create challenges for the security applications. This is specifically true for authorization and authentication, because the traffic might be delayed while waiting a flow rule to be updated by the controller in large congested networks making SDN prone to saturation attacks. Machine learning techniques can be applied to produce smarter controllers that can make informative decisions based on previous traffic, thus, minimizing this delay.

IV. Proposed Software Defined Security Controller

In SDN, the controller plays a crucial role by providing a centralized decision-making center; in addition to performing network manageability and functions abstraction. SDSec has been proposed as an addendum to SDN. In SDSec, the security functionality has also been abstracted, decoupled and centralized as with SDN. Therefore, the controller is responsible for the security functionality in addition to the networking functionality. This extension bridges the same positive and negative aspects from SDN to SDSec. In this section, a new experimental SDSec controller based on OVS is proposed, discussed and evaluated against IP and MAC spoofing attacks in SDN.

A. Design

In essence, the proposed SDSec controller is a SDN controller that its functionality has been extended in order to accommodate the proposed security functionality as well as maintaining its native networking capabilities. Mininet network emulator was chosen to design and implement the proposed SDSec controller for its versatility, wide use and ability to rapidly prototype and test different scenarios. The proposed SDSec controller extends the OVS Controller from Mininet, which is a test controller provided by the Open vSwitch project. Nonetheless, the proposed SDSec controller can extend any other SDN controller supported by Mininet including the basic Mininet controller and NOX controller.

The proposed SDSec controller relies on two in-memory tables that were added to the controller, which are the Switches Table and the Hosts Table. The former contains information about trusted network switches including the switch name, IP address, MAC address and available interfaces. The hosts table holds information about network hosts including the host name, IP address, MAC address, to which switch and on what interface it is connected in addition to its authentication status and the action to be taken to its traffic (i.e. forward or drop). Although, only two tables were introduced, the design supports adding multiple tables natively with minimum changes to the code. SQLite in-memory database is used to store the two tables and their data.

Table I and II show the different fields and their definition for both the switches table and the hosts table respectively.

Before a host or a switch can communicate on the network, it has to perform an authentication process with the SDSec controller. This process commences when the new device is discovered on the network, the controller checks the device IP address and MAC address against both the switches and hosts tables prior allowing it to communicate on the network. In case a match is found in either tables, then a false value is assigned to the AUTH field; this might indicate IP/MAC spoofing attack. Otherwise, a true value is assigned to the AUTH field. For example, if a new host is added to the

TABLE I
SDSec Controller Switches Table

Field	Datatype	Constraints	Data
ID	Integer	Auto Increment Primary Key	Sequence number
SWITCH	Text	-	Switch name
INTERFACE	Text	-	Switch interface (NIC)
IP	Long	-	Switch IP address
MAC	Long	-	Switch NIC MAC address

TABLE II
SDSec Controller Hosts Table

Field	Datatype	Constraints	Data
ID	Integer	Auto Increment Primary Key	Sequence number
HOST	Text	-	Host name
IP	Long	-	Host IP address
MAC	Long	-	Host MAC address
SWITCH	Text	-	Switch name
INTERFACE	Text	-	Switch interface (NIC)
AUTH	Boolean	-	True/False
ACTION	Boolean	-	Forward(True)/Drop(False)

network, the controller will search for its IP/MAC address in the switches and hosts tables. If the controller finds an entry in either tables where the IP and/or MAC address of the host, then, this means that the new host is impersonating an existing host IP/MAC address. The device information is removed from both tables when it leaves the network in order to allow it to join the network again in the future.

B. Implementation

The proposed SDSec controller is implemented and evaluated on the following virtual environment: Ubuntu 14.04.4 LTS OS with Python 2.7.6, Mininet 2.3.0d1, Open vSwitch 2.3.1 and OpenFlow version 1.3. Both OVS and UserSwitch were used in the testing of the proposed SDSec controller, as well as, the basic Mininet Controller, OVSController and NOX controller. The source-code of the SDSec controller is available for researchers at GitHub[1]. The steps to setup the implementation environment is provided as well.

