2022 IEEE 23rd International Conference on High Performance Switching and Routing (HPSR) | 978-1-6654-0607-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPSR54439.2022.9831333

LLDM: Low-Latency DoS Attack Detection and Mitigation in SDN

Zixu Huang*, Xuanbo Huang*, Jian Li*, Kaiping Xue*†§, Qibin Sun*, Jun Lu†
* School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China
† Department of EEIS, University of Science and Technology of China, Hefei, Anhui 230027, China
§Corresponding author,lijian9@ustc.edu.cn (J. Li), kpxue@ustc.edu.cn (K. Xue)

***Abstract*—Software-Defined Networking (SDN) is a new and highly flexible network architecture, but the bottleneck between the control plane and the data plane makes it vulnerable to the control plane saturation DoS attacks. When the attack happens, traditional schemes in DoS scrubbing agent use a binary classification and a First In First Out (FIFO) queue to filter attack flows. However, this scheme is inimical to the end-to-end latency of benign traffic. To tackle this issue, we propose LLDM, leveraging a dynamic priority scheme and a priority queue to detect, mitigate the attacks while ensuring low latency for benign traffic. After detecting the attack, LLDM leverages a two-phase scheme for mitigation. First, LLDM marks packets from the ports under attack as suspicious and migrates them to the mitigation agent. Then, the dynamic priority manager assigns each packet a priority corresponding to its legality, which is used in the priority queue for DoS scrubbing. We evaluate LLDM in a simulation SDN environment. The experimental results show that LLDM can reduce 90.4% of the queuing delay compared with the traditional scheme under a 5000 Packets Per Second (PPS) attack, and it is also resistant to more sophisticated attacks. Under the high rate attack of 50000 PPS, LLDM installs a flow rule for legitimate traffic in 0.2 seconds. Moreover, for benign HTTP requests, LLDM can keep the request time at 1.39 seconds.**

***Index Terms*—Software-Defined Networking, DoS Attack, Priority Queue, Low Latency**

I. INTRODUCTION

Software-Defined Networking (SDN), a new and highly innovative network architecture, has been widely used in data center and cloud computing in recent years. By decoupling the control plane from the data plane, SDN is more flexible than traditional networks. A logically centralized controller can provide a global view of the network, facilitating fine-grained precise network control. OpenFlow [1] is the most commonly used protocol for communication between the control and the data plane. OpenFlow suggests that switches process incoming packets according to the flow rules. When a packet does not match any flow rules, the switch sends a PACKET-IN message to the controller for instructions.

However, Shin *et al.* [2] showed that this design brings risks of Denial of Service (DoS) attacks, named the control plane saturation attacks. The attacker uses massive malicious table-miss packets to trigger PACKET-IN messages, exploiting the communication channel, exhausting the controller resources and making the benign flow request unresponsive.

Previous works migrate the suspicious flows to a DoS scrubbing agent to protect the control plane from exhaustion [3]–[6]. The scrubbing agent can either restrict the sending rate [3] or filter malicious flows based on whitelist [4], [6] and blacklist. However, these methods based on binary classification can only filter the attack flows in a first-in-first-out (FIFO) manner. Figure 1 shows the traditional binary classification method that scrubs the mixed network flows with a single queue FIFO module. When the attack flows are in a large number, the benign traffic has to wait and queue in the scrubbing agent, making them suffer from long end-to-end delays. Besides, whitelist based method cannot deal with new benign flows during the attacks.

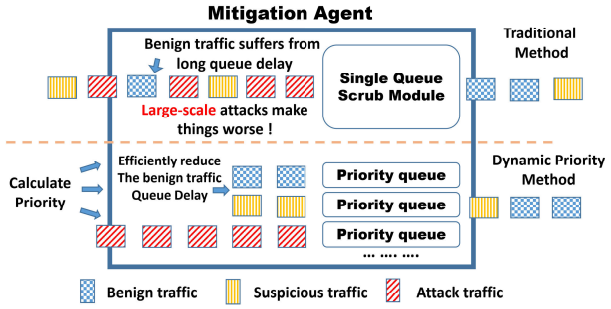

Fig. 1. The traditional single queue mitigation agent v.s. The dynamic priority based priority queue mitigation agent

We design LLDM to solve these problems. After attack detection, we design a two-phase mitigation scheme to scrub the attack flows. First, we launch a coarse-grained filtration scheme, dividing the traffic by the switch interface and migrating the minimal suspicious traffic to the mitigation agent. Next, as Figure 1 shows, the dynamic priority manager gives each flow a priority and updates it continuously and dynamically, based on the behavior of the flows. Then, the mitigation agent enqueues each flow to their corresponding priority queue to quickly deliver the high-priority packets, reducing their queuing delay. We implement a prototype LLDM system in an SDN simulation environment with the help of Mininet and Ryu Controller. The results show that our method can defend against DoS attacks while maintaining low latency for benign traffic.

