FlowRanger: A Request Prioritizing Algorithm for Controller DoS Attacks in Software Defined Networks

Lei Wei
School of Computer Engineering
Nanyang Technological University
Singapore

Carol Fung
Dept. of Computer Science
Virginia Commonwealth University
US

***Abstract*—Software Defined Networking (SDN) introduces a new communication network management paradigm and has gained much attention from academia and industry. However, the centralized nature of SDN is a potential vulnerability to the system since attackers may launch denial of services (DoS) attacks against the controller. Existing solutions limit requests rate to the controller by dropping overflowed requests, but they also drop legitimate requests to the controller. To address this problem, we propose FlowRanger, a buffer prioritizing solution for controllers to handle routing requests based on their likelihood to be attacking requests, which derives the trust values of the requesting sources. Based on their trust values, FlowRanger classifies routing requests into multiple buffer queues with different priorities. Thus, attacking requests are served with a lower priority than regular requests. Our simulation results demonstrates that FlowRanger can significantly enhance the request serving rate of regular users under DoS attacks against the controller. To the best of our knowledge, our work is the first solution to battle against controller DoS attacks on the controller side.**

I. Introduction

Software Defined Networks (SDN) introduces a new communication network management paradigm and has gained plethoras attention from academia and industry due to its fast growing potential in the market. One primary goal of SDN is to allow a network controller, called the *control plane*, to overlook and manage the entire network by configuring routing mechanisms for underlying switches. The switches, also called the *data plane*, are solely responsible for data forwarding according to their forwarding tables. Routing computation and network management are handled by the controller. The entries of forwarding tables on switches are determined and managed by the controller through a secured communication protocol between switches and the controller.

More specifically, enterprises adopt OpenFlow [6] as communication protocol for secure and efficient communication between switches and the controller. As specified in the OpenFlow protocol [1], switches must forward a *packet-in* request to the controller if they receive a new packet (an indication of new flow) that they do not have any matching entry in the flow table to forward the packet. The controller receives the packet-in request and computes a routing path for the new flow before sending *packet-out* packets to notify corresponding switches to update their forwarding tables, which is an indication of the establishment of a new flow path. Subsequent packets belonging to the same flow will be forwarded accordingly by switches.

However, the centralized paradigm of SDN is a potential vulnerability to the system since attackers may launch denial of services (DoS) attacks or distributed denial of service (DDoS) attacks against the controller. For example, an attacker may create a large number of new flows in a short period of time [7], intending to overwhelm the controller and cause network failure for legitimate users. The current OpenFlow mechanism provides a solution where switches use short buffer size and drop excessive requests before sending them to the controller, hereby avoid overloading the controller. However, this solution causes a high dropping rate of legitimate flow requests under attack since all requests are handled equally. Its first-come-first-serve mechanism does not grant legitimate flows any advantage.

To address the above problem, we propose FlowRanger[1], a flow prioritizing algorithm to enhance the quality of service for data flows from legitimate regular clients. In this solution, we first use a ranking algorithm to identify regular normal users based on their past requests to the controller. We then prioritize the execution of requests by using multiple priority buffers on the controller's side. The packets in a higher priority buffer are processed with higher priority than packets in lower priority buffers. Our simulation demonstrates that FlowRanger gives significant advantage to requests from regular users compared to requests from attack users under controller DoS attacks. Therefore, it can effectively reduce the impact from attacks and maintain the normal operation of an enterprise network. To the best of our knowledge, this paper is the first proposed controller-side solution to battle against controller DoS attacks in SDN.

The rest of this paper is organized as follows: In section 2, we give a literature overview of SDN and (D)DoS attacks to SDN. Section 3 describes the (D)DoS attacks to controllers in SDN. Our solution named FlowRanger is presented in Section 4. We demonstrate our simulation results of FlowRanger in Section 5. Finally Section 6 concludes this paper and provides future work remarks.

