PAPER

MinDoS: A Priority-Based SDN Safe-Guard Architecture for DoS Attacks

Tao WANG[†a)], **Hongchang CHEN**[†], ***Nonmembers*, *and* Chao QI**[†], ***Member***

SUMMARY Software-defined networking (SDN) has rapidly emerged as a promising new technology for future networks and gained considerable attention from both academia and industry. However, due to the separation between the control plane and the data plane, the SDN controller can easily become the target of denial-of service (DoS) attacks. To mitigate DoS attacks in OpenFlow networks, our solution, MinDoS, contains two key techniques/modules: the *simplified DoS detection module* and the *priority manager*. The proposed architecture sends requests into multiple buffer queues with different priorities and then schedules the processing of these flow requests to ensure better controller protection. The results show that MinDoS is effective and adds only minor overhead to the entire SDN/OpenFlow infrastructure.

key words: *software-defined networking, denial-of-service attack, priority queue, round-robin scheme*

1. Introduction

Software-defined networking (SDN) has rapidly emerged as a new networking paradigm that has considerably modified the traditional network architecture and garnered much attention from both academia and industry. By decoupling the control plane from the data plane, SDN provides more fine-grained network management services. However, due to the separation between the control plane and the data plane, the controller is more likely to be the target of DoS attacks. As illustrated in Fig. 1, attackers may launch DoS attacks toward the SDN controller simply by sending a large number of spoofed packets (**Step** 1). These incoming packets will trigger table-miss events (**Step** 2) and make the switch send *packet-in* messages (**Step** 3) to the controller. Consequently, the controller must handle these spoofed packets to create flow entries. This will rapidly consume the computational resources of the controller and overload the flow table space of the switches (**Attacks** 1 and 2). In this state, the attacked controller can only slowly respond to legitimate flow requests (**Steps** 4 and 5). Moreover, the DoS attack may cause a cascading failure of controllers [1] in the SDN multiple-controller model [2], which is even more harmful to the SDN controller.

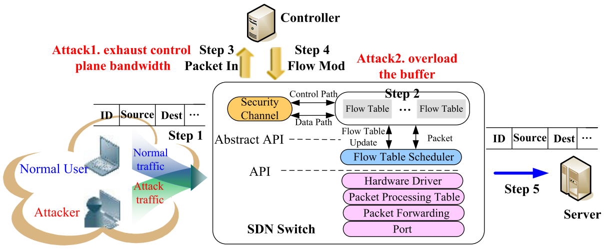


Fig. 1 Example of DoS attacks on an SDN controller.

With the development of SDN technology, some research has been conducted to address this problem. For example, Avant-Guard [3] introduced *connection migration* and *actuating triggers* into the SDN architecture to defend against SYN flood attacks; however, these measures do not prevent any other types of DoS attacks on the SDN. FloodGuard [4] uses a *proactive flow rule analyzer* and *packet migration* to defend against data plane saturation attacks; however, these methods are too costly for most implementations. Previous works [5], [6] have employed self-organizing maps (SOM) to identify abnormal traffic and thus defend against DoS attacks; however, due to the high overhead required for classification, the method cannot be implemented in real time. Yan [7] proposed a solution to detect DDoS attacks based on a fuzzy synthetic evaluation decision-making model. Although it is a lightweight detection method, its detection accuracy is barely satisfactory. All of the above studies have obvious shortcomings. In addition, most of the solutions focus on the data plane to ease the attack traffic, which is not consistent with the trend of SDN security development. Therefore, we propose MinDoS, an efficient and lightweight defense architecture for SDN networks, to mitigate DoS attacks and ensure better controller protection.

The MinDoS architecture has the following four advantages:

- MinDoS is jointly designed with other existing DoS detection algorithms to provide high detection accuracy and ensure the protection of entire networks.
- MinDoS is implemented on the control plane, which is consistent with the trends of SDN security development and does not require any modifications to the data plane. Therefore, MinDoS is easier to deploy than previous solutions.
- MinDoS is valid for all types of DoS attacks.
- MinDoS adds little overhead into the entire SDN/OpenFlow infrastructure.

The remainder of this paper is organized as follows. In Sect. 2, some related works are introduced. We present the detailed design in Sect. 3, followed by a performance evalu-

---

Manuscript received December 27, 2017.
Manuscript publicized May 2, 2018.