The main contributions of our work can be summarized as follows:

- We propose LLDM, a DoS attack detection and mitigation framework whose goal is to reduce the end-

to-end latency of benign traffic. By introducing the two-phase mitigation scheme, we reduce the attack influence on benign traffic and decrease the asymmetry cost on attackers and defenders.

- We design and implement a dynamic priority manager in the DoS scrubbing agent for SDN saturation attacks. Our method leverage dynamic priority to ensure non-whitelist benign traffic communication and leverage the priority queues to reduce the end-to-end latency of benign traffic.
- We implement the LLDM prototype system in a simulation SDN platform and measure the reduction in the latency of benign packets through experiments. The experimental results show that LLDM can reduce 90.4% of the queuing delay compared with the traditional FIFO scheme, and keep the HTTP response time within 1.39 seconds.

The rest of this paper is structured as follows: We discuss some related works in section II. Then, Section III introduces the whole framework and the detailed design of LLDM. Next, Section IV present the evaluation of experiments. Finally, we draw our conclusion in Section V.

II. Related Works

1) Attack Detection: FloodGuard [3], and FloodDefender [7] leverage the PACKET-IN rate for attack detection, which is the critical parameter to determine the attacks. DETpro [8] uses machine learning method to detect attacks. Dong *et al.* [9] classified the network state using the sequential probability ratio test. Li *et al.* [5] proposed a detection method based on self-similarity of the OpenFlow traffic. Fouladi *et al.* [10] proposed a time-series analysis for SDN to judge attacks. Also, entropy-based attack detection has been widely used [4], [11]. In this paper, we leverage the attack detection method in [6], which use the control channel distribution rate and the entropy to detect the attacks.

2) Attack Mitigation: AVANT-GUARD [12] and Line-Switch [13] extend the data plane, using a proxy to handle TCP-based attacks to avoid the control plane overload. However, as table-miss flows are protocol-independent; attackers can manipulate non-TCP flows to attack the network. FADM [4], and FSDM [6] migrate the suspicious flows to a DoS scrubbing agent and use whitelist to filter attack flows, but it is difficult to distinguish a benign new packet during an attack. FloodDefender [7] uses the packet frequency and the existence of pair-flow to verify the legality of a packet. However, these works migrate the suspicious flows to a DoS scrubbing agent and use a binary classification method, which harms the end-to-end latency of benign traffic. As shown in Figure 1, traditional binary classification scrubs the mixed network flows with a single queue module. However, when the attack flows are in a large number, the benign traffic has to wait and queue in the scrubbing agent, making them suffer from a long end-to-end delay. In this paper, we propose a dynamic priority based scrubbing method to reduce the scrubbing price of benign traffic.

III. LLDM System Design

A. System Overview

In this section, we introduce the architecture of LLDM. Fig. 2 shows the blueprint of our method.

Our system contains two key modules: the Attack Detector module and the mitigation agent. The Attack Detector is an application on the controller. When the network is working, this module collects traffic statistics on each switch port. Once an attack is detected, this module locates the ports where the attack is from and marks the packets from these ports as suspicious packets. And then, the Attack Detector installs a flow rule on the victim switch to migrate the suspicious packets. Upon receiving the packets, the mitigation agent begins to filter the packet based on a priority queue, and the mitigation proxy generates PACKET-IN messages for benign traffic to the controller. The PACKET-IN messages restore the original information such as port number, so that the mitigation agent is transparent to the controller and the apps running on it.