II. Related Work

The emergence of software defined networks provides a new paradigm of solving traditional network routing problem,

[1]FlowRanger - Flow arRanger

by introduing separate layers for routing and data forwarding [2]. At the mean time, SDN security and dependability has become an open research field for researchers. Recently many works have been published addressing the security issues in SDN. For example, FRESCO [8] is an OpenFlow security application development framework to facilitate the development and deployment of security applications in SDN. VeriFlow [4] is proposed as an extra layer sitting in between control plane layer and data plane layer, which is to check for network-wide invariant violations dynamically as each forwarding rule is inserted, modified or deleted. Kreutz, et.al. [5] provides us an insight of possible threat vectors of SDN and their respective solutions. Among them, the first threat they address is forged or faked traffic flows to SDN, where attackers create packets with spoofed address to the SDN network, which possibly causes denial of service attacks to switches or controllers.

The most crucial part of the SDN is the controller, and it is also a single point of failure in the entire SDN ecosystem. This vulnerability can be exploited by attackers. A series of intentionally crafted data packets arriving from multiple attackers which does not have matching flow entries in flow tables can accumulate at the same time and exhaust the resources of the switch. These intentionally crafted data packets [7], [3] can arrive to the controller as *packet-in* queries for the missed flow entries and will also exhaust the control plane resources, resulting in a denial of service attack to the controller, also called data-to-control plan saturation attack [10]. Several solutions have been proposed to address the data-to-control plane saturation attack, such as Avant-Guard [9] and OF-Guard [10].

Avant-Guard [9] is an extension to the open flow data plane called connection migration (CM). The CM responds to handshake packets if no matching flow entries found. Only when connection is established, then the packet is sent to controller to ask for routing path. The purpose of CM is to fight against DoS attack based on IP spoofing, by effectively reducing the amount of data to control plane under DoS attacks. However, Avant-Guard is not effective on controller DoS attacks using real IPs or through proxy, since TCP connections can be established. In addition, Avant-Guard is limited to TCP-based DoS attacks. Another weakness of Avant-Guard is its implementation on switches. All switches need to be Avant-Guard equipped, otherwise the entire network is still vulnerable.

Another proposed solution is OF-Guard [10], which use a intermediate server called data plane cache, to filter probable attacking packets before sending them to controller. However, this approach is limited to known attackers, such as attackers in blacklists. For DDoS attacks by nodes not in the black list, OF-Guard is not effective. The same as Avant-Guard, OF-Guard requires all switches to be equipped with the OF-Guard extension. Otherwise the system is vulnerable.

In our work, we propose FlowRanger, a controller-side solution to battle against DDoS attacks to controllers. Our solution does not require any modification on the data plane side. Therefore, it is easier to deploy compared to the previous solutions.

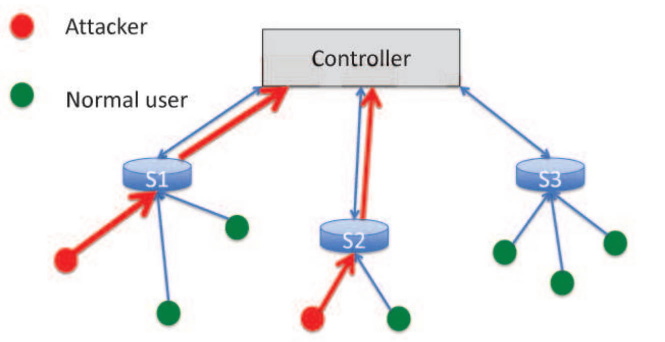


Fig. 1. An illustration of controller DDoS attack

III. Problem Description

In a SDN network, when a switch receives a packet but have no corresponding entries in the flow table indicating where to forward the packet, it sends the packet to a controller through a *packet in* request for routing path computation. The controller receives requests from the switches, computes the "appropriate" routing path, and sends *packet out* packets to update the flow tables of involved switches. By default, the requests are queued in the controller's buffer upon arrival and are served by *first-come-first-serve* manner. If the buffer is full, new requests will be dropped and will never be served.

