SaturationGuard: Control Plane Saturation Attack Mitigation in Software Defined Networks

Neminath Hubballi and Kanishk Patel
Department of Computer Science and Engineering
Indian Institute of Technology Indore, India
{neminath, cse180001025}@iiti.ac.in

***Abstract*—Recent works have shown that the interaction between control and data plane in the Software Defined Networks can be chocked by an adversary with saturation attack. This attack is generated by sending large number of new flows to a switch exploiting the switch-controller communication. A switch sends a packet-in message to the controller if a new flow is seen. A flux of new flows results in a large number of packet-in messages at the controller. In this paper, we present `SaturationGuard` which mitigates this attack by adopting an early attack detection method. An anomaly detection method deployed at the controller observes the patterns of packet-in messages and identifies the attack. In particular, we capture normal interaction between switch and controller using the arrival rate of packet-in messages with a probability distribution. To mitigate the attack, we propose to throttle the bandwidth of the affected switch port in proportion to the arrival rate of new flows. We implement a proof of concept solution with Mininet and an external controller and show that `SaturationGuard` is effective in handling the saturation attacks with early stage detection.**

Keywords*—*Saturation Attack, Anomaly Detection, Mitigation, Bandwidth Throttling, Software Defined Networking

I. Introduction

Software Defined Networking (SDN) simplifies the task of network management by separating the control and data planes in the network. It concentrates the control plane operation into a software entity which has a global visibility of the network. This central entity called the controller makes decisions of what action should be taken on packet(s) at the data plane. The action to be taken is indicated by installing a set flow rules in the relevant switches. Centralized decision making brings several advantages including programmibility, resource optimization and also innovative application development. OpenFlow [1] is the standard interface between the control plane and data plane. SDN has also been used to design new security solutions for the traditional security issues. However, it also brings several new security issues [11] including new DoS attacks. Denial of Service and Distributed Denial of Service attacks have been studied very extensively. Peng et al. [9] argue that several design decisions of TCP/IP network protocols have led to such attacks. More recently these attacks have taken the form of targeting the application layer protocols [14] and also software defined networks [13]. There are methods to detect and possibly mitigate the effects of traditional DoS and DDoS attacks. However the DoS attack on SDN is unique and the conventional methods to handle the DoS attacks will not be effective. In the traditional DoS attack a specific server, application is targeted. However a single attack can be launched against control plane in SDN affecting the entire network.

`Saturation Attack`: This attack is a consequence of scalability issues of OpenFlow. Typically a SDN switch has a set of flow rules installed by the controller and if a new flow is seen for which there is no flow rule, the switch sends a packet-in message to the controller asking for a new flow rule to handle this packet. To mount the attack, an adversary generate a flux of new flows potentially with spoofed addresses which hogs the bandwidth between the switch and controller. This can also overwhelm the controller and switches also become saturated as they have limited buffering ability. A flow is defined with five attributes namely source and destination IP addresses, source and destination port numbers and layer four protocol. By changing one or more attributes in these fields an adversary can generate a flux of flows and generate the attack. As there are many attributes available for manipulation, the attack can be very easily generated with a program which can craft custom packets and send it.

In this paper we describe a method to detect and subsequently mitigate the control plane DoS attacks. Our proposed model has two sub-modules. The first one detects the attacks against control plane using anomaly detection technique and second one takes remedial action to mitigate the attack. In particular we make the following contributions in this paper.
(i) We describe an anomaly detection system to detect attacks against controller by modeling the packet-in messages as a probability distribution.
(ii) We propose a method to mitigate the attack or minimize the effect of attack by throttling the bandwidth of the affected port proportinal to the intensity of the attack.
(iii) Perform simulation experiments with Mininet to validate the proposed method.

II. Literature Review

In this section, we present the related works on saturation attack detection and mitigation. The existing works majorly fall into following three categories.
(i) Proxy-based Mitigation: These works propose to add state information to the switches and convert them to proxies.