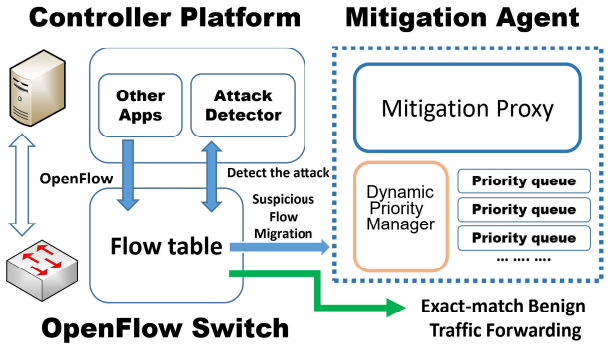

Fig. 2. System Design and Working Process of LLDM. Suspicious flows are migrated to mitigation agent and scrubbed, then benign traffic is forwarded to controller.

B. Attack Detection

We design an attack detection module to detect the occurrence of attacks, and then it makes a rough classification of traffic. This module can be divided into two functions, attack detection and traffic classification.

1) Analyze: Typically, an attacker takes control of a botnet and sends attack traffic from multiple hosts to the SDN network. We assume that the attackers tend to launch table-miss flows, taking the opportunity to generate numerous PACKET-IN messages. Attackers can reconnaissance some matching fields of the flow rules to achieve that goal [14]–[16].

Based on this attack model, we find two distinct characteristics. First, attack traffic is always in a large number. The attacker uses malicious requests to exhaust the bandwidth between the controller and the switch. Once the controller receives PACKET-IN requests far beyond controller processing ability, it can be considered an attack. The second is the low frequency, which is a unique feature of the SDN saturation attack. Under normal circumstances, users usually access

limited addresses in a short period, and we can treat these packets as identical. Continuously sending the same flow is not profitable for the attackers. Because only the new flows can trigger the PACKET-IN messages, the repeated flows can not exhaust the control plane. Therefore, the attacker will continually use the new stream, resulting in a rare high entropy. So, we can then set thresholds on these features to detect an attack. We deploy the Attack Detector in the controller to get the global view and save the switch resources.

2) Features Collection: We can set a time window *T*, and count the flow within each time window. For the number of packets, we only need to count the number of PACKET-IN messages. The PACKET-IN messages contain the *dpid* and the *port-numbers* information, which help us to categorize. For the entropy feature, we use five-tuple to distinguish the packets. First, we need to know the number of each packet in this time window. The packet received by the switch can be divided into two types, one is the packet not matching flow table, which triggers the PACKET-IN message, so the number of such packets can be inferred from the five-tuple information contained in the PACKET-IN message. The other is the packet that hits the flow table, which does not cause the PACKET-IN message. We can use the flow table statistics function, sending Stat-Request at the beginning and end of each time slice. According to the flow table counter information, we can calculate the number of these packets. For example, for a flow table $flow_i$ that starts with a counter value of 50 and ends with a counter value of 59, the number of packets S_i is 9. After obtaining the number of packets $(S_1, S_2, S_3, ..., S_i)$, we can calculate the probability distribution $(P_1, P_2, P_3, ..., P_i)$ of the packets, and then calculate the entropy of the packets by the following formula:

$$H = -\sum_{i=1}^{n} P_i * log(P_i).$$

3) Threshold Setting: We need two thresholds, S_t for the number of packets and H_t for entropy. The first threshold S_t means the load capacity of SDN, which is related to many factors. The bottleneck is most likely to be the processing ability of the controller and the bandwidth between the controller and the switch. For the processing ability, we can obtain it through the pressure test, we denote it as S_{ctrl}. For the bandwidth between the controller and the switch, assuming available bandwidth is *W*, the number of PACKET-IN messages that this link can tolerate is $\frac{W}{N}$, where *N* is the average size of each PACKET-IN message. But if we have already detected $\frac{W}{N}$ PACKET-IN messages, the actual attack might be more serious than that, leading to queuing delay in the switch. Furthermore, because the link is filled, our Stat-Request and PACKET-IN messages from other ports will be blocked. To make room, we need a coefficient λ to report an attack before the bandwidth is exhausted. λ can be understood as maximum channel usage and $(0 < \lambda < 1)$ Finally, S_t is given by:

$$S_t = \min\{S_{ctrl}, \quad \lambda \cdot \frac{W}{N}\}.$$

The second threshold for H_t can be set empirically. By statistical observation of the normal case of the entropy of packets, we set a threshold H_t far outweigh that.