Due to the contralized nature of SDN, controllers can be the attacking target of DoS/DDoS. Attackers can be malicious hosts in the network or malicious switches. For example, malicious hosts/switches can initialize a large number of new flows simultaneously inside the network. Since by definition all new flows should be sent to the controller to find routing paths, a large number of simultaneous *packet-in* requests can overload the controller and cause a large number of packets dropping out of the controller's buffer. Therefore new flows requests from benign users may not be served since their routing requests are dropped by the controller together with other attacking requests since they are not handled differently. In a SDN network, there commonly exisit some regular users who use the network services on a regular basis. For example, they can be corporations using clound services for their daily bisiness operation. Those regular users should have priority and be protected from being influenced by DoS attacks. The goal of our work is to find a solution to minimize the impact of controller DoS attack to legit regular users.

In SDN 1.4 standard, the controller can configure a *flow meter* parameter on switches to limit the request sending rate from each switch. However, it does not solve the problem since it only allows most legible packets to be dropped at a earlier phase. It is also not clear how to set an optimal sending rate limit for each switch in order to have optimized performance. Other solutions, such as Avant-Guard [9] and OF-Guard [10], propose to add an additional packet filtering function on the switch-side. However, those switch-side solutions impose the following constraints: 1) All switches have to implement the solution otherwise the left-out becomes the vulnerability of the entire system. Sometimes having all switches implement the switch-side solution may not be practical. 2) It can be infeasible or impossible to coordinate

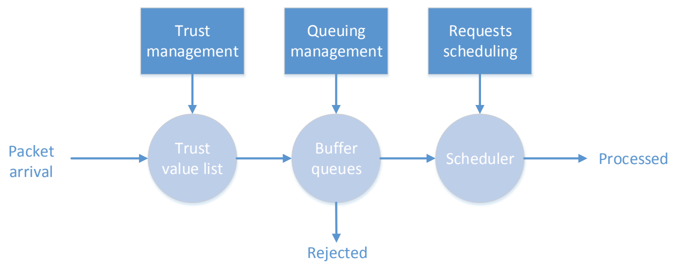


Fig. 2. Design of FlowRanger

the filtering thresholds on all switches to optimize the overall system performance due to computation or communication overhead constraint. 3) If a switch itself is compromised and be malicious, then the switch-side solution is rendered useless.

In this paper we propose a novel solution where we handle the controller DoS attack problem through job prioritization on the controller side. Our goal is to protect legit regular users from heavily influenced from controller DoS attack. In our solution, the packet-in requests to the controller are not served through a first-come-first-serve manner (which is by default), but through a job prioritization process based on the attacking likelihood of the source. For example, as illustrated in Figure 1, both attackers' routing requests and legit users' routing requests are sent to the controller. When requests are not prioritized, most legit requests will be dropped since the controller job buffer is full. If the requests from legit users have higher priority, then their service quality can be significantly improved. Through the job prioritization process, the service quality of normal and legit hosts will not be overly influenced even the controller is flooded by a large number of malicious requests, hence improving the overall performance of the SDN network.

It is also worth noting that our solution is not a replacement of Avant-Guard or OF-Guard since our solution is implemented on the controller side. Instead our solution can work seamlessly with those two solutions to further improve the robustness of SDN against controller DoS attacks. To the best of our knowledge, FlowRanger is the first controller-side solution to battle against controller DoS attacks.