[†] The authors are with National Digital Switching System Engineering and Technological Research Center, Zhengzhou 450003, China.
a) E-mail: wangtaogenuine@163.com

DOI: 10.1587/transinf.2017EDP7419

ation in Sect. 4. The last section presents the conclusions of the study.

2. Related Work

The emergence of SDNs provides a new paradigm for solving the problems of the traditional network architecture. By decoupling the control plane from the data plane, an SDN can provide more fine-grained network management services. However, because of this same separation, the security of the SDN is open to attack and has thus received considerable research attention. Many works have recently been published to address the security issues in SDNs. For example, FRESCO [8] proposed the development of OpenFlow security applications. Additionally, a software extension to NOX was proposed in FortNoX [9], which offers role-based authorization and security constraint enforcement for the NOX controller. Both SE-Floodlight [10] and Rosemary [11] proposed a security enforcement kernel to improve the robustness of SDN controller. In addition, some recent works [12]–[14] have noted that the SDN controller is vulnerable to DoS attacks. Yan [15] argues that although the SDN controller itself is vulnerable to DDoS attacks, it provides us with an unexpected opportunity to mitigate DDoS attacks in cloud computing environments at a certain level. Several solutions have been proposed to mitigate SDN DoS attacks. For example, Kotani [16] proposed a packet-in message filtering mechanism to protect the SDN control plane. The packet-in filtering mechanism can initially record the values of packet header fields before sending packet-in messages and then filter out packets that have the same values as the recorded ones. Of course, if the DoS attacker deliberately sends new packets that have values that are different from the recorded ones, the packet-in filtering mechanism will be completely ineffective. Mousavi [17] introduced an early detection method for DDoS attacks on the SDN controller based on the entropy variation of the destination IP address. He assumed that the destination IP addresses of normal flows should be nearly evenly distributed and the destination IP addresses of malicious flows are always destined to several targets. However, it is not difficult for DoS attackers to generate a large amount of evenly distributed new traffic flows to overload the SDN controller. An OpenFlow agent was introduced in a content-oriented networking architecture (CONA) [18] to mitigate DDoS attacks. However, the time required for rate limiting to be implemented and the attack to be halted is uncertain. By analyzing the preceding works, we realize that a more systematic approach is required to address these types of attacks. In the following section, we will describe our design.

3. Design

3.1 System Architecture

MinDoS is proposed to defend the SDN controller against DoS attacks. It includes two functional modules: the *simplified DoS detection module* and the *priority manager module*. The conceptual architecture is shown in Fig. 2. The work process of MinDoS mainly includes the following three steps.

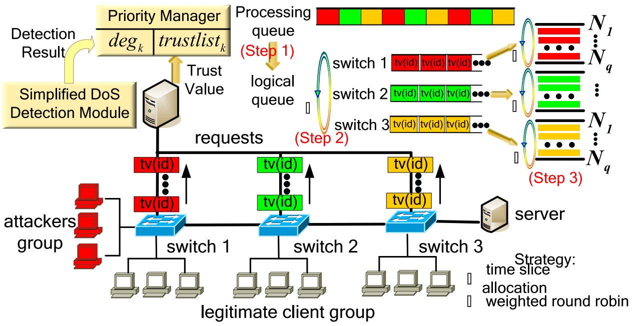


Fig. 2 The architecture and work process of MinDoS.

- Step 1. Divide the single processing queue at the controller into multiple logical queues, each of which corresponds to one SDN switch.
- Step 2. Change the first-come-first-serve (FCFS) processing mechanism and instead serve the logical queues based on a time slice allocation strategy.
- Step 3. Design a priority management strategy to improve the quality of service (QoS) for normal users.

The details of the three steps are described in the remainder of this section.

3.2 Time Slice Allocation Strategy

Normally, the flow requests of each switch are all placed in one static physical queue and processed by the FCFS mechanism. However, the controller resources are more likely to be fully consumed by one static queue in this approach. Therefore, we must make changes to this mechanism. Specifically, MinDoS initially places the flow requests into the logical queue of the corresponding switch and then schedules the processing of these flow requests based on the time slice allocation strategy. In this method, the controller can create an isolated allocation of resources for each switch.