4) Classify: Next is the first phase of the two-phase filter. The attack detection mechanism returns a set of port numbers $\{Port_1, Port_2, Port_3, ..., Port_i\}$. We send Flow-Mod messages to these switches, with the lowest priority flow tables that forward all traffic from these ports to the mitigation agent. The lowest priority is set to prevent packets that already have other flow table matches from being incorrectly matched to this mitigation flow table. All the attack traffic is forwarded to the mitigation agent without any impact on the other port users.

C. Mitigation Agent

Mitigation agent is the second phase of the two-phase filter. After the first phase of filtration, all suspicious traffic will be migrated to this module. We can not simply discard these traffic because some benign traffic also mix in them. Our second phase filter distinguish these traffic, and generate the PACKET-IN messages. The framework of this module is shown in Fig. 3

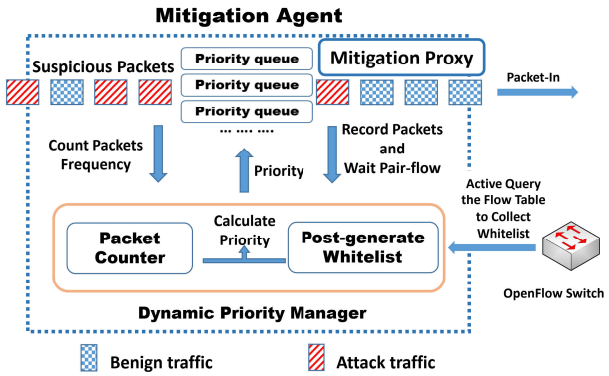

Fig. 3. Design of the mitigation agent and the working process of the dynamic priority manager. Benign packets are more likely to obtain higher priority

The detailed process is shown as follows:

- When an attack occurs, the mitigation agent receives all packets on the link.
- These packets are given the appropriate priority and enter the priority queue. The priority queue keeps a transmission rate that the switch can tolerate.
- The mitigation proxy module generates PACKET-IN message for each packet leave from the priority queue, and sends it to the controller with an information of the original switch and switch port.
- At the same time as the step 3, whitelist record the packets and expect pair-flow, adding the packets with pair-flow to the whitelist.

Next, we take a closer look at the implementation of each component.

1) Priority Queue: We mark packets with a five-tuple, and packets that have the same five-tuple are treated as the same packet. We need to give a priority to the packet before it enters

the priority queue. We design a dynamic priority manager module to give its priority, benign packets are easier to get high priority. High priority means that packets can preempt the limited link resources and controller resources without waiting for other packets, which makes the packet delay very low.

2) Dynamic Priority Manager: Packet priority is not fixed, we adjust the packet priority according to the characteristics of the packet and network behavior. We devised two strategies, a post-generate whitelist and a packet counter. The whitelist priority is higher than the packet counter. For a packet, we first check if it is in the whitelist, if so, we give it a maximum priority. Conversely, if it is not in the white list, the packet counter determines its priority. The packet counter counts times the packets occur, which is a condition for determining priority. For a new packet, we give it priority 1, a low priority. For a packet that has already appeared, its priority is the number of times it has appeared. For example, if mitigation agent receives five consecutive packets from a same TCP connection; dynamic priority manager gives the first packet priority 1, the second packet priority 2, and so on. When the same packet repeats, its priority becomes higher and higher, and it is easier to pass through the priority queue. This strategy makes sense because the frequency of benign packets is usually much higher than that of illegal packets.

The whitelist collects after the attack, therefore, we call it a post-generate whitelist. For each packet that leaves the priority queue, the whitelist records its five-tuple. If its pair-flow is discovered after a period of time, the packet is added to the whitelist with its pair-flow. But not all pair-flow comes to the mitigation agent, because a stream and its pair do not come from the same switch port. If the port that pair-flow comes is not under attack then the traffic from it will not be migrated to the mitigation agent. To solve this problem, the whitelist then "actively queries" a packet before long, requesting the switch for flow table information. We assume that the switch is reliable, and a corresponding flow table on the switch means the flow is benign. This is another case where we add it to the whitelist. On the contrary, if no pair-flow is observed, the packet priority is reduced.