IV. The Design of FlowRanger

TABLE I
List of Notations

Notation	Meaning
$\vec{tv}, tv_{max}, tv_{min}$	Trust value list, max, and min value
s_i	A SDN network user with index i
α	Forgetting factor, $\alpha \in (0, 1)$
θ_i	Abnormal threshold for user i
T_{min}	Threshold trust for malicious users
P_i	Total request from user s_i
N_q	Number of priority buffer queues
L_{max}	Total buffer length of N_q buffer queues
L_j	Length of the j^{th} buffer queue
β	Weight scale factor for buffer queues

To minimize the impact from controller DoS attacks (or DDoS attacks) to legit regular users in SDN networks, we propose *FlowRanger*, a controller-side job prioritization mechanism by providing differential services to routing requests from different sources based on their likelihood of being attackers. FlowRanger consists of three components: *trust management*, *queuing management*, and *requests scheduling*. When a new packet-in request arrives at the controller, the trust management component computes the trust value of the request sender. After that the queuing managent component maps the request to a corresponding priority queue based on the trust value of the sender. Finally, the request scheduling component determins the processing order of all requests in different priority queues. Figure 2 shows the work flow of FlowRanger. The details of the three components are described in the rest of this section. Table I list all notations used in this paper.

Algorithm 1 Trust management

```
1:
2: In each time slot $t$
3: for Each request $r$ arriving at controller do
4:   if The sender $s$ is not in trust list $\vec{tv}$ then
5:     Add an entry for sender $s$ in $\vec{tv}$
6:     $tv(s) = 1$
7:   else
8:     $i \leftarrow$indexof$(s, \vec{tv})$
9:     if The total request from sender $i$ in this time slot
       $P_i > \theta_i$ then
10:      Reject request $r$
11:      Break
12:    else
13:      if Controller not under attack then
14:        $tv_i = \alpha \cdot tv_i + 1$
15:      else
16:        $tv_i = \alpha \cdot tv_i - 1$
17:      end if
18:    end if
19:    if $tv_i < T_{min}$ then
20:      Blacklist sender $i$ and notice switches
21:    end if
22:  end if
23: end for
24:
25: In the end of each time slot $t$
26: for Each sender $j$ in trust list do
27:   if sender $j$ does not appear in time slot $t$ then
28:     $tv_j = \alpha \cdot tv_j$
29:   end if
30:   Update $\theta_j$ based on the total requests in time slot $t$
31: end for
```

A. Trust Management

To provide service differenciation to requests, it is important to know the how likely a request is from an attacker. The trust management component aims at tracking the trust values (likelihood of being benign) of request users in SDN network. The users we refer here is the device which uses the SDN resource. Therefore we can use IP addresses to identify users in SDN. In FlowRanger, the controller maintains a list of frequent users and their corresponding trust values. The trust values are updated in realtime to reflect the frequency of usage from the users in normal condition. More specifically, if a user uses the SDN resource during normal condition (no sign of attacks), then its trust increases. Otherwise if it appears during the time SDN is under attack, then its trust value decreases. To keep the list fresh, the system peridically removes the least trusted users from the list to make room for new users. The length of the trust value list depends on the number of regular users of the system and the resource on the controller that can be used to maintain the list. As specified in Algorithm 1, during each time slot, if a request is from a new user and there is room in the tracking list, a new trust tracking entry is created for the new user with a default initial trust value. Otherwise if it is an existing user in the list, then the trust value is updated accordingly. If the trust value of a user is lower that a threshold T_{min}, the controller considers the user to be an attacker and can blacklist the user and notify switches to drop any further packets from the user. Note that a request will be dropped immediately if the number of requests received in the time slot exceeds the abnormal threshold of the user. The abnormal threshold for each user is computeted based on the statistics of the user.