C. Evaluation

In order to evaluate the proposed experimental SDSec controller, a scenario of a malicious host trying to perform IP spoofing attack is developed. Figure 1 shows the custom evaluation network topology, which consists of 8 hosts (i.e. h1-h8), 3 switches (i.e. s1-s3), and a single controller (i.e. c0). The third host (i.e. h3, MAC address 00:00:00:00:00:03) is assumed to be the malicious host. It tries to spoof the fifth host (i.e. h5, MAC address 00:00:00:00:00:05) IP address (i.e. 10.10.2.2).

[1]https://github.com/alzewairi

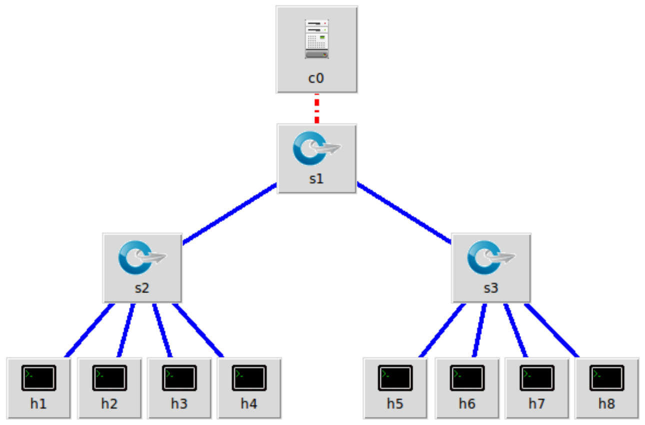


Fig. 1. Custom network topology used to evaluate the proposed SDSec controller.

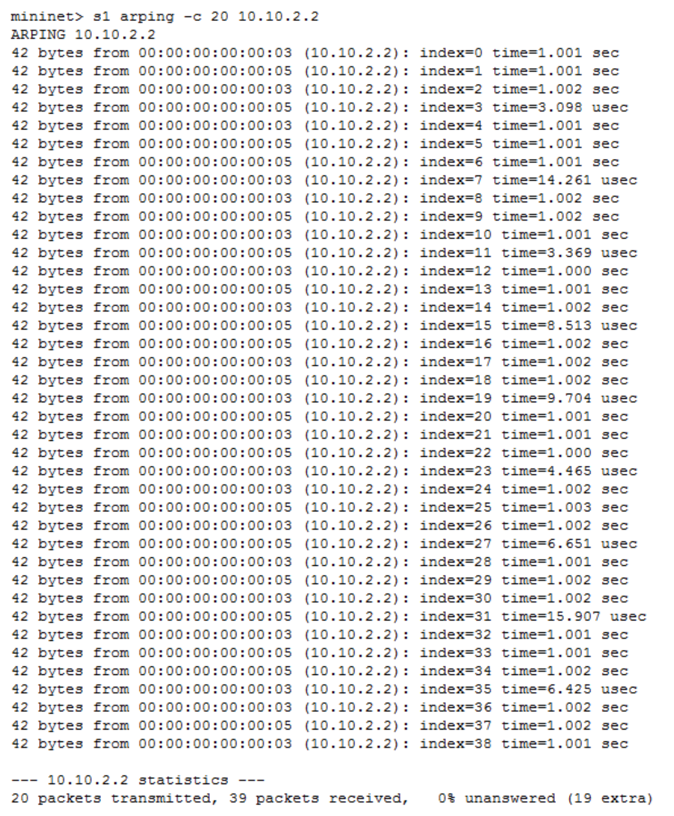
The ARPING Linux tool was used to generate traffic on the network by sending continues ARP echo requests from s1 to h5. Figure 2 shows the debugging output of the ARP ping command without using the proposed SDSec controller. Notwithstanding an IP duplication on the network, both h3 and h5 have been able to receive and successfully respond to the ARP echo requests from s1. Interestingly, in some cases the malicious host was able to answer the ARP echo requests significantly faster than the genuine host, which proofs that the IP spoofing attack was successful.