The whitelist doesn't work for an entire new flow, but it can work for its pair-flow. Or, when the packet re-enters the mitigation agent after the flow table has expired, the white name will also come into play, which will help us deal with some of the more sophisticated attacks. The whitelist does not block packets and therefore does not introduce additional latency.

3) Forwarding: Finally, the mitigation proxy generates the PACKET-IN message instead of the switch, which identifies the switch *dpid* from which the packet originated and the *port-number*. The mitigation agent is transparent to the APP, so the flow table issued by the controller is installed directly on the original switch.

IV. EVALUATION

A. Experiment Description

We carry out extensive experiments in a simulation testbed to evaluate LLDM functionality for answering the following questions:

- Can LLDM effectively reduce the benign traffic queuing delay compared with traditional method FIFO and how much it can reduce?
- In what scenarios will our post-generated whitelist work? What is the effect of it compared with a scheme without post-generated whitelist?
- How does LLDM perform under high attack rate? Can LLDM keep HTTP response time within a reasonable range?

B. Build Simulated Environment

We built a simulated environment on a computer with Ryzen5 3600 @3.925Ghz CPU and 8 GB memory to evaluate our solution, and the network topology is shown in Fig. 4. We used Mininet [17] to create the SDN environment, which consists of eight virtual hosts and three OpenVswitches. One host works as a web server; one deploys mitigation agent; two works as test hosts; and the other four behave as normal users. The bandwidth of each link is set to 1 Gbps, which is sufficient for our experiment. The SDN Controller uses the Ryu [18], and runs two main APPs. The first is Flow-Manager, which is used to implement the basic functions of the network, sending flow tables, controlling forwarding, etc. To simplify the experiment, we use five-tuple as the matching field of the flow table. The actions of the flow table are forward and drop only. Pressure tests have shown controller can handle up to 1,000 PACKET-IN messages per second. The second is our Attack Detector, which continuously collects packet information for judging attacks and classification.

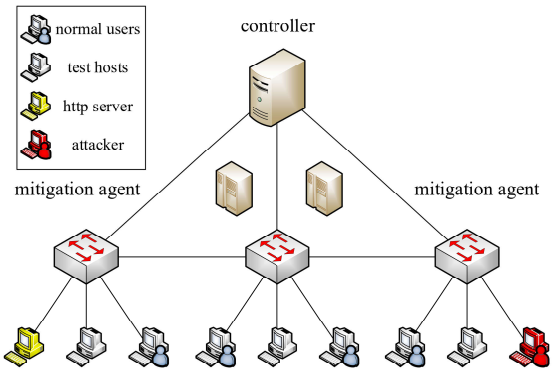

Fig. 4. Experimental Network Topology with LLDM Deployed

The controller connects with switch and mitigation agent through TCP connection, and the connected components are distinguished by unique *dpid*. But mitigation agent does not have *dpid*, theoretically, the controller can not distinguish the flow from it. Mitigation agent strictly controls the transmission rate to the controller, sending no more than 1,000 packets with

same *dpid* per second. Each host has its own IP address and MAC address. In order to simulate the real situation, we deploy scripts on the hosts for simulating user. The scripts control these hosts to communicate with each other at random, or send HTTP request to the web server.

An attacker can attack SDN from any port of any switch and uses *Trafgen* to construct the packet, which uses forged IP address and MAC address, and uses different protocols. Attack rate according to different requirements, is set to 1000 to 50000 PPS, far beyond the controller processing capacity.

C. Experiment

1) Compare with FIFO: Firstly, we research the queuing delay, which is one of the main sources of packet delay. We implement a simple baseline scenario with a FIFO queue implemented mitigation agent. New mitigation agent still filters packets by packet frequency, but without dynamic priority manager, it can only filter packets once per second. Usually, the storage is not infinite, and we assume that mitigation agent can hold up to 2,500 packets at a time. This means that only 2,500 packets per second will be checked under the scheme, and the extra packets will be blocked. Our attack rate is between 1000 and 5000 PPS, up to twice the system processing ability, in which case the packets will suffer a huge queuing delay. We calculate the queuing delay by recording the packet time stamp from entering the queue to leaving the queue. We compared our method using the priority queue with baseline, and the result is shown in Fig. 5.