Algorithm 2 Queuing Management

```
1:
2: for Each request $r$ arriving at buffers do
3:     Buffer index of $r$: $i = \left\lceil \frac{tv_i - tv_{min}}{tv_{max} - tv_{min}} * N_q \right\rceil$
4:     if $\sum_{j=1}^{N_q} L_j < L_{max}$ then
5:         Append $r$ to queue $i^{th}$ queue
6:         $L_i = L_i + 1$
7:     else if $i$==1 then
8:         Reject request $r$
9:     else
10:        for $j = 1$ to $i - 1$ do
11:            if $L_j > 0$ then
12:                Reject a request from the tail of $j^{th}$ queue
13:                $L_j = L_j - 1$
14:                Append $r$ to queue $i^{th}$ queue
15:                $L_i = L_i + 1$
16:                Break
17:            end if
18:        end for
19:    end if
20: end for
```

B. Queuing Management

After trust comutation, each request is labled with the trust value of the sender. The priority of a request is deterimed by its trust label. To be specific, if a request is from a highly trusted user then it will be served in a higher priority. To serve the goal of serving requests with different priorities, FlowRanger proposes SDN controller to maintain multiple (N_q) buffer queues with different priority levels, and their accumulative length is up to L_{max}. Requests are appended to corresponding buffer queues based on their trust labels. As shown in Algorithm 2, the buffer queue number is computed based on the trust value of the sender. If the buffer queues are full, FlowRanger drops the tail reuest from the lowest priority non-empty queue before inserting the new request, therefore keeping the total length of buffers under the bound.

Algorithm 3 Request scheduling

```
1:
2: Calculate weights in round robin for each buffer queue
3: for $j = 1$ to $N_q$ do
4:     $w_j = \left\lceil \frac{L_j}{\max(L_1, 1)} \beta^{(j-1)} \right\rceil$
5: end for
6:
7: while true do
8:     for $j = 1\ to\ N_q$ do
9:         if $w_j < L_j$ then
10:            Serve $w_j$ requests in $j^{th}$ queue
11:            $L_j = L_j - w_j$
12:        else
13:            Serve $L_j$ requests in $j^{th}$ queue
14:            $L_j = 0$
15:        end if
16:    end for
17: end while
```

C. Request Scheduling

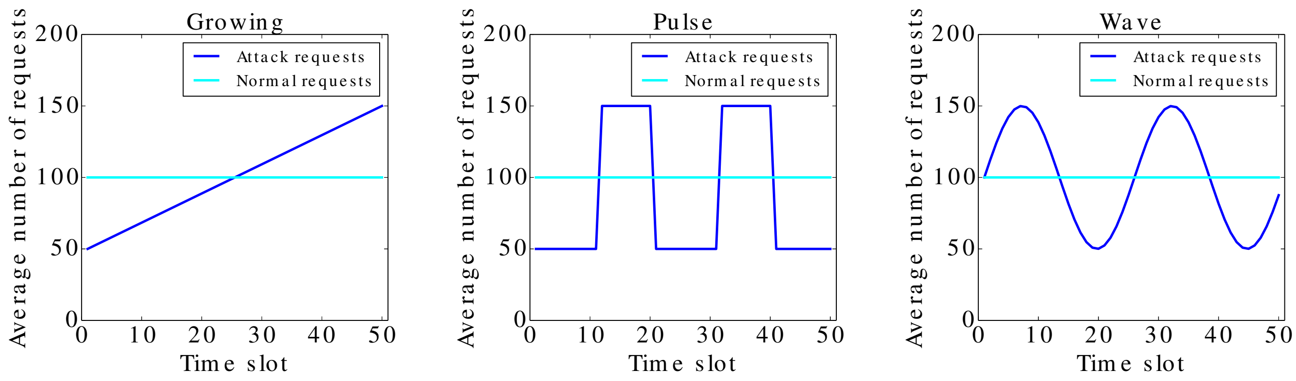
After appending a request to a corresponding beffer queue, the request is waiting to be served. FlowRanger uses a weighted round robin strategy to process requests in the list of queues with different priority levels. As shown in Algorithm 3, the weight of each buffer queue is computed based on the length and priority level of the queue. In each round, the number of the requests processed from a queue is equal to the weight of the queue. Therefore requests in the queues with higher priority are processed faster than those in queues with lower priority. This way FlowRanger serves the goal of providing differential services on requests from senders with different likelihood of being attackers.

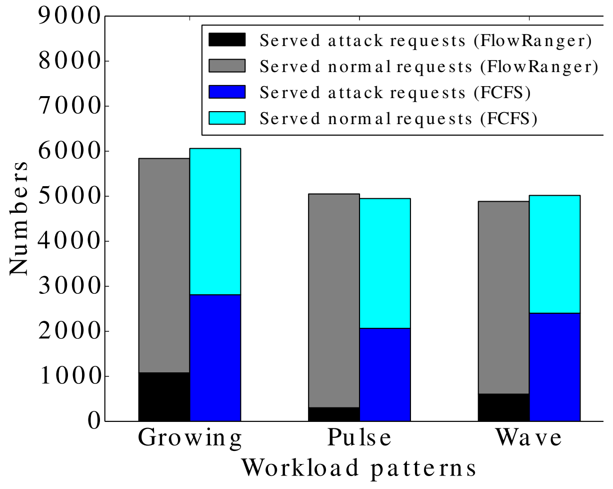
V. Experiments

In this section, we use a simulation approach to evaluate the performance of FlowRanger. We first discuss about the simulation setup, then our evaluation results on the comparison of the impact of attacks with and witout FlowRanger under different paramter settings.

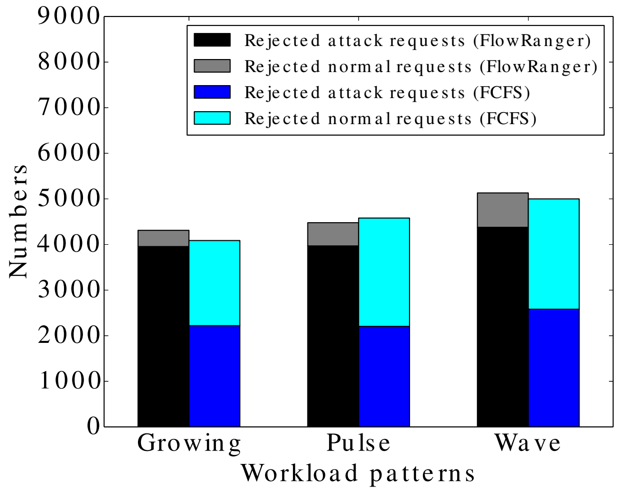
A. Experiment setups

Simulator: We develop a discrete event simulator to evaluate the performance of FlowRanger. We consider a SDN network with one controller and three switches. the network controller is able to process 100 requests per time slot (service rate is 0.01). There are in total 100 users using the network. The default maximum buffer length of controller is set to 200. The simulation lasts for 150 time slots. We use $\alpha = 0.9$ as the forgetting factor in the trust value updating in trust management. The default number of queues in the priority buffers is $N_q = 3$ and the priority factor in request scheduling increase with a ratio of $\beta = 2$. We are interested to observe the ratio of served normal request to total number of normal requests. For the purpose of comparison, we also simulate the





(a) Served requests



(b) Rejected requests

Fig. 4. Sensitivity studies number of priority queues, amounts of attack requests and forgetting factor.

first-come-first-serve (FCFS) scheduling method with only one buffer queue.

Attack injection: To evaluate the effectiveness of FlowRanger against DoS attacks, we synthesize three different DoS attacking workloads patterns with different dynamic variations of router requests, namely Growing, Pulse and Wave as shown in Fig. 3. Each workload pattern lasts for 50 time slot. By default, the requests towards SDN controller in each time slot follows a Poisson distribution. Specifically, the normal requests are all Poisson arrivals with average value of 100 for each time slot for three patterns. We then inject attack requests with dynamic arrival rates as shown in Fig. 3 where the peak loads are set to 150 and lowest loads are 50.

B. Experiment results

Overall results: Fig. 4 shows the overall results of both our proposed FlowRanger and basic FCFS method for three workload patterns. We can see that FlowRanger is able to serve much more normal requests than FCFS (as in Fig. 4(a)) under three different attack patterns. Since FCFS have no mechanism to detect attack requests and normal requests, the number of served attack requests and normal requests by FCFS are determined by their workload. Similar results are shown on the ratio of rejected attack requests and normal requests in Fig. 4(b). The percentage of rejected attack requests by FlowRanger is much higher than FCFS. These results demonstrate that FlowRanger is capable of detecting and rejecting most of the DDoS attacks towards SDN controller effectively.

The impacts of the volumn of attack requests: Fig. 5 illustrate the results in each time slot for the ratio of served normal requests to the total normal requests. We can see that, in general FlowRanger serves much (43%) more normal requests than FCFS. The service rate of normal requests is not heavily impacted when the total requests arrival rate exceeds the service rate. As a contract the service rate of normal requests drops by large when using FCFS. We also notice that that the ratio of served normal requests by SDN controller drops significantly when there are burstive attack requests (e.g., peak loads in pulse and wave workload patterns). The reason is that FlowRanger does not react immediately to the burstive situation. As the time goes, the controller will lower the attacker's trust value and reject them.

The impacts of the number of buffer queues: In this experiment we study the impact of the number of priority buffer queues in FlowRanger. We can see from Fig. 6(a) that, the effectiveness of defending DDoS attacks improves with more number of buffer queues. However, the improvement is not significant after two queues. This is because with queues, FlowRanger can well separate the attack flows and normal flows. Thus, two queues can be sufficient to provide an efficient detection when the difference between normal requests and attack requests is apparent.

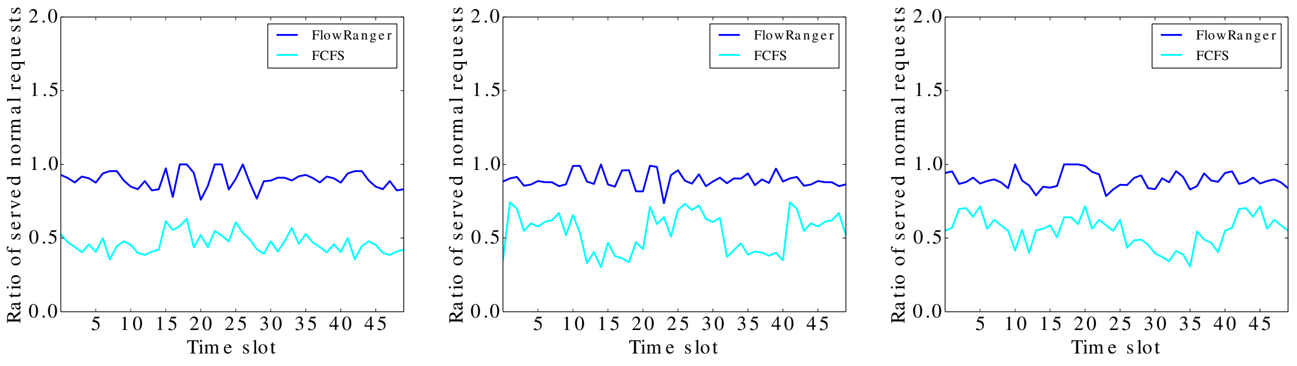


Fig. 5. Ratio of served normal requests in each time slot for three workload patterns