The implementation of the time slice allocation strategy is based on the existing DoS detection module [5], [7]. The simplified DoS detection module uses the fuzzy synthetic evaluation decision-making model [7] and a novel 6-tuple feature vector [5] to detect DoS attacks. This module can calculate a comprehensive judgment score that takes a value between 0 and 1, where the score is 0 if there is no DoS attack and 1 if there is a severe DoS attack. After quantifying the scores for each switch, MinDoS applies different time slice allocation strategies to different switches to preliminary reduce the impact of DoS attacks. We use the following equation to describe the time slice allocation strategy:

$$FP_k = \begin{cases} \left\lfloor \dfrac{\overline{Deg_k}}{\sum\limits_{i=1}^{N} \overline{Deg_i}} * \mu \right\rfloor & 1 \le k < N,\ \ \overline{Deg_k} = 1 - Deg_k \\ \mu - \sum\limits_{i=1}^{N-1} FP_i & k = N \end{cases} \tag{1}$$

where the request processing capacity of the controller is μ, the degree of attack on k_{th} switch is Deg_k, and FP_k is the processing capacity allocated to switch k. Equation (1) allocates fewer resources to a switch that has a greater probability of being attacked. In this way, the system can preliminarily defend against DoS attacks on the SDN controller.

3.3 Priority Management Strategy

When launching DoS attacks, both attackers' requests and normal users' requests from the same switch are placed in the logical queue of the same corresponding switch in the above steps. If these packets have no priority, most normal requests will be dropped since the logical queue of the corresponding switch is mostly occupied by attackers' malicious requests. However, if the requests from normal users are given a higher priority, their QoS can be greatly improved. Therefore, in this step, our goal is to protect normal users from DDoS attacks that are launched by attackers who connect to the same switch as the normal users.

To provide different QoS levels, the priority manager module must first rank users based on their trust values. The pseudo code of trust value management is shown in **Algorithm** 1. The priority manager maintains a trust list $trustlist_k$ for each switch to store the trust value of each user in its subnet. When a new user (IP) connects to the network for the first time, the priority manager will initialize its trust value and insert <IP, Trust Value> into the current corresponding list (lines: 1-9). The trust list is updated in real time to precisely reflect the trust information of the network users (lines: 10-20). Specifically, if a user has no signs of attacks when using SDN resources, their trust value will increase. Otherwise, if a user decreases the performance of the controller in handling other requests, their trust value will decrease. As described in Algorithm 1, we first add an entry for user *s* into the trust list and set its initial trust value to 1 (lines: 5-6). In this time slot, if the total request from user *s* is higher than the abnormal threshold, their trust value will decrease (lines: 10-11). If the total request from user *s* is less than the abnormal threshold, their trust value will increase (line: 12). It is also worth noting that the abnormal threshold for each user is computed based on the statistics of the user. In addition, if user *s* does not appear in this time slot, we can infer that the user tends to be inactive, and its trust value needs to be reduced as well (lines: 17-19). Above all, the trust value management algorithm can dynamically adjust the priority of users by managing the trust values so as to mitigate SDN DoS attacks.

Algorithm 1. Trust Value Management

```
1: In each time slot $t$
2: for Each switch $i$ do
3:   for Each request $r$ from user $s$ ($ip$) arriving at priority manager do
4:     if The $ip$ of user $s$ is not in the trust list $\overrightarrow{tl_i}$ then
5:       Insert an entry for $s$ into $\overrightarrow{tl_i}$ =< ID $id$, IP $ip$, TrustValue $tv$ >
6:       Associate $id$ with $ip$ and initialize its trust value $tv(id) = 1$
7:     else
8:       $id \leftarrow$ index of $(ip, tl_i)$
9:     end if
10:    if The total request from this $ip > \theta_i$ then
11:      $tv(id) = \lambda tv(id) - 1$
12:    else
13:      $tv(id) = \lambda tv(id) + 1$
14:    end if
15:  end for
16:  for Each user $s$ in trust list $tl_i$ do
17:    if The $ip$ of user $s$ does not appear in this time slot $t$ then
18:      $id \leftarrow$ index of $(ip, tl_i)$
19:      $tv(id) = \lambda tv(id)$
20:    end if
21:  end for
22: end for
```