Using the traditional FIFO, when the attack rate is below 2500 PPS, the packets can be processed in time and the queuing delay is even lower than our priority queue. However, when the attack rate is more than 2500 PPS, the queuing delay has a sudden change from 28.4ms to 379.8ms. Then, the queuing delay increase with the increase of attack rate, and reach 564.2ms in the worst case of 5000 PPS. When using the priority queue, the packets do not need to wait for the attack packet which arrives earlier than them, so the queuing delay is hardly affected by the attack rate. Even if the attack rate is 5000 PPS, the queuing delay is still only 54.3ms, 90.4% less than FIFO.

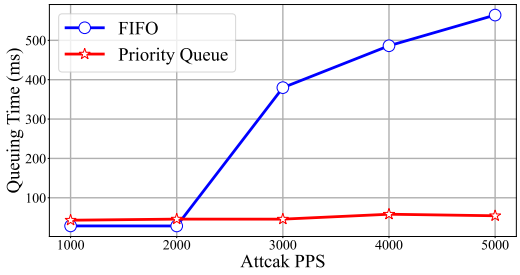

Fig. 5. Queuing Delay with different queue structures

2) Post-generate Whitelist: The whitelist helps us deal with smarter attacks. Assuming the attacker knows that we use the packet frequency feature to identify the attack traffic, attacker will replay some traffic later to make its traffic more like benign traffic. We evaluate our system long enough after the attack, during which time the attacker has replayed most of the attack traffic, and the earlier benign flow tables have expired, therefore, traffic needs to migrate to the mitigation agent again. The total attack rate is fixed at 6000 PPS, and Fig. 6 shows that without post-generate whitelist, benign requests can only continue to retransmit packets until the priority is higher than the attack packet. Since HTTP requests use TCP connections, the retransmission interval increases exponentially, which is reflected when the number of replay times does not exceed 3. When the number of replays is more than three, the effective attack rate has become very low, the HTTP response time is only slowly rising, reaching its maximum value 13.78s when the number of replay times is five. When the replay time reaches six, the effective attack rate is 1000 PPS, so low that the controller can handle it alone, and therefore the HTTP response time drops. Our post-generate whitelist avoids multiple retransmissions under these circumstances, and keeps the HTTP response time under 0.6 seconds.

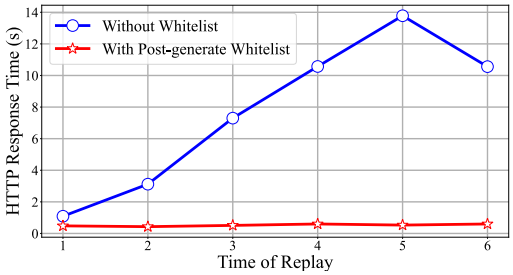

Fig. 6. HTTP Response Time under replay attack

3) High Pressure Scenario: Finally, we evaluated the performance of our system under high pressure scenario. In this experiment, mitigation agent sends no more than 25 packets with same *dpid* and *port-number* per second. To better reflect the actual network scenario, we evaluated the worst-case performance of packet latency distribution. The worst-case occurs between the time the first benign packet arrives and the time the flow table is installed. During this time, packet priority will gradually increase from the lowest, and the white list can not do anything. We use the of packet latency distribution in this period as the worst-case latency distribution. During this time we send continuous ICMP packets whose RTT value serves as the packet latency. We also tested the length of this worst-case period, which we call the flow table installation time. Fig. 7 shows the network packet latency distribution and TABLE I shows time of worst-case period under high attack rate.

TABLE I
FLOW RULE INSTALLATION TIME IN THE WORST-CASE

Attack PPS	10000	20000	30000	40000	50000
Time (s)	0.11	0.14	0.18	0.15	0.20

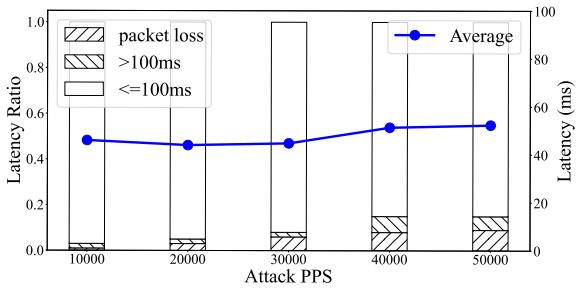

Fig. 7. Packet Latency under different attack PPS

With our dynamic priority manager, high attack rate has little effect on the average delay of continuous packets, with a maximum of 52.31ms. Most packets can be treated as low delay packets($\leqslant 100ms$), which make up no less than 85% of the packets. At the attack rate of 50000 PPS, the packet loss rate reached 9%. But that is the worst-case scenario. Once the flow table is installed, the packets can be forwarded normally, which means that 9% of the high packet loss rate will only occur during the worst-case period of 0.2 seconds. All these results show that our scheme is very effective in reducing the delay.