AVANT-GUARD [13] described a data-plane solution to handle saturation attacks resulting from TCP SYN flood attacks. It proposes to use Connection Migration where the SDN switch acts as a proxy and handles the TCP 3-way handshake without maintaining state information. Only the flows completing 3-way handshake will be notified to the controller. It proposes to delay the notification till a valid data packet arrives on that TCP connection. Unfortunately this introduces another vulnerability called buffer saturation and also has significant limitations as shown by Ambrosin et al. [5]. Alternatively the Authors of [5] suggest to proxy the TCP connection probabilistically on per IP basis which they adopted in their design of LineSwitch to show that it required much less resources.

(ii) Anomaly based Detection: These methods monitor the state of controller and take necessary action which are either in the form of blocking the source of attack or diverting the packets arriving to other switches or buffer them in the switch. Li et al. [8] propose an anomaly based detection system using the arrival patterns of incoming packet-in messages. They argue that the control plane traffic is self-similar and if an anomaly is detected the model installs a flow rule to redirect the packet-in messages to a cache. OFF-Guard [15] uses a threshold to detect saturation attack by counting the number of packet-in messages. FloodGuard [16] proposes to install proactive-rules to mitigate the attack by observing the controller status. FlowRanger [17] detects anomalies in the incoming packet-in messages and uses job scheduling technique to penalize attack sources. FloodDefender [12] installs a flow rule at the affected switch to detour attack flows to the victim's neighbor switches.

(iii) Source Validation: These methods rely on the fact that spoofed source IP addresses are used for launching the saturation attack. Hence they initiate the validation of source addresses. Zhang et al [18] use this method and deploy verification on switches connecting to end hosts. FSDM [7] also invokes a verification method to validate the source IP address when the attack is detected.

From the above discussion, it is clear that proxy based methods require switch to be intelligent and track TCP state information. This requires additional resources at the switch. In addition successive packets may not flow in the same path, they may not be available at the same switch which makes these proxy schemes ineffective. Further source verification also requires maintaining similar state at the switch which can become bottleneck. Both proxy schemes and source verification techniques requires modification to switch functionality and add complexity. Taking motivation from this, we propose an anomaly based attack detection and mitigation technique in this paper.

III. Proposed Approach

In this section we describe our proposed method `SaturationGuard` for mitigating the control plane saturation attack. Precursor to the mitigation is the detection of attack. We propose a method to detect the attack and subsequently describe a way to handle this. We consider a reference architecture shown in Figure 1 to present our detection and mitigation methods. This has three SDN switches (can be any number), a controller, an attacker and a web server. Dotted lines in the graph indicate a logical connection between the switch and the controller and the normal lines indicate a direct connection. As mentioned earlier, every time a new flow (new combination of packet attributes) is detected the switch sends a packet-in message to the controller. In `SaturationGuard` the attack detection module runs on the controller and for mitigation it communicates an action to be taken to one or more switches. In the following two sub-sections the detection and mitigation techniques are elaborated.

A. Saturation Attack Detection

In order to detect the attack, we model the communication between the switch and controller as seen in a normal condition. There are two ways to model this communication, first is where individual switches communication with controller is modelled and the second is modelling the aggregated communication of all switches. First one can not scale to large networks. Later one has the advantage of being simpler and one model suffices for the entire network. It serves our purpose as it does not matter whether the packet-in messages are coming from a single switch or many switches. In `SaturationGuard`, we represent the arrival of packet-in messages as a probability distribution. The idea is to capture the probability or the likelihood of packet-in message ranges and identify the deviations. In particular, we model it as a Poisson probability distribution [10]. The choice of using this distribution is motivated by the fact that this distribution captures the average number of such messages in an interval and represent the normal behavior. We want to identify the abnormal behavior or deviations particularly large number of packet-in messages using this distribution.

Poisson probability distribution is a discrete probability distribution which uses mean number of events λ obtained from $\mathcal{N}$ observation intervals. In our case this is the number of packet-in messages as observed by the controller. The probability distribution function of Poisson distribution is given by Equation 1.

$$P(X = n) = \frac{e^{\lambda} \times \lambda^{n}}{n!} \tag{1}$$

In this equation, X is a discrete random variable, n is a non negative integer value. $P(X = n)$ represent the probability of random variable X taking the value n. An important feature of this discrete distribution is that, the maximum probability value is around the mean λ. The probability value of random variable decrease on either side of mean. Figure 2 is a sample distribution generated with a λ value of 4.5. By taking the mean number of packet-in messages as seen in the controller, we generate a distribution and to detect the attack, we identify an event which has a low probability value with increased number of packet-in messages. The value of n at this probability will become the threshold for detecting the attack. If the number of packet-in messages are larger than this threshold number, saturation attack is detected. In order to set this threshold value, we use chebyshev's one sided inequality which is given by

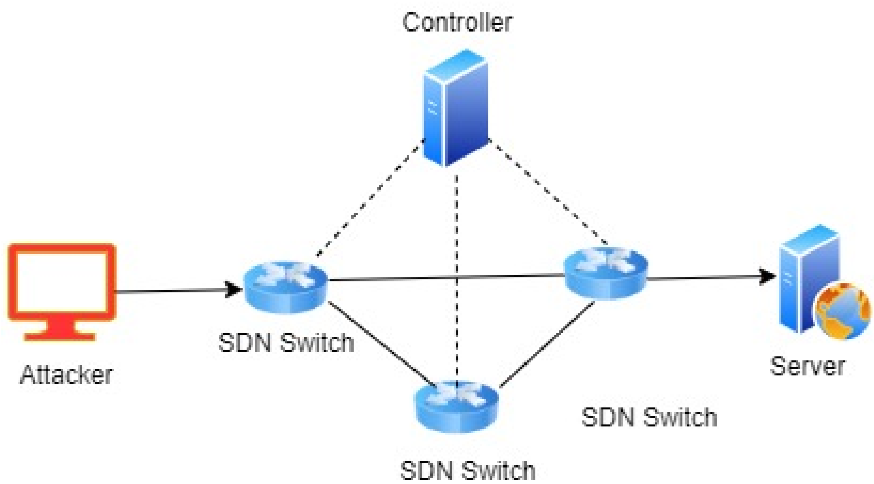

Fig. 1: Control Plane Saturation Attack Mitigation Setup

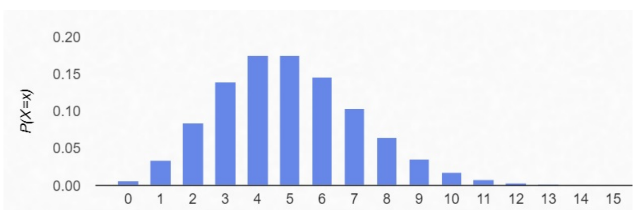

Fig. 2: Poisson Probability Distribution

Equation 2.

$$P(X \geq \lambda + C) \leq \frac{\sigma^2}{\sigma^2 + C^2} \tag{2}$$

In Equation 2 σ is the standard deviation and C is a positive constant. We set this value of $\lambda + C$ as a multiple of λ i.e. $\gamma \times \lambda$ such that a major portion of the probability distribution is covered under normal case and rare events falling in the tail of distribution are detected as saturation attack.

B. Attack Mitigation

Once the attack is detected (by the method in the previous stage) the next step is to minimize the effect of the attack. There are many options one can think of. If the attack is generated using a single source (one source IP address) by modifying the other attributes like destination IP address, layer 4 port numbers, etc mitigation is very easy which is to just block that source. However if the attack is generated with spoofed address (which is likely the case), then blocking is not the solution. We propose to handle this by looking at the port number of switch from which the packet-in messages have originated and reduce the bandwidth of the particular port on that switch proportionately using traffic policing techniques. This we estimate by calculating a probability value which indicates by what factor the bandwidth needs to be reduced. The probability value is calculated as in Equation 3.

$$prob = 1 - e^{\frac{threshold - observed}{threshold}} \tag{3}$$

In this equation threshold is the value used to detect the saturation attack as calculated in the previous phase. `SaturationGuard` requires the bandwidth reduction to be proportional to the probability value calculated in Equation 3. The choice of Equation 3 is motivated by a similar use-case in handling network congestion with Random Early Detection [6] where the packets arriving at a router are dropped based on the queue length which indicate what fraction of the input queue is full. As the probability value ranges between 0.0 to 1.0 and there are infinitely many values in between. For easier handling, we divide the range for probability values into $\mathcal{K}$ blocks as shown in Figure 3. Depending on which block the probability value $prob$ falls, the bandwidth of the port from which these packets are originating is throttled. This is done by defining different transmission rate corresponding to every block and setting the switch to operate at that rate. In the diagram low probability values indicate less aggressive transmission and probability value close to 1 indicate aggressive transmission. Thus if the value is above 0.8 which falls in the last block $T5$, transmission on that port is completely blocked for a fixed duration.

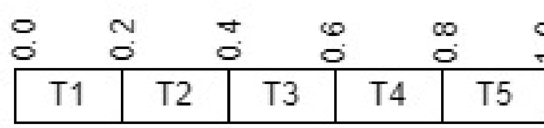

Fig. 3: Probability Ranges

IV. Experiments and Evaluation

In this section we describe the experimental setup and evaluations done with our proposed method `SaturationGuard`.

Experimental Setup: We used two systems running Ubuntu 20.0 having 8 GB RAM with Intel i5 processor. In one of the machine we setup Mininet [2] emulator and in the

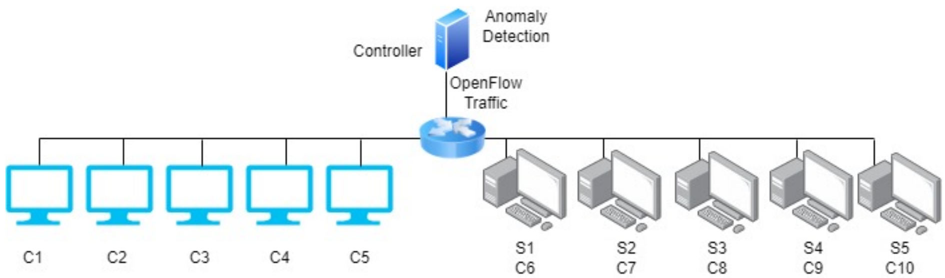

Fig. 4: Experimental Setup

other machine we setup pox controller. The controller was able to install flow rules in the Mininet switch. We designed a network topology with ten systems and emulated it in Mininet. The topology is shown in Figure 4. It has a SDN switch and all the hosts connect to this host in a star topology. In the diagram clients are denoted as C_i's and Servers are marked as S_i's. Out of five servers in two we ran web servers, other two were accessed with iperf tool and the remaining one is pinged by clients. In this setup all the servers also act as clients connecting to other servers. In our experiments, we limit the bandwidth between the switch and controller to 2 Mbps using Netem tool [3] installed on the controller.

Simulating the Normal Traffic: In order to mimic the normal operation in the network, we generated traffic by setting up interaction between different clients and servers. Algorithm 1 shows the connection and service accesses established. Every client randomly chose a server to connect to and establishes a connection request for some service and subsequently sleep for a random amount of time. These interactions between the systems generate some number of flows. The controller installs rules in the switch for those flows. We used a timer of 10 seconds for rule expiry after which the rule will be flushed from the switch. We collected the packet-in messages and calculated the average number of such messages in a window period of 5 seconds. Using this mean number (λ), we generated the probability distribution as shown in Figure 5. As the mean number of packet-in messages were around 30, the distribution graph has the highest probability at this value.

Algorithm 1 Generating Normal Traffic

Input: η - Total Number of Servers
Input: t_1 and t_2: Minimum and Maximum Time Delay

while not interrupted **do**
  $K_1 \leftarrow$ RandomNumber(1,η)
  ConnectToServer(K_1)
  AccessService(K_1)
  $K_2 \leftarrow$ RandomNumber(t_1,t_2)
  Sleep(K_2)

Generating the Saturation Attack: Saturation attack requires generating packets which belong to different flows such that each one of them results in a packet-in message to the controller. This can be generated by changing the source and destination addresses, source and destination port numbers. We used a tool named hping3 [4] to generate the attack. This tool has the ability to send different packet types and using different protocol types. It can also send the packets by randomizing the source and destination addresses.

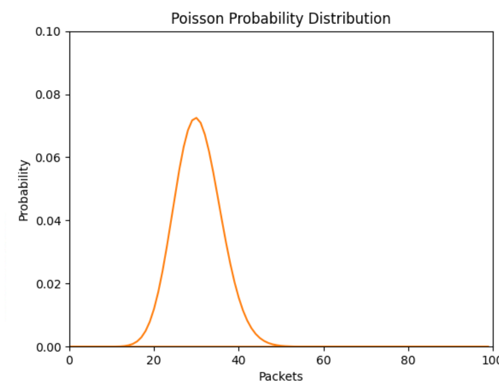

Fig. 5: Probability Distribution of Packet-In Messages

Impact of Attack on Controller CPU Utilization: We generated the attack by sending packets with random addresses (source and destination) from client C1 with different intensity by scheduling the time delay between

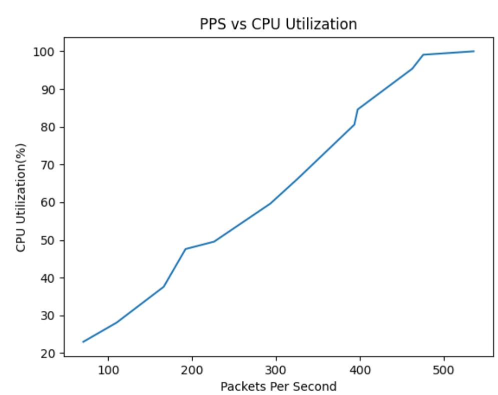

Fig. 6: CPU Utilization

successive packets in hping3. These packets generated packet-in messages to the controller. We measured the CPU utilization on the controller when the attack was on. Figure 6 shows the variation of CPU utilization with attack intensity. The X axis in the graph shows the number of packets per second and Y axis is the percentage of CPU utilization. We can notice that, as the attack intensity increases the CPU utilization also increases almost linearly and saturates at about 500 packets per second rate nearly with 100% utilization. This observation is consistent with other works [16].

Impact of the Attack on RTT: In order to assess the impact of the attack on the communication of other hosts connected to the switch, we measured Round Trip Time (RTT) variation with attacks of different intensity. In particular two client machines (C2 and C3) and one client (C4) and server machine (S1) connected to the switch. This we measure with and without `SaturationGuard`. For the attack detection we use a probability threshold which is 20 times the mean (λ). Figure 7 shows this variation of RTT values. We can notice that till the saturation point is reached the RTT values are very low and more or less constant and when the saturation occurs (which is about 500 PPS) the RTT values increase significantly in the absence of `SaturationGuard`. The low RTT values even when the attack is on is due to the fact that, a small amount of bandwidth available at the switch is good enough to establish a communication. When the saturation attack is peaked, the switch is no longer able to handle the new flows that's when the RTT increases significantly. However, when the proposed mitigation method is used, the RTT values are still maintained around the same range as in the previous case. This is becuase `SaturationGuard` throttled the bandwidth (using traffic policing techniques) of the corresponding input port early on which enabled the flows corresponding to RTT measurement handled successfully.

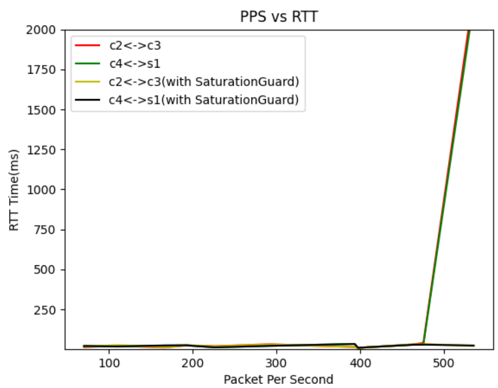

Fig. 7: RTT Variation with Attack

Availability of Bandwidth: `SaturationGuard` throttles the bandwidth of the affected switch based on the intensity of the attack. In order to assess the impact of this when the saturation attack is on, we conducted a study by measuring the available bandwidth between the switch and the controller. We measure this bandwidth by successively generating the attack with different intensities as in the previous case. Figure 8 shows the variation of available bandwidth between switch and controller. We can notice that for lower intensity attacks the available bandwidth difference is small, however as the attack intensity increases, `SaturationGuard` preserves the available bandwidth between switch and the controller and the difference is significant. This is due to the fact that `SaturationGuard` throttles the bandwidth proportionate to the attack intensity. For higher intensity attacks, the rate of throttling is also higher which preserves bandwidth between the switch and the controller.

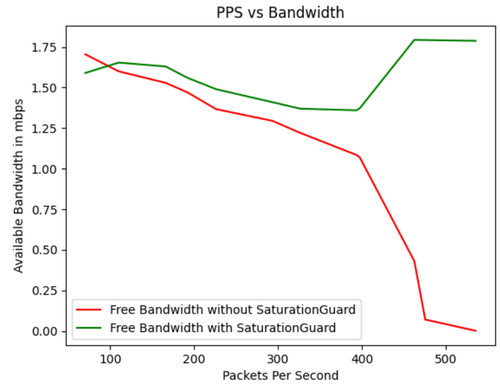

Fig. 8: Bandwidth Available between Switch and Controller

Impact of Detection Threshold on Attack Detection Time: It is worth noting that probability threshold used for detecting the attack determines when the attack is detected. If the threshold is too small, then the attack is detected quickly and bandwidth is throttled quickly. On the other hand if the threshold is high then the attack is detected very late and by that time the attack might have caused significant damage. We study the sensitivity of threshold on the latency in the attack detection. For this we fixed the attack PPS at 500 and measured the delay in the detection of attack i.e. first throttling of bandwidth by varying the threshold used for

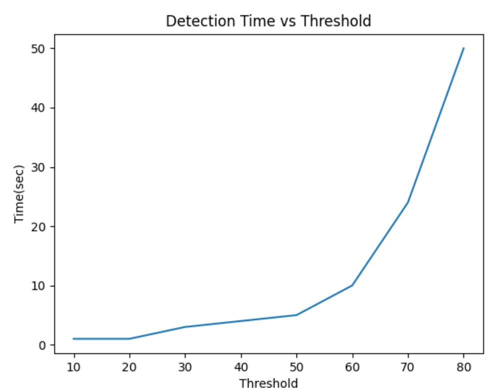

Fig. 9: Attack Detection Time Variation with Threshold

detecting the attack between 10 times of λ to 80 times of the λ. Figure 9 shows the variation of delay in detecting the attack. We can notice that as the probability threshold increases the delay also proportionately increases.

Impact of Blocking the Switch Port: When the attack

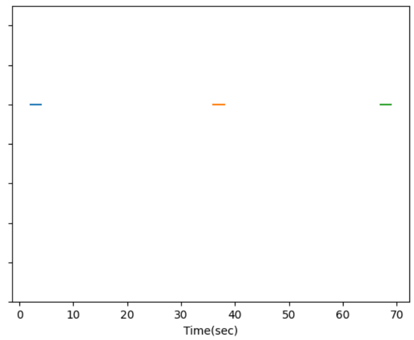

Fig. 10: Port Active Time with Flooding

intensity is very high then the probability of throttling will be close to 1. This in our case is mapped to blocking the port(s) of impacted switch(es). Once a port is blocked, it is blocked for a duration of 30 seconds. Subsequently even if the source is continuing with the attack those packets will not impact the controller. Infinitely blocking a port is not a feasible solution either. In order to validate how a continuous high intensity attack is handled by `SaturationGuard`, we performed an experiment by flooding the switch with randomly generated flows. As the rate is quite high, it triggered blocking of the port. Figure 10 shows (few initial seconds) the on-off periods during which there were some packet-in messages generated or not to the controller. We can see that when the attack is detected for the next 30 seconds there is no communication (indicated with blank) generated and once the port is unblocked the communication is restored for a very short duration of 2 seconds. This is the time lag for detecting the attack at the threshold which is 20 times the mean as seen in Figure 9.