After the trust value management process, each packet is labeled with the user's trust value. Next, these trust values are used to design an algorithm to manage priority queues. The pseudo code of priority queue management is shown in **Algorithm** 2. When packets arrive at the priority manager, the priority manager then appends these packets to the corresponding priority queues according to their trust labels:

$$N_r = \left\lceil \frac{tv(id) - \boldsymbol{min}\left\{tv \text{ in } \overrightarrow{tl_i}\right\}}{\boldsymbol{max}\left\{tv \text{ in } \overrightarrow{tl_i}\right\} - \boldsymbol{min}\left\{tv \text{ in } \overrightarrow{tl_i}\right\}} * N_q \right\rceil \tag{2}$$

where $\boldsymbol{min}\left\{tv \text{ in } \overrightarrow{tl_i}\right\}$ and $\boldsymbol{max}\left\{tv \text{ in } \overrightarrow{tl_i}\right\}$ represent the minimum and maximum trust values in trust list $\overrightarrow{tl_i}$ respectively, and N_r is the buffer index. In addition, to keep the total length of the buffers within the necessary bounds, the priority manager must periodically remove the lowest priority packets.(lines: 3-11)

After placing packets into the corresponding priority queues, the priority manager processes them based on the weighted round-robin strategy. In this process, the number of packets processed from a queue is proportional to the weight of the queue, which is defined based on the queue length and priority level (line: 13). Therefore, when the controller allocates processing resources for flow requests to the logical queue of the switch using Eq. (1), requests with higher priority are processed faster than those with lower priority (line: 15). We can use this priority management strategy to provide different QoS levels.

Algorithm 2. Priority Queue Management

```
In each time slot $t$
for Each switch $i$ do
  for Each request $r$ from user $s$ ($ip$) arriving at buffers do
    $id \leftarrow$ index of $(ip, tl_i)$
    Buffer index of $r$: $N_r = \left\lceil \frac{tv(id) - \boldsymbol{min}\{tv \text{ in } \vec{tl_i}\}}{\boldsymbol{max}\{tv \text{ in } \vec{tl_i}\} - \boldsymbol{min}\{tv \text{ in } \vec{tl_i}\}} * N_q \right\rceil$
    if $\sum_{n=1}^{N_q} L_n < L_{max}$ then
      Append r to $N_r^{th}$ queue
    else
      Drops the tail request from the lowest priority non-empty queue
      Append $r$ to $N_r^{th}$ queue
  end for
  for $n = 1$ to $N_q$ do
    Calculate the weight of each queue: $\omega_n = \left\lceil \frac{L_n}{L_{max}} \delta^{n-1} \right\rceil$
  end for
  In each round the number of the requests processed from $n^{th}$ queue
    $FP_i * \frac{\omega_n}{\sum_{n=1}^{N_q} \omega_n}$
end for
```

3.4 Fairness and Starvation Analysis

In this subsection, we describe how fairness and the starvation of requests are handled. In our approach, we first split requests into multiple logical queues, each of which corresponds to a requested SDN switch. Next, the time slice for processing each queue is allocated based on the inverse of the degree of attacks. For example, if there are three switches in the SDN network ($FP = 24$) and the degree of attack on each switch is ($Deg_1 = 0.2$, $Deg_2 = 0.4$, $Deg_3 = 0.8$), the processing capacity assigned by the controller to each switch is: ($FP_1 = 12$, $FP_2 = 9$, $FP_3 = 3$). Therefore, a grater probability that the switch is attacked results in fewer resources being allocated to the switch. Finally, requests in a queue are prioritized by the trust values of the requesters. We assume that there are m priority queues in each switch's logical queue. The weight of each priority queue is $\omega_1, \omega_2, \ldots, \omega_m$, and ω_{max} and ω_{div} denote the maximum and maximum common divisions, respectively. Before processing these queues, the controller initializes a parameter ω to ω_{max}. In each round, all the priority queues are traversed by the controller. If the weight of queue ω_i is greater than or equal to ω ($\omega_i \geq \omega$), the controller processes one packet from the queue. After a round, the controller updates the value of ω to $\omega - \omega_i$. If the value of ω is less than 0 ($\omega < 0$), the controller updates it to ω_{max} again. For example, we assume that there are 3 priority queues (q_1, q_2, q_3) in switch 1 ($FP_1 = 12$), and the weight of each priority queue is ($\omega_1 = 3, \omega_2 = 2, \omega_3 = 1$). Then, the controller scheduling order is (q_1, q_2, q_3, q_1, q_2, q_1, q_1, q_2, q_3, q_1, q_2, q_1). This method not only improves the fairness but also avoids starvation.

4. Performance Evaluation

We implemented MinDoS and a fuzzy synthetic evaluation decision-making model (FSEDM) by modifying the Floodlight controller, and evaluated them in both software and hardware environments.