Accordingly, Figure 3 shows the time both hosts (i.e. h3 and h5) have taken to respond to each ARP echo request. The red line represents the responses of the malicious host (i.e. h3) to the ARP echo requests from switch s1, while the blue line represents the answers from the genuine host (i.e. h5).

On the contrary, Figure 4 shows the results of the same command but with the SDSec Controller enabled showing that the malicious host was unable to communicate on the network at all represented by the red line and that the genuine host was able to communicate with constant response time represented by the blue line. It is worth mentioning that the response time was normalized to microseconds (μs).

V. Conclusion

In this paper, an experimental controller for Software Defined Security based on the Open vSwitch Controller aiming to detect and prevent IP and MAC spoofing attacks on Software Defined Networking was proposed. The controller was implemented using Python by extending the Open vSwitch controller in Mininet. Moreover, the proposed controller can extend any SDN controller with minimal changes to the code. The proposed controller was evaluated using a custom network topology in Mininet. The results showed that the proposed controller was capable of detecting and preventing IP spoofing attacks. As future work, the authors recognize several areas of improvement including the separation of SDSec controller from the SDN controller and implementing defense policies against other types of attacks.



```
mininet> s1 arping -c 20 10.10.2.2
ARPING 10.10.2.2
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=0 time=1.001 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=1 time=1.001 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=2 time=1.002 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=3 time=3.098 usec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=4 time=1.001 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=5 time=1.001 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=6 time=1.001 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=7 time=14.261 usec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=8 time=1.002 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=9 time=1.002 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=10 time=1.001 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=11 time=3.369 usec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=12 time=1.000 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=13 time=1.001 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=14 time=1.002 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=15 time=8.513 usec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=16 time=1.002 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=17 time=1.002 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=18 time=1.002 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=19 time=9.704 usec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=20 time=1.001 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=21 time=1.001 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=22 time=1.000 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=23 time=4.465 usec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=24 time=1.002 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=25 time=1.003 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=26 time=1.002 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=27 time=6.651 usec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=28 time=1.001 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=29 time=1.002 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=30 time=1.002 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=31 time=15.907 usec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=32 time=1.001 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=33 time=1.001 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=34 time=1.002 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=35 time=6.425 usec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=36 time=1.002 sec
42 bytes from 00:00:00:00:00:05 (10.10.2.2): index=37 time=1.002 sec
42 bytes from 00:00:00:00:00:03 (10.10.2.2): index=38 time=1.001 sec

--- 10.10.2.2 statistics ---
20 packets transmitted, 39 packets received,   0% unanswered (19 extra)
```

Fig. 2. Debugging results of ARPING tool without using the proposed SDSec controller.

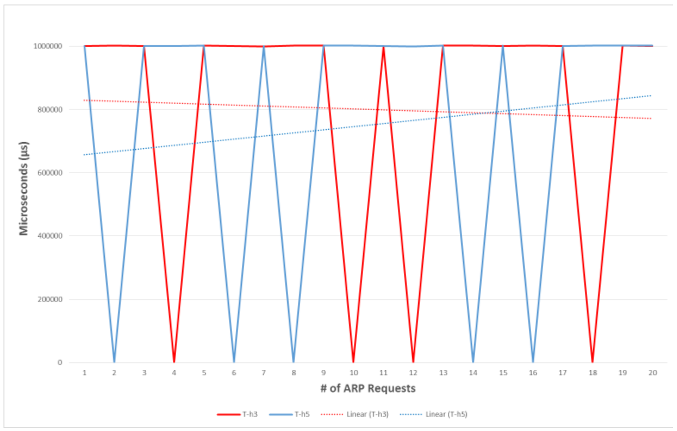


Fig. 3. Time-to-Respond with IP spoofing attack without using the proposed SDSec controller.

References

[1] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A Survey of Security in Software Defined Networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[2] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[3] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in Software Defined Networks: A Survey," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[4] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, "SDSecurity: A Software Defined Security experimental framework," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, Jun. 2015, pp. 1871–1876.