Collateral Damage of `SaturationGaurd`: The proposed solution `SaturationGaurd` can be easily used with switches deployed at the edge of the network where the hosts directly connect to the switch. In this deployment scenario there is no collateral damage of blocking the port of the switch. However, when the packet-in messages are observed from a switch located in the backbone of the network then blocking a port can cause collateral damage impacting other communications passing through that port. We argue that, this is a rare case and happens when the intensity is very high. Otherwise bandwidth throttling at the worst will reduce the available bandwidth to other communications as well. In addition, if the solution is used with every switch it is the edge switch which is likely to report all the packet-in messages rather than a switch in the backbone. In order for a switch in the backbone to generate these many packet-in messages, an adversary has to intelligently craft an attack such that switches at the edge do not generate packet-in messages, which we believe is not easy to achieve.

V. Conclusion

Saturation attack chokes the communication between a SDN controller and switch by sending large number of new flows. This can cause Denial of Service to other legitimate traffic and services as these connections are affected. In this paper, we described the working of `SaturationGuard` which can mitigate the effect of such attacks by throttling the rate of transmission at the affected switches. An anomaly detection system running at the controller observes the rate of arrival of packet-in messages and detects the attack. By identifying the infected ports of switch, it throttles the bandwidth available at the switch in proportion to the attack intensity. We implemented and evaluated a working prototype of `SaturationGuard` with Mininet and an external controller.

References

[1] https://www.section.io/engineering-education/openflow-sdn/ (accessed22-04-2022).
[2] http://mininet.org/(accessed29-04-2022).
[3] https://www.linux.org/docs/man8/tc-netem.html(accessed29-04-2022).
[4] https://www.kali.org/tools/hping3/(accessed28-04-2022).
[5] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran. Lineswitch: Tackling control plane saturation attacks in software-defined networking. *IEEE/ACM Transactions on Networking*, 25(2):1206–1219, 2017.
[6] B. B. et al. Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309, RFC Editor, April 1998.
[7] X. Huang, K. Xue, Y. Xing, D. Hu, R. Li, and Q. Sun. Fsdm: Fast recovery saturation attack detection and mitigation framework in sdn. In *MASS'20: IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 329–337, 2020.
[8] Z. Li, W. Xing, S. Khamaiseh, and D. Xu. Detecting saturation attacks based on self-similarity of openflow traffic. *IEEE Transactions on Network and Service Management*, 17(1):607–621, 2020.
[9] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Computing Surveys.*, 39(1):1–42, 2007.
[10] S. M. Ross. *A First Course in Probability*. Fifth edition, 1998.
[11] S. Scott-Hayward, G. O'Callaghan, and S. Sezer. Sdn security: A survey. In *SDN4FNS'13: IEEE SDN for Future Networks and Services*, pages 1–7, 2013.
[12] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui. Flooddefender: Protecting data and control plane resources under sdn-aimed dos attacks. In *INFOCOM'17: IEEE Conference on Computer Communications*, pages 1–9, 2017.
[13] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *CCS '13: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pages 413–424, 2013.
[14] N. Tripathi and N. Hubballi. Application layer denial-of-service attacks and defense mechanisms: A survey. *ACM Computing Surveys.*, 54(4), 2021.
[15] H. Wang, L. Xu, and G. Gu. Of-guard: A dos attack prevention extension in software-defined networks. In *USENIX'14:*, pages 1–2, 2014.
[16] H. Wang, L. Xu, and G. Gu. Floodguard: A dos attack prevention extension in software-defined networks. In *DSN'15: 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 239–250, 2015.
[17] L. Wei and C. Fung. Flowranger: A request prioritizing algorithm for controller dos attacks in software defined networks. In *ICC'15: IEEE International Conference on Communications*, pages 5254–5259, 2015.
[18] M. Zhang, J. Bi, J. Bai, and G. Li. Floodshield: Securing the sdn infrastructure against denial-of-service attacks. In *TrustCom/BigDataSE'18: 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering ()*, pages 687–698, 2018.