4.1 Software Environment Setups

The topology used for the software experiment is shown in Fig. 2. We used four servers in the experiment. Each server was equipped with two Intel(R) Xeon(R) x5690 3.47 GHz CPUs with 48 GB of RAM and CentOS 7. The 1st, 2nd and 3rd server used virtual machines to separately run clients (including attackers). The Floodlight controller and virtual switches were implemented on the 4th server. To overload the controller in our experiment, we set $\alpha > \mu = 60$ [reqs/s], where α [reqs/s] is the rate of the DoS attack flow and μ is the flow request processing ability of the controller. In particular, we set α/μ between 1.5 (mild attack) and 8 (severe attack). In the simulation, the legitimate users under each switch were modeled as 'random'. Thus, the legitimate users sent requests at random rates (the value was between $\mu/6$ and $2\mu/3$). We set the capacity of the controller processing queue to $Q_{max} = 3000$ [reqs], the logical queue capacity of each switch to $L_{max} = 200$ [reqs], the rate threshold to $\theta_i = 25$ [reqs/s], the decay factor of the trust value to $\lambda = 0.9$, the default number of priority buffer queues to $N_q = 5$, and the weight scale factor to $\delta = 2$. During the entire simulation process, i.e., 50 seconds, we measured four parameters: the TCP connection setup failure rate R_f, the successful connection setup delay D_s, the delay of prioritizing requests and the processing of priority queues. To evaluate the performance of MinDoS, we compared it with the other two schemes, FCFS and FSEDM. The results under different DoS attack intensities of $\alpha = 1.5\,\mu, 2\,\mu, 4\,\mu, 8\,\mu$ are shown in Fig. 3, 4 and 5.

4.1.1 Defense Effect Analysis

The results in Fig. 3 show that the connection failure rate (R_f) when applying FCFS increases sharply as the attack intensity increases. In this respect, MinDoS and FSEDM perform better. For example, assuming that $\alpha = 8\,\mu$, the connection failure rate using MinDoS or FSEDM is much lower than that found when applying FCFS, and the successful connection delay (D_s) is also low. The connection failure rate and the successful connection delay are higher when using FCFS because the FCFS approach does not include a mechanism to detect attack requests. The results in Fig. 3 also show that MinDoS outperforms FSEDM, regardless of whether it is measured by D_s or R_f. A more intense attack leads to a greater advantage. These results show that MinDoS can effectively protect the controller and normal

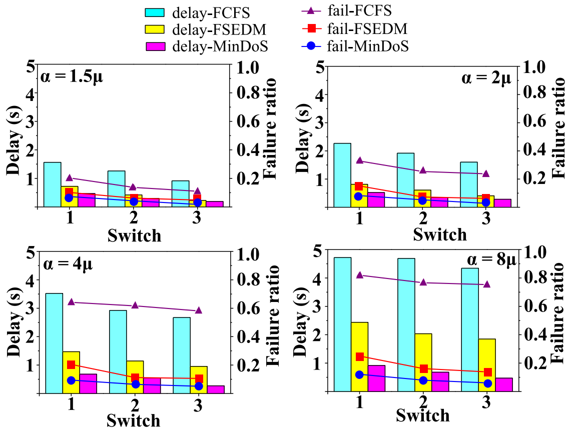


Fig. 3 Comparison of R_f and D_s under different DoS attack intensities.

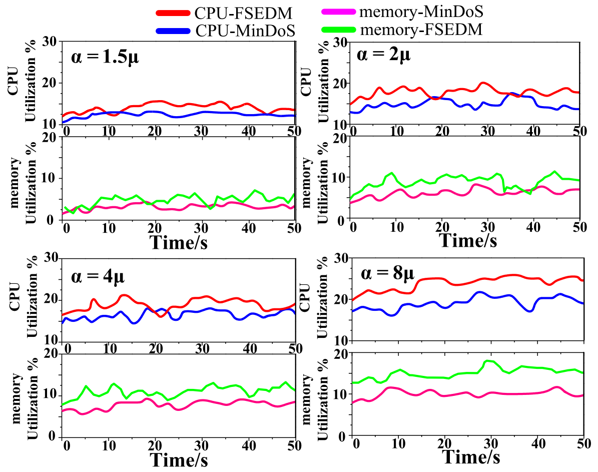


Fig. 5 CPU/memory utilization under different DoS attack intensities.

