Impact of Network Load for Anomaly Detection in Software-Defined Networking

Ashish Gupta, Bharat Didwania, Gaurav Singh, Hari Prabhat Gupta, Rahul Mishra and Tanima Dutta

Abstract Software-Defined Networking (SDN) introduces a new network paradigm for separating the control plane and data plane. The control plane manages the packet flow in the data plane of the network. The anomaly detection in the context of SDN is to identify potentially harmful traffic. If an anomaly occurs because of malicious packets in SDN, inspecting the payload of packets is an effective way to recognize abnormal traffic. In this paper, we consider different bandwidths and topologies of the network for the detection of an anomaly in SDN. We also evaluate the performance of the SDN on the same network. We have implemented different tree topologies on OpenFlow controller using Mininet network emulator. We considered OpenFlow messages as a performance metric for evaluating the performance of the network with different tree topologies.

Keywords Anomaly detection · Mininet network emulator · Software-defined networking

A. Gupta · H. P. Gupta (✉) · R. Mishra · T. Dutta
Department of Computer Science and Engineering, IIT (BHU), Varanasi, India
e-mail: hariprabhat.cse@iitbhu.ac.in

A. Gupta
e-mail: ashishg.rs.cse16@iitbhu.ac.in

R. Mishra
e-mail: rahulmishra.rs.cse17@iitbhu.ac.in

T. Dutta
e-mail: tanima.cse@iitbhu.ac.in

B. Didwania · G. Singh
Department of Electrical Engineering, IIT (BHU), Varanasi, India
e-mail: bharat.didwania.eee14@iitbhu.ac.in

G. Singh
e-mail: gaurav.singh.eee14@iitbhu.ac.in

© Springer Nature Singapore Pte Ltd. 2020

M. L. Kolhe et al. (eds.), *Advances in Data and Information Sciences*,
Lecture Notes in Networks and Systems 94,
https://doi.org/10.1007/978-981-15-0694-9_13

1 Introduction

Software-Defined Networking (SDN) is a mechanism of improving network efficiency, management, and security by separating the control and data plane of the network [10, 11]. In the traditional networking concept, two important functions are as follows: first decide how to process incoming packets, e.g., to which physical port the packets will forward, and then output the data to the predecided port. In SDN, these two mentioned tasks are decoupled into data plane functions and control plane functions. The separation of control plane and data plane makes a vertical decentralization of the traditional network. The forwarding decision (control plane) is made by a controller, which manages and operates a network through open interfaces. The controller located above the data plane maintains a centralized view of the entire network, which provides a decisive advantage over the current network architectures. The logically centralized architecture supports programmability of the control plane that allows bifurcation of control plane functionality from network devices like routers, switches, etc., to specified controller instances running in a software as shown in Fig. 1.

The controllers in SDN have granular control over the switches to handle data and the ability to automatically prioritize or block certain types of packets. This increases network efficiency without investing in expensive and application-specific network switches. Multiple types of network technologies are designed for SDN that can make the network more agile and flexible to support the visualization of the server. It also supports storage infrastructure of the data center. In short, SDN can be defined as an approach for designing, building, and managing networks that creates a decentralization by separating the control plane from the data plane of the network. In other words, it is the separation of the network infrastructure and the control functions of the network.

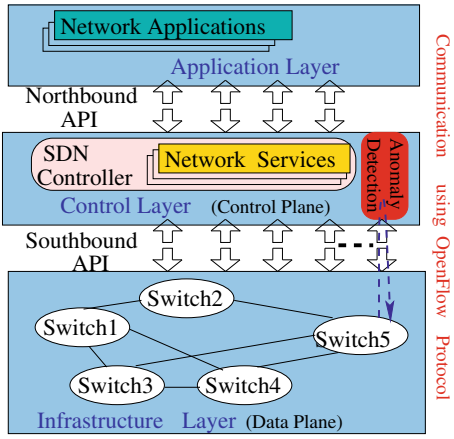


Fig. 1 Illustration of the SDN Architecture with anomaly detection

Figure 1 shows that the control plane act as a middleware between the network devices at the bottom and the network applications at the top. In SDN, flow management is a crucial task for making the network intelligent. The network intelligence by controlling the flows in the network is provided by the brain of the network, i.e., controllers. The switches and routers in SDN are not provided with any intelligence and they therefore simply accept rules from the controller for forwarding the packets. SDN mainly uses two interfaces, i.e., southbound and northbound interfaces. The southbound interface is used to pass information from the SDN controller to routers and switches. Similarly, the northbound interface is used to relay information from the SDN controller to applications and services that are running over the network. OpenFlow is a protocol used in SDN for providing centralized control functionality to the switches and traffic flows in a given network.