HTTP requests are more representative of general network activity, and we also evaluate the performance of HTTP requests in high pressure scenario. Each HTTP request is an entirely new access, and the average HTTP request time is shown in TABLE II. LLDM can always keep HTTP response times within acceptable range. At the worst case that attack rate comes to 50000 PPS, it also not leads to high latency, and a whole HTTP request can be completed in 1.39 seconds.

TABLE II
HTTP RESPONSE TIME UNDER HIGH ATTACK RATE

Attack PPS	10000	20000	30000	40000	50000
HTTP Response Time (s)	1.08	1.09	1.17	1.21	1.39

V. CONCLUSION

In this paper, we proposed LLDM, a low latency DoS attack detection and mitigation framework in SDN. Firstly, we designed an attack detector module that can locate the attack source, and quickly separate the attack traffic and benign traffic, migrating suspicious traffic to the mitigation agent. Next, we designed a novel dynamic priority manager to update packet priority accurately by packet counter and post-generate whitelist. Then, we used a dynamic priority method and a priority queue instead of the traditional FIFO queue to scrub suspicious flows. The benign traffic that behaves legitimately will have a high priority and be forwarded quickly. The experimental results show that LLDM can guarantee the quality of communication, significantly reducing the negative impact of the attack.

ACKNOWLEDGMENT

This work is supported in part by the National Natural Science Foundation of China under Grant No. 61972371 and No. U19B2023, and Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant No. Y202093.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

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

[3] H. Wang, L. Xu, and G. Gu, "Floodguard: A dos attack prevention extension in software-defined networks," in *Proceeding of the 2015 Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2015, pp. 239–250.

[4] D. Hu, P. Hong, and Y. Chen, "FADM: Ddos flooding attack detection and mitigation system in software-defined networking," in *Proceedings of the 2017 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2017, pp. 1–7.

[5] 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*, vol. 17, no. 1, pp. 607–621, 2019.

[6] 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 *Proceedings of the 2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2020, pp. 329–337.

[7] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "FloodDefender: Protecting data and control plane resources under sdn-aimed dos attacks," in *Proceedings of the 36th IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2017, pp. 1–9.

[8] Y. Chen, J. Pei, and D. Li, "Detpro: A high-efficiency and low-latency system against ddos attacks in sdn based on decision tree," in *Proceeding of the 2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[9] P. Dong, X. Du, H. Zhang, and T. Xu, "A detection method for a novel ddos attack against sdn controllers by vast new low-traffic flows," in *Proceeding of the 2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[10] R. F. Fouladi, O. Ermiş, and E. Anarim, "A ddos attack detection and defense scheme using time-series analysis for sdn," *Journal of Information Security and Applications*, vol. 54, p. 102587, 2020.

[11] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn," in *Proceedings of 2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 27–35.

[12] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 2013 ACM Conference on Computer and Communications Security (CCS)*. ACM, 2013, pp. 413–424.

[13] 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*, vol. 25, no. 2, pp. 1206–1219, 2016.

[14] M. Zhang, G. Li, L. Xu, J. Bai, M. Xu, G. Gu, and J. Wu, "Control plane reflection attacks and defenses in software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 623–636, 2020.

[15] J. Sonchack, A. Dubey, A. J. Aviv, J. M. Smith, and E. Keller, "Timing-based reconnaissance and defense in software-defined networks," in *Proceedings of the 32nd ACM Annual Conference on Computer Security Applications (ACSAC)*. ACM, 2016, pp. 89–100.

[16] S. Achleitner, T. La Porta, T. Jaeger, and P. McDaniel, "Adversarial network forensics in software defined networking," in *Proceedings of the Symposium on SDN Research*. ACM, 2017, pp. 8–20.

[17] "Mininet," [Online], 2021, available: http://mininet.org/.

[18] "Ryu controller," [Online], 2021, available:https://osrg.github.io/ryu/.