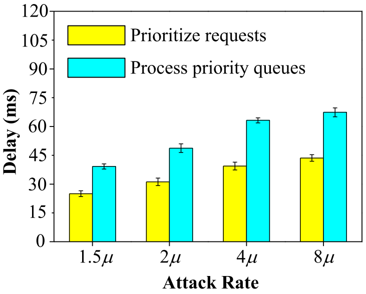


Fig. 4 Overhead of prioritizing requests and processing priority queues.

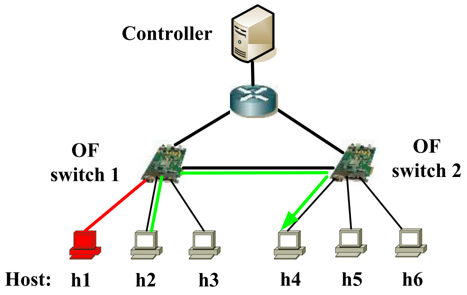


Fig. 6 Hardware environment.

users from DoS attacks. Because MinDoS makes full use of the advantages of FSEDM, the time slice allocation strategy, and the priority management strategy, better attack defense effects are ensured.

4.1.2 Overhead Analysis

In this section, we present our evaluation of the overhead required for MinDoS. To include the costs of scanning the queue to prioritize and organize requests, we measured the average overhead required to prioritize requests and process priority queues in the simulation. The results for different DoS attack intensities are shown in Fig. 4. The data in Fig. 4 show that the overhead of prioritizing requests and processing priority queues increases slightly with increasing attack intensity because the flooding attacks inevitably add extra load to the priority manager. After the attack rate reaches a certain level, the overhead tends to stabilize. For the worst case ($\alpha = 8\mu$, severe attack), the overhead of prioritizing requests is less than 43.6 ms, and the overhead of scanning requests is less than 67.4 ms. This overhead is acceptable for our system.

Next, we compared and analyzed the overhead for MinDoS with that used for FSEDM. We also compared the CPU and memory utilization to determine the computational overhead of the system. The evaluation results are presented in Fig. 5. The data show that the overall utilization (either CPU or memory utilization) of MinDoS is relatively low, which indicates that MinDoS is highly scalable and able to provide security services for more network devices. In contrast, the overhead of the FSEDM is high. The FSEDM must make a comprehensive judgment according to multiple factors and then take the appropriate protective measures. This process involves many complex calculations. Therefore, the evaluation results are not surprising. As attack intensity increases, the overall utilization of MinDoS and FSEDM also slightly rise. The CPU utilization is below 30%, and the memory utilization is below 20%, which is reasonable given the small scale of the experiment.

4.2 Hardware Environment

We set up a test bed with three physical switches and six users, as shown in Fig. 6. Here, OF switches 1 and 2 are FPGA-based OpenFlow switches. The switch uses four 1 Gb/s ports to forward packet and one 1 Gb/s port to connect the controller. The other experimental parameter settings are the same as described in Sect. 4.1.

We let h1 be the attacking host and let h2 ping h4. The round-trip times (RTTs) between h2 and h4 are shown in Fig. 7. The data in Fig. 7 show that using the FCFS mechanism, the RTTs between h2 and h4 become very large, with an attack rate up of to 1000 packets/s. Additionally, the RTTs between h2 and h4 remain small when using FSEDM or MinDoS. This result reflects the effectiveness of MinDoS

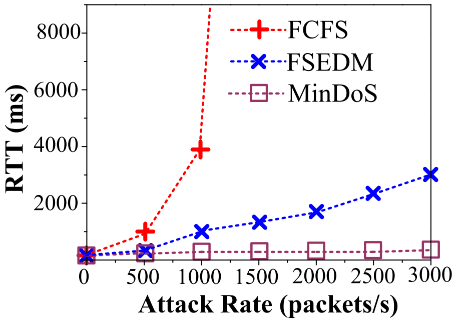


Fig. 7 RTTs (round-trip times) when h2 ping h4.

and FSEDM in defending against TCP flooding attacks. As the attack rate increases, the RTTs between h2 and h4 begin to sharply increase when using FSEDM; however, the RTTs between h2 and h4 remain nearly constant when using MinDoS. Furthermore, FSEDM fails to process the flow requests when the attack rate is above 1500 packets/s. This demonstrates that the defensive effect of MinDoS is better than that of FSEDM under TCP flooding attacks.