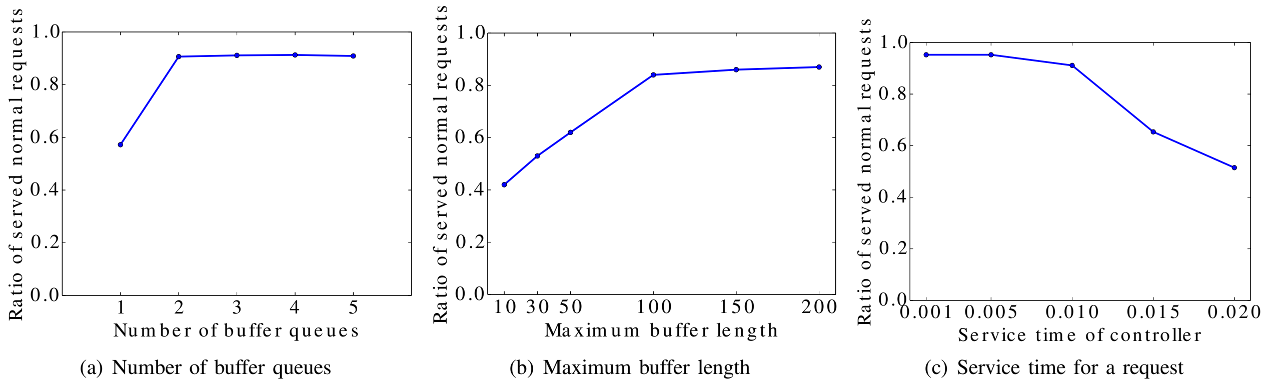


Fig. 6. Sensitivity studies number of priority queues, amounts of attack requests and forgetting factor.

The impacts of the maximum buffer length: In this experiment we study the impact of maximum buffer length on controller to the performance of FlowRanger. Fig. 6(b) shows the results by using different maximum buffer length in FlowRanger. We can see that the ratio of served normal requests increase with the buffer length and then become stable after the buffer length reaches 100. This is because the request processing time of SDN controller is set to 100 in the experiment. If the maximum buffer length is less than the service rate of controller, many request are rejected.

The impacts of the service time of controller: In this experiment we study the impact of service time of controller on DoS defending. From Fig. 6(c), we can see that the ratio of served requests decreases as the service time gets longer. The reason is that longer service time means the controller is slower in processing requests, which leads to high rejection rates for both attacking and normal requests.

VI. CONCLUSION

In this paper we propose FlowRanger, a controller side buffer priority solution that protect SDN from data-to-controler plane flooding attack. FlowRanger uses a trust-based mechanism to evaluate the likelihood of packet-in requests are from attacking sources and prioritize them into multiple buffer queues with different priorities. Requests from trusted sources are arranged in high priority queues and are served faster than requests in low priority queues. Our simulation results shows that FlowRanger can effectively reduce improve the request serving rate from trusted normal users and the impact from DDoS attack is significantly reduced compared to a simple FCFS mechanism. More specifically, our results show that FlowRanger is able to serve 43% more regular requests than first-come-first-serve policy.

As of our future work, we plan to further improve the performance of FlowRanger by improving request scheduling algorithm to optimize the performance of FlowRanger. We also plan to implement FlowRanger on a real controller in our deployment phase.

REFERENCES

[1] Open flow switch specifications v.1.4. https://www.opennetworking.org/images/stories/downloads/sdnresources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[2] Sdn architecture overview. https://www.opennetworking.org/images/stories/downloads/sdnresources/technical-reports/SDN-architecture-overview-1.0.pdf.

[3] K. Benton, L. J. Camp, and C. Small. Openflow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 151–152. ACM, 2013.

[4] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey. Veriflow: Verifying network-wide invariants in real time. *ACM SIGCOMM Computer Communication Review*, 42(4):467–472, 2012.

[5] D. Kreutz, F. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 55–60. ACM, 2013.

[6] 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*, 38(2):69–74, 2008.

[7] 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*, pages 165–166. ACM, 2013.

[8] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security services for software-defined networks. In *NDSS*, 2013.

[9] 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 SIGSAC conference on Computer & communications security*, pages 413–424. ACM, 2013.

[10] H. Wang, L. Xu, and G. Gu. Of-guard: A dos attack prevention extension in software-defined networks. In *Open Networking Summit 2014, Poster Session*. USENIX.