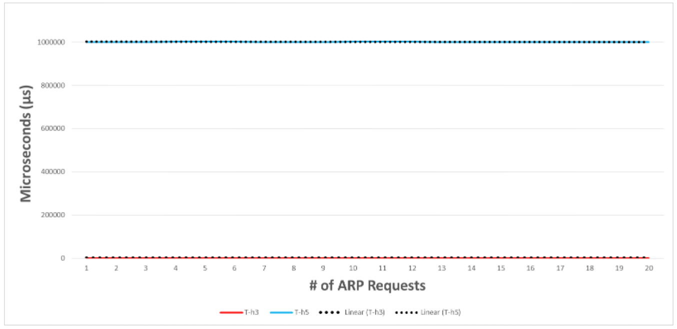


Fig. 4. Time-to-Respond with IP spoofing attack using the proposed SDSec controller.

[5] A. S. Prasad, D. Koll, and X. Fu, "On the Security of Software-Defined Networks," in *2015 Fourth European Workshop on Software Defined Networks (EWSDN)*, Sep. 2015, pp. 105–106.

[6] X. Wang, M. Chen, and C. Xing, "SDSNM: A Software-Defined Security Networking Mechanism to Defend against DDoS Attacks," in *2015 Ninth International Conference on Frontier of Computer Science and Technology (FCST)*, Aug. 2015, pp. 115–121.

[7] L. Yanbing, L. Xingyu, J. Yi, and X. Yunpeng, "SDSA: A framework of a software-defined security architecture," *China Communications*, vol. 13, no. 2, pp. 178–188, Feb. 2016.

[8] A. Lara and B. Ramamurthy, "OpenSec: Policy-Based Security Using Software-Defined Networking," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 30–42, Mar. 2016.

[9] T. Alharbi, M. Portmann, and F. Pakzad, "The (in)security of Topology Discovery in Software Defined Networks," in *2015 IEEE 40th Conference on Local Computer Networks (LCN)*, Oct. 2015, pp. 502–505.

[10] I. Alsmadi and D. Xu, "Security of Software Defined Networks: Asurvey," *Computers & Security*, vol. 53, pp. 79–108, Sep. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016740481500070X

[11] P. Yasrebi, S. Monfared, H. Bannazadeh, and A. Leon-Garcia, "Security function virtualization in software defined infrastructure," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 778–781.

[12] J. Jeong, J. Seo, G. Cho, H. Kim, and J. S. Park, "A Framework for Security Services Based on Software-Defined Networking," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Mar. 2015, pp. 150–153.

[13] L. Schehlmann, S. Abt, and H. Baier, "Blessing or curse? Revisiting security aspects of Software-Defined Networking," in *2014 10th International Conference on Network and Service Management (CNSM)*, Nov. 2014, pp. 382–387.

[14] S. Dotcenko, A. Vladyko, and I. Letenko, "A fuzzy logic-based information security management for software-defined networks," in *2014 16th International Conference on Advanced Communication Technology (ICACT)*, Feb. 2014, pp. 167–171.

[15] S. Sadri, Y. Jarraya, A. Eghtesadi, and M. Debbabi, "Towards migrating security policies of virtual machines in Software Defined Networks," in *2015 1st IEEE Conference on Network Softwarization (NetSoft)*, Apr. 2015, pp. 1–9.

[16] H. Tu, W. Li, D. Li, and J. Yu, "A scalable flow rule translation implementation for software defined security," in *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*, Sep. 2014, pp. 1–5.

[17] M. Liyanage, I. Ahmed, M. Ylianttila, J. L. Santos, R. Kantola, O. L. Perez, M. U. Itzazelaia, E. M. d. Oca, A. Valtierra, and C. Jimenez, "Security for Future Software Defined Mobile Networks," in *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, Sep. 2015, pp. 256–264.

[18] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Software-defined networking security: pros and cons," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 73–79, Jun. 2015.