Computer Standards & Interfaces 54 (2017) 293–304

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Challenges of and solution to the control load of stateful firewall in software defined networks

Thuy Vinh Tran, Heejune Ahn*

Seoul National University of Science and Technology, Electrical & Information Engineering department, 232 Gongneung-ro, Nowon-gu, Seoul, South Korea

ARTICLE INFO

Keywords:
Software defined networking
SDN security
stateful firewall
DoS attack

ABSTRACT

Whereas SDN (Software Defined Networks) provides the opportunity for the flexibility of network configuration, the introduction of controller systems raises new issues about developing firewall system design, such as controller attack, rule setup, and communication overhead for control messages. Especially, the delay and overload for dynamic control of stateful firewall are serious bottlenecks. This paper examines the current challenges and their origins, and presents a comprehensive solution to the key operational steps: topology-based selective filtering rules for setup and maintenance stage, three-layer rule structure for in-switch flow tables, and controller attack protection based on adaptive host connection tracking with multiple hashing queues named FlowTracker algorithm. The experiment results using multiple OVS switches and virtual hosts in GENI testbed demonstrate FlowTracker succeeds in monitoring all network connections and completely profiling host normal routine with acceptable latency increment (170 ms). Moreover, by utilizing multiple request queues, the proposed DoS attack detection algorithm reduce the response time to DoS 5 to 20 times less than using a single queue.

1. Introduction

Software Defined Networking (SDN) [1] introduces the possibilities of faster evolution, hardware independence and centralized-control network. These objectives are realized by decoupling network control functions from packet forwarding. This decoupling feature of SDN impacts firewall system design, and enables the firewall logic and policies of firewalls to be implemented in the controller side whereas the switches execute the switching and filtering operation according to the configured rules. However, the separation of control and data plane causes an increase in the communication bandwidth and controller load to set up the firewall rules.

In the case of a 'stateless' firewall, the controller can pre-install the firewall rules in the flow table of each switch, so that the forwarding switch can perform stateless packet filtering without interfering the controller on runtime. On the other hand, a 'stateful' firewall requires far more interaction between the control plane and data plane for managing connections. OpenFlow SDN switches utilize the 'flow' concept [27]. The OpenFlow-compatible switches compare the packet header with the matching fields defined in the flow to perform the associated action with the flow. For the stateful firewall function, the controllers have to track connections at the switches and transfer the flow table configuration to the switches. As a result, communication interaction between the switches results in controller communication overhead and the increment of connection latency. Moreover, this connection setup latency and controller overhead can be exploited by malicious users, i.e., hackers, introducing a new type of DoS attack toward the controller [11].

In this paper, we consider the increased communication and load in the perspective of designing stateful firewall system in SDN. Recently IETF proposed SDN based security architecture including firewall function [31]. Our system firewall architecture complies with one proposed therein. In both our solution and solution [31], switches report to the firewall logic in controller about the unknown flows, but beside examining the packet header and content as in [31], we also use the connection tracking function of FlowTracker to verify whether the packet belong to a valid and ongoing connection or not. Regarding the DoS/DDoS detection, SDN security architecture [31] use flows' inter-arrival pattern while our solution use host connection routines to detect possible attack. We believe our solution and [31] are compatible and can run alongside to offer additional security measures. Through our literature reviews, SDN Stateful firewalls were studied in only few works [6,7,23,24,32]. The overall designs of stateful firewall are proposed but in only simplified network scenario are considered. Especially, the challenges in reducing controller load for tracking and potential risks of controller attack are largely unexplored. To the best of

* Corresponding author.
E-mail addresses: tranvinhthuy@seoultech.ac.kr (T.V. Tran), heejune@snut.ac.kr (H. Ahn).

http://dx.doi.org/10.1016/j.csi.2017.01.012
Received 16 April 2016; Received in revised form 20 January 2017; Accepted 26 January 2017
Available online 27 January 2017

our knowledge, there are no researches addressing these problems and proposing countermeasures to improve the performance and resilience of SDN stateful firewall.

The delay and overload for dynamic control of stateful firewall are inherent and can be serious bottlenecks for stateful firewall design. It is not likable that a single mechanism can solve this problem easily, and a comprehensive set of solutions is proposed. For firewall setup and maintaining stages, we use topology data for selectively and efficiently installing and updating firewall rules in appropriate switches. To differentiate between approved traffics, unidentified traffics and forwarding traffics, we introduce three-layer structure for in-switch flow table. To deal with the potential of DoS attack toward controller, we exploit the connection statistics collected by stateful firewall for early detecting of suspicious host behaviors. Finally, we deal with the DoS mitigation issues by utilizing multiple queues with different priorities for controller incoming requests to rapidly discard the whole group of harmful packets and ensure the requests from trusted hosts are processed with preference.