5. Conclusions

In this paper, we propose a priority-based SDN safe-guard architecture, MinDoS, to mitigate DoS attacks. Specifically, the time slice allocation strategy and priority management strategy greatly enhance system security. The experimental results demonstrate that MinDoS ensures better protection of the controller and adds only minor overhead to the entire SDN infrastructure. In future work, we plan to implement MinDoS on larger experimental topologies to improve our results.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No.060802, 61521003), the National key Research and Development Program of China (No.2016YFB0800100, 2016YFB0800101), the National Natural Science Foundation for Young Scientists (No.61602509). the Science and Technology Project of Henan Province (172102210615), and the Emerging Direction Fostering Fund of Information Engineering University (2016610708). We gratefully acknowledge anonymous reviewers who read drafts and made many helpful suggestions.

References

[1] G. Yao, J. Bi, and L. Guo, "On the cascading failures of multi-controllers in Software Defined Networks," 2013 21st IEEE International Conference on Network Protocols, pp.1–2, 2013.

[2] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, and H.J. Chao, "Improving the performance of load balancing in software-defined networks through load variance-based synchronization," Computer Networks, vol.68, pp.95–109, 2014.

[3] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks," Proc. 2013 ACM Sigsac Conference on Computer & Communications Security, pp.413–424, 2013.

[4] H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks," Proc. 2015 International Conference on Dependable Systems and Networks, pp.239–250, 2015.

[5] T. Wang and H. Chen, "SGuard: A Lightweight SDN Safe-Guard Architecture for DoS Attacks," China Communications, vol.14, no.6, pp.113–125, 2017.

[6] M.M. Alsulaiman, A.N. Alyahya, R.A. Alkharboush, and N.S. Alghafis, "Intrusion Detection System Using Self-Organizing Maps," Proc. 3rd International Conference on Network and System Security, pp.397–402, 2009.

[7] Q. Yan and Q. Gong, "Detection of DDoS attacks against wireless SDN controllers based on the Fuzzy synthetic evaluation decision-making model," Ad Hoc & Sensor Wireless Networks, vol.3, no.1, pp.275–299, 2016.

[8] S. Shin, P. Porras, and V. Yegneswaran, "FRESCO: Modular composable security services for SoftwareDefined networks," Proc. 2013 Network & Distributed Security Symposium, pp.1–16, 2013.

[9] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," Proc. first workshop on Hot topics in software defined networks, pp.121–126, 2012.

[10] P. Porras, S. Cheung, M. Fong, K. Skinner, and V. Yegneswaran, "Securing the Software Defined Network Control Layer," Proc. 2015 Network and Distributed System Security Symposium, pp.1–15, 2015.

[11] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B.B. Kang, "Rosemary: A robust, secure, and high-performance network operating system," Proc. 2014 ACM SIGSAC conference on computer and communications security, pp.78–89, 2014.

[12] 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.

[13] 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.

[14] K. Benton, L.J. Camp, and C. Small, "OpenFlow vulnerability assessment," Proc. ACM SIGCOMM Workshop on Hot Topics in Software Defined NETWORKING, pp.151–152, 2013.

[15] Q. Yan and F.R. Yu, "Distributed denial of service attacks in software-defined networking with cloud computing," IEEE Communications Magazine, vol.53, no.4, pp.52–59, 2015.

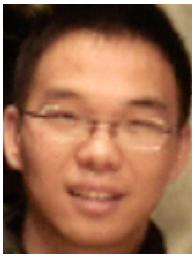
[16] D. Kotani and Y. Okabe, "A packet-in message filtering mechanism for protection of control plane in openflow networks," Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pp.29–40, 2014.

[17] S.M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against SDN controllers," Proc. International Conference on Computing, Networking and Communications, pp.77–81, 2015.

[18] J. Su, H. Choi, and W. Yoon, "Implementation of Content-oriented Networking Architecture (CONA): A focus on DDoS countermeasure," Proc. 2010 European NetFPGA Developers Workshop, pp.1–6, 2010.

Tao Wang is currently a Ph.D. candidate at National Digital Switching System Engineering and Technological Research Center (NDSC, Zhengzhou, China). His research interests include software-defined networking and security.

Hongchang Chen is currently professor at National Digital Switching System Engineering and Technological Research Center. His research interests include future network communication and future network architecture.



Chao Qi is currently a Ph.D. candidate at National Digital Switching System Engineering and Technological Research Center (NDSC, Zhengzhou, China). His research interests include software-defined networking and security.