The growing popularity and recent advancements in SDN enable the inclusion of management plane along with control and data planes. The software services that can help in remotely monitoring and configuring the controller function are part of the management plane. It includes the definition of the policies of the network and its execution on the control plane, where data is forwarded by the data plane.

With the invention of the SDN, there is an architectural and structural modification in the traditional network. The logical centralization of SDN helps in dynamically adjusting the data traffic from a single point of control, i.e., controller. In SDN, the single point control is beneficial in providing the vertical decentralization of network and increasing the network efficiency but it is a vulnerable point for attack. The person who can get the access of the controller can demolish the entire network performance.

The intrusion detection in SDN is one of the critical attacks where less attention from the researcher is paid despite a huge amount of work done in the area of Openflow. The work emphasized on anomaly detection in SDN is covered in [4, 6, 7, 12]. In [4], the authors have used the Openflow architecture for detecting DDoS attack. In [12], the authors used various anomaly detection algorithms in experiments, where the algorithm was validated in both Small Office/Home Office (SOHO) and purely home networks. In [5, 6], the authors have used the OpenFlow protocol for enhancing the Remote Triggered Black Hole (RTBH) routing approach such that it can help to overcome the DDoS attack. The authors in [16] proposed an algorithm that is capable of dynamically changing the measurement granularity in both spatial and temporal dimensions for balancing the trade-off between error monitoring overhead and accuracy of anomaly detection. The author in [15] has elaborated attack scenarios and implementing them as SDN applications. They have used machine learning algorithms that are evaluated for their aptitude to detect anomalies in the SDN control plane.

• Our Contributions: In this paper, we consider different bandwidth of the network for detection of an anomaly in SDN. We also evaluate the performance of the SDN on the same network with different tree topologies. We have implemented tree topology on the OpenFlow controller using Mininet network emulator. We

considered OpenFlow messages as performance metrics for evaluating the performance of the network with different tree topologies.

The rest of the paper is organized as follows: In the next section, we discuss anomaly detection approaches in SDN. The experiment parameters, performance metrics, and results are given in Sects. 3 and 4 conclude this paper.

2 Anomaly Detection

The aim of the anomaly detection technique is to identify potentially harmful traffic in the computer network. Anomalies in the computer network are defined as *the patterns in data that do not resemble a well-defined notion of normal traffic flow in the network* [8, 9, 13, 15]. If an anomaly occurs because of malicious packets (e.g., originating from malware), inspecting the packet's payload is an effective way to recognize abnormal traffic. Two types of payload-based classifiers exist—Deep Packet Inspection (DPI) and Stochastic Packet Inspection (SPI). These methods provide very accurate results; however, the computational costs are high. Thus, approaches that merely need header fields instead of packet payload are required. However, building a strict model which is able to isolate the *normal* network traffic is very difficult. Hence, detecting anomalies in network traffic is a complex task. Traffic classification can also be employed to detect anomalies in a network.

In this paper, we are using rule-based traffic filtering as a subset of common protocols. We consider that the packet which has to pass through the centralized SDN controller must satisfy at least one filtering rule. In the case of filtering rule satisfaction, the flows are installed on the data plane. These installed rules help in packet forwarding inside the network in the future course of action without any involvement of the controller. In our experiment, we include non-IP packets and TCP packets. The controller computes an anomaly count based upon the arrival pattern of each byte of the data packet in the network. A predefined threshold is calculated for matching each packet against it. If the frequency of arrival of the packet is more than the threshold, then it is declared as an anomaly. Figure 1 illustrates a packet passing through the SDN controller. The anomaly detection process works with the SDN controller. The packet flows in the switch if the anomaly detection conditions are satisfied.

3 Experimental Results

In SDN, various types of OpenFlow messages have different impacts on user traffic. The accuracy and granularity of network flow measurement play an important role in anomaly detection in a network. In this paper, we investigate an easier and effective mechanism for implementing SDN in Mininet.

3.1 Experimental Setup on Mininet

In the proposed anomaly detection method, we used Mininet network emulator version 2.2.1 for creating network topologies in SDN [1]. Mininet is written in Python programming language and creates Open vSwitch version 2.0.2. We used POX controller that is a Python programming language based open source SDN controller. We created three tree topologies on Mininet. The first topology (Topology 1) consists of two OpenFlow virtual switches, two hosts, and a controller. The second topology (Topology 2) consists of three OpenFlow virtual switches, six hosts, and a controller. The third topology (Topology 3) consists of three OpenFlow virtual switches, four hosts, and a controller. Figure 2 illustrates all three topologies. The controller was made to run on a separate Internet Protocol (IP) address to avoid unnecessary traffic and for the proper capture of OpenFlow messages. We used *Wireshark* for capturing the traffic.