The future trends for network security suggest the next generation firewalls to be more dynamic in policy enforcement and have the ability to protect the network from the inside [22,33,35]. SDN technologies are anticipated to provide new securities capabilities for the ordinary network devices like switches. With the centralized firewall logic and distributed firewall operation, the firewall policies could be implemented throughout the network at every ingress switch ports and on every network links [34]. The potential for an SDN based next generation firewall are huge, however an practical model and de facto standard is not yet introduced. Also from [34] we can acknowledge that even among the big players in networking industry and SDN focused research organization like Cisco's Application Centric Infrastructure, Open Virtual Switch, Floodlight and VMware NSX have yet to introduce a SDN stateful firewall solution. We strongly believe that more research efforts are much needed in the development and fully realization the potential of SDN stateful firewall.

To ensure the feasibility of a SDN stateful firewall, we look back into the OpenFlow specification [14], which is the core part of the whole SDN standardization. The flow entry matching fields defined in the OpenFlow specifications are tightly involved with packet filtering process of switches, which in turn has impact on the operation of SDN firewall. From one of the earliest version of OpenFlow (0.8.0), only layer-2 packet header fields are defined, enable for merely stateless layer-2 filtering. Toward version 0.8.9, IP addresses and layer-4 protocol fields are added enabling the switch to aware of connection characteristics between end to end hosts. Moreover, with the proposal of the latest OpenFlow version 1.5.0, the inclusion of TCP flag matching fields even allow stateful firewall to track the state of TCP connection by the switch itself. It can be seen that the development of SDN stateful firewall are in line with the development of SDN standard. SDN standard evolvement naturally enable more and more possibility-possibilities for a practical and efficient SDN stateful firewall.

With the acknowledgement for the ever-growing research interest and industry attention for SDN stateful firewall, this research focused on the most distinct issues of stateful firewall in SDN environment. A set of novel solutions are introduced to tackle these issues, moreover we tried to evaluate our solution in a as closed as possible network setup. What we want to achieve is a practical SDN stateful firewall solution that addresses the unresolved issues and take advantages of the rapidly evolved SDN standard to maximize the potential of SDN stateful firewall.

The structure of this paper is as follows. Section 2 discusses the new challenges of stateful SDN firewall design. After that, Section 3 presents our stateful firewall solution. Section 4 presents the simulation results of the proposed stateful firewall system using GENI test bed and an extended discussion on the results. The last section concludes the results and proposes future works for SDN firewall design.

2. Background: challenges in sdn firewall design

The decoupling feature of packet-switching and control is the key feature of SDN technology. The decoupling feature impacts operating firewall function and network operations. The main issues in operating SDN stateful firewall are discussed as follows:

- Configuration issues: According to Brent S. [19], TCAM is the most expensive component of the switches. Because switch TCAM table has finite capacity of flow entries [20], the flow rules need to be carefully and concisely installed in the switches for firewall operation to avoid duplicate and redundant flows in multiple switches. Moreover, the firewall rules should enable switches to differentiate unidentified flows from approved ones to eliminate multiple reports for the same connection.
- Connection tracking issues: There are prior works examining SDN firewall utilizing Access Control List (ACL) [2–4]. However, ACL only allows for stateless packet filtering. To achieve stateful firewall, the connection state tracking need to be taken into account. The protocols in transport layer are various, but can be divided into two categories: connection-oriented and connectionless. Connection-oriented protocols such as TCP have their connection states clearly defined. In contrast, connectionless protocols like UDP has no procedure of connection establishment or tear-down, which means the state can only be tracked in pseudo manner. The difference in characteristics requires adaptive tracking from stateful firewall. Moreover, the inevitable additional latency for tracking should be considered. Minimizing this delay is imperative to warrant the feasibility of stateful firewall solution.
- Security issues: With the current stateful firewall solutions, the controller is requested to process every single new connection. This can be a potential security hole that hackers exploit for DoS attack. The attacker host can rapidly emit a large number of packets with different footprints so that the stateful firewall forwards all of them to the controller. Not only the controller processing resource is drained, but also the control plane-data plane channel can be compromised. The SDN Controller attack is discussed in SDN security surveys [10–13]. Scott-Hayward et al. [10] categorized 10/13 SDN securities risks affect control plane. The survey [11] listed compromised controller attack as one of the new issues that SDN network is exposed to. Sakir et al. [12] raised concerns about DoS attack as a potential threat to both the controller and the individual forwarding devices. In [13] Kreutz listed the controller attack as the most dangerous attack vector in SDN network because of the risk of compromising network operation. As a network security entity, stateful should be equipped to protect the controller against dangerous DoS attack. The goal of firewall design should be to timely detect DoS attack symptoms and rapidly mitigate the negative effects of DoS attack upon the controller.