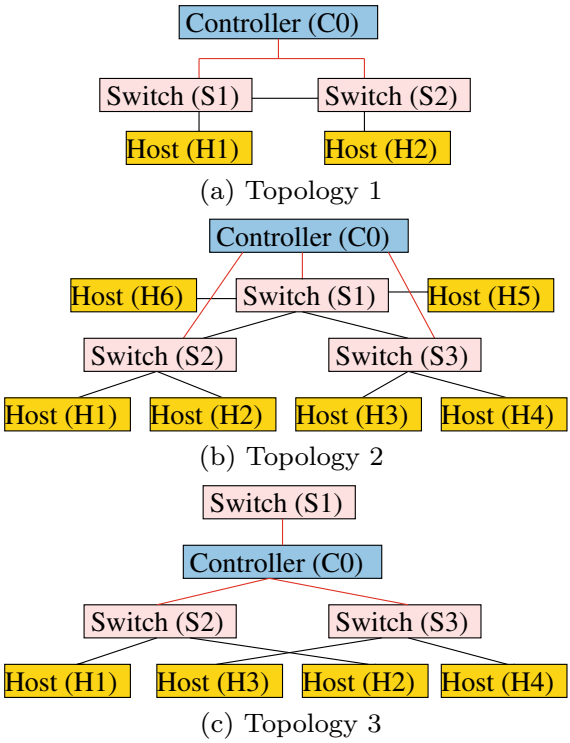


Fig. 2 Illustration of different tree topologies that are considered for experimentation

3.2 Dataset and Traffic Generation

We are using CAIDA dataset [14] in order to generate network traffic that aims to evaluate our anomaly detection method. Since the dataset size was huge, we split the dataset into smaller files using *Wireshark*. We replaced the IP addresses in the dataset with the IP addresses of the hosts in Mininet topology using bittwiste. The dataset was replayed using *tcpreplay* [3] at different speeds in mbps. In order to delete the flow tables of the switches after each and every packet, we modified the POX controller [2] code so that the controller may receive all the packets in *l2_learning.py*.

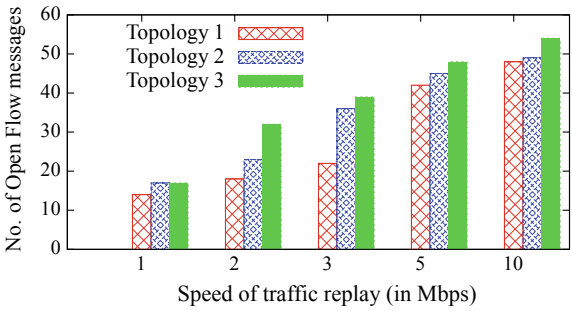
3.3 Impact on Packet Transmission

We run *tcpreplay* for different speeds in packet transmission and count the number of packets received by the controller. For anomaly detection, we have to use the minimum possible speed of traffic in order to check all the packets. We can get control messages for all the packets as the controller is deleting the flows after each packet. We set the bandwidth of topology links in *miniedit* at different bandwidths and calculated the number of packets received by the controller. We have considered three topologies as shown in Fig. 2. The packet dataset was replayed at different speeds using *tcpreplay*. The number of packets received was determined by using *Wireshark* and we observed that:

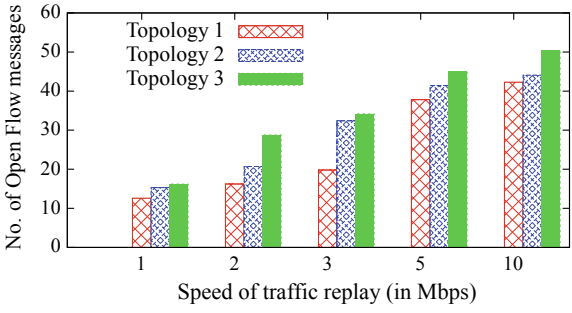
1. Below the set threshold bandwidth, the number of packets received increased by increasing the speed of replay.
2. Above the set bandwidths, the number of received packets remained almost the same.

We used three different topologies for analysis which were with varying depths and complexities. The results illustrate that with the increase in the complexity of the topology, the number of received packets also increases for the same speed of replay. We used *iperf* command to cross-check whether the bandwidths were set with accuracy.

Figure 3 illustrates the impact of the network topologies (Topology 1, Topology 2, and Topology 3) and network speed on the OpenFlow messages. Part (a) and Part (b) of Fig. 3 illustrate that the OpenFlow messages count increases with an increase in the network data flow speed. The figure shows that Topology 1 requires less number of messages than Topology 3. This is because Topology 1 consists of less number of switches. The figure shows that the low network speed requires less number of OpenFlow messages. An interesting observation from the results is that less number of switches (i.e., Topology 1) require less OpenFlow messages and therefore the time complexity of anomaly detection will decrease.



(a) Network Speed 6Mbps.



(b) Network Speed 3Mbps.

Fig. 3 Impact of network topologies and speed on the OpenFlow messages

4 Conclusion and Future Work

In this paper, we considered different network topologies to evaluate the performance of SDN. We have implemented the topologies on OpenFlow controller using the Mininet network emulator. We implemented the anomaly detection process on OpenFlow SDN controller. The process is written in Python programming language. We have used a standard database to evaluate the performance of SDN. We considered OpenFlow messages as performance metrics for evaluating the performance of the network. The results show that the number of OpenFlow messages increases with the speed of the network. We also observed that the network topology plays an important role in improving the performance of SDN. As a future work, we plan to detect anomalous packets by implementing machine learning algorithms in the SDN controllers. We will train our model on network dataset and merge this model with an SDN controller.

Acknowledgements This work is supported by the Science and Engineering Research Board (SERB) file number ECR/2016/000406/ES, project entitled as Development of an Energy-efficient Wireless Sensor Network for Precision Agriculture, and scheme Early Career Research Award.

References

1. (2016) Mininet: An Instant Virtual Network on yout Laptop. http://mininet.org.
2. (2016) POX: An Openflow controller. http://www.noxrepo.org/pox/about-po.
3. AppNeta. (2016). Tcpreplay. http://tcpreplay.synfin.net.
4. Braga, R., Mota, E., & Passito. A. (2010). Lightweight DDoS flooding attack detection using NOX/OpenFlow. In *Proceedings of IEEE Conference on Local Computer Networks (LCN)*, pp. 408–415.
5. Giotis, K., Androulidakis, G., & Maglaris, V. (2014). Leveraging SDN for efficient anomaly detection and mitigation on legacy networks. In *Proceedings of European Workshop on Software Defined Networks*, pp. 85–90.
6. Giotis, K., Argyropoulos, C., Androulidakis, G., Kalogeras, D., & Maglaris, V. (2014). Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Computer Networks*, *62*, 122–136.
7. Gupta, H. P., Rao, S. V., & Tamarapalli, V. (2015). Analysis of stochastic *k*-coverage and connectivity in sensor networks with boundary deployment. *IEEE Transactions on Intelligent Transportation Systems*, *16*(4), 1861–1871.
8. Gupta, H. P., Rao, S. V., & Venkatesh, T. (2016). Sleep scheduling protocol for k-coverage of three-dimensional heterogeneous wsns. *IEEE Transactions on Vehicular Technology*, *65*(10), 8423–8431.
9. Gupta, H. P., Venkatesh, T., Rao, S. V., Dutta, T., & Iyer, R. R. (2017). Analysis of coverage under border effects in three-dimensional mobile sensor networks. *IEEE Transactions on Mobile Computing*, *16*(9), 2436–2449.
10. Lange, S., et al. (2015). Heuristic approaches to the controller placement problem in large scale sdn networks. *IEEE Transactions on Network and Service Management*, *12*(1), 4–17.
11. Lopes, F. A., Santos, M., Fidalgo, R., & Fernandes, S. (2016). A software engineering perspective on sdn programmability. *IEEE Communications Surveys Tutorials*, *18*(2), 1255–1272.
12. Mehdi, S. A., Khalid, J., & Khayam, S. A. (2011). Revisiting traffic anomaly detection using software defined networking. I: *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID), Springer Berlin Heidelberg*, pp. 161–180.
13. Prabhu, G., & Jagatheesan, S. (2018) An efficient predictive network anamoly detection and visualization. *International Journal of Engineering Science*, 16651.
14. Singh, K. J., Thongam, K., & De, T. (2018). Detection and differentiation of application layer ddos attack from flash events using fuzzy-ga computation. *IET Information Security*, *12*(6), 502–512.
15. Sommer, V. (2014). Anomaly detection in the SDN control plane. Master's thesis, Technische Universität München.
16. Zhang, Y. (2013). An adaptive flow counting method for anomaly detection in SDN. In *Proceedings of ACM Conference on Emerging Networking Experiments and Technologies*, pp. 25–30.