  Several researchers recently studied SDN stateful firewalls. Jake C. et al. [6] introduced an OpenFlow-based prototype of SDN-oriented stateful firewalls. They consider a simple scenario and evaluate the latency caused by Stateful firewall when the number of firewall rules controller keeps increasing. Similarly, Karamjeet et al. [24] presented an implementation of stateful firewall and performed an experiment of TCP and ICMP packets filtering on a simple topology with two hosts and two switches. Changhoon et al. [7] also examined the feasibility of stateful firewall in SDN. They proposed the conceptual design and implementation of SDN stateful firewall on top of Floodlight [25] controller framework. Their stateful firewall solution means a complete application layer tracking, with an additional NIC on the controller side.

  Our observations on the limitations of the prior researches [6,7,24] on SDN firewall design can be summarized as follows.
- While the structure and filter process of SDN stateful firewall are described in detail, the setup phase including firewall rule placement

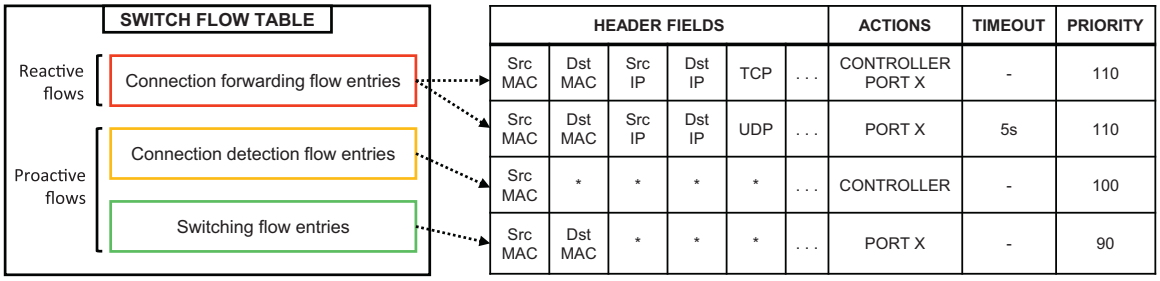

Fig. 1. Three-layer structure for switch flow table.

and management in switches is largely unexplored. Only conceptual, simple scenario with just one or two switches and two hosts are considered. Network control protocol and algorithm should be studied considering multiple switches and various numbers of hosts, because the complexity and performance of network control usually depends upon the size of networks (scalability issue).

– In the previous works [6] and [24], there is no consideration for solving the current issues of SDN stateful firewall, especially controller communication and processing overhead. The study [7] expresses the concern about impacts of network performance with the operation of stateful firewall. However, this work presents neither the detail of the problems or propose any countermeasure.
– The previous study [6] evaluates the scaling behavior of average latency when the number of firewall rules increases. A single latency figure of stateful firewall is presented in [7]. Similarly, study [24] measures the latency of stateful firewall to process the requests. Whereas the result shows insight into the overall performance, the controller performance variation factor in the case of different numbers of simultaneous connections is not discussed.

With the main goal of analyzing and undertaking the challenges of stateful firewall operating in the new SDN network paradigm, we try to consider the more realistic network case which helps to address each problem with the corresponding solution. To resolve the complication of installing firewall rules for multiple switches and hosts, the topology based approach our previous work [5] is used for selectively setting up the firewall rules with the topology data maintained by controller. To reduce the packet classification load, the switches process separately the new traffic requests and authorized connection traffics. The novel three-layer switch flow table structure is proposed to differentiate the incoming traffic in switches and avoid the duplicate new connection requests to controller. Additionally, the hybrid of 'proactive' and 'reactive' flow rules [9] are used to minimize the controller effort in manipulating switch flow tables while the control flows are efficiently updated for ongoing connection traffics. Furthermore to take into account the difference of transport layer characteristics in connection-oriented protocols (TCP) and connectionless protocols (UDP), 'double-action flow rules' are used for tracking stateful protocols (e.g. TCP) and predefined timeout flow rules are used for monitoring stateless protocols (e.g. UDP).

Also in this work, we address the possibility of control plane attack comes from heavy involvement of controller in the tracking operation. An algorithm is developed to detect and mitigate the DoS controller attack. Connection statistics collected by stateful firewall support individual host connection profiling to detect the suspicious behaviors. Multiple message queues with MAC address hashing are incorporated in the request of processing operation of the stateful firewall to rapidly discard the attacking packets and regain controller processing and memory resource. To the best of our knowledge, this is one of the first SDN controller side solutions to deal with DoS attack [28].

3. Flowtracker – a comprehensive sdn stateful firewall solution

3.1. Connection tracking module

Compared to stateless networking, stateful networking requires far more interactions between the controller and underlying switches [8]. The key operation of stateful firewall is to monitor all the connections in the network. This is one of the main challenges of designing an efficient stateful firewall solution to SDN. Our goal of FlowTracker is to timely and accurately track the connection status, which means tracking all connection states in real time with reasonable controller communication overhead.

FlowTracker tracks the protocols at Transport layer (OSI layer 4) to be fully compatible with OpenFlow protocol coverage and ready to be deployed on any SDN system. Two representative Internet transport protocols, TCP and UDP are considered. Because TCP and UDP behave very differently in establishing and maintaining their connection, FlowTracker tracks TCP and UDP in different ways so that this stateful firewall solution can adaptively track wherever the connection states are clearly defined (TCP) or vice versa (UDP).

Each flow entry represents one unique connection between two hosts in the network. The header fields including all representative information of the connection including source and destination MAC address, source and destination IP, layer 4 protocol (TCP or UDP). The 'Action' field of the entry defines the action that the switch will perform on the packets that matches this flow entry. The possible actions are forwarding to specified port, forwarding to all port (flood), reporting to controller or dropping the packet. The timeout fields can be specified so that after this period if no traffic coming through the switch match this flow entry then the entry will be removed. This is a predictive method to remove redundant flow entries.

Fig. 1 illustrates our three-layer switch flow table setup. The structure of flow table in each switch is formed by three layers. Each layer is defined the group of flow entries with a similar properties and states and purpose and handled differently (in priority) by the switch. The first layer consists of the flow rules already approved by FlowTracker controller, and processed with the highest priority. The middle layer detects a new connection for establishing a flow entry, and the lowest layer is to switch rules of forwarding the traffics in intermediate switches. The incoming traffic to the switches will be compared for matching in order from first to third flow entry layers.

Notably, each switch only maintains the connection forwarding flow entries and connection detection flow entries related to the directly connected to the switches itself, while the traffics coming from other hosts will match with the switching flow entries. This structure works well in separating between three kinds of flows: identified and approved flows from directed hosts, new initiating flows, and immediate traffic flows coming from other hosts. Each switch only reports to the controller about the connections initiated from the hosts directly connected while the traffics from other switches will only match the switching flows and be forwarded without reporting. Naturally, the network will be divided to groups and each group will be managed and

monitored by one switch. The divide-and-conquer strategy helps create a concise flow table by eliminating redundant flows in multiple switches to handle a single traffic flow.

Connections detection flow entries and switching flow entries are 'proactive' flows [9], which means they are preset in the switches by controller. On the other hands, connection forwarding flow entries are 'reactive' flow entries [9]. They are dynamically added to the switch flow table when a new connection is reported and approved, and are removed when the termination of connection is detected by the controller. The hybrid usage of 'proactive' flows and 'reactive' flow minimizes unnecessary flow table altering in switches while allowing in time table update for controlling the new connections.

The tracking function focuses on tracking the two connection states: initiation and termination. For tracking connection initiation, only the packet does not match with any existing monitored connection is forwarded to the controller, FlowTracker makes sure that only first packet initiating the connection is reported. Therefore, the communication overhead is kept to minimum for tracking new connection. For connection termination tracking, different methods are used to deal with TCP and UDP connections. In case of UDP, timeout parameters are used to detect expired traffic, so no overhead for tracking UDP connection ending with the cost of the timeout delay. With TCP connections, the switch has to send the packets continuously to controller for examining TCP FIN flag, so the communication overhead is existing in this case.

We consider a simple connection tracking scenario of two hosts connected to two switches and a controller in Fig. 2. The details of how the FlowTracker sets up and populates the three-layer flow table of the switches will be elaborated in each step.

(0) During the setup phase, FlowTracker controller populates the flow table of each switch with connection detection flow entries and switching flow entries. As our previous work [5], FlowTracker uses the topology data for selectively installing these flow rules. The topology data is recorded as a data table where each entry will contain the host MAC address and the switch that the host directly connects to. After this phase, the flow table of switch 1 is populated as in Table 1.

(1) Host A sends the first packet to request a connection to host B.
(2) This packet matches the connection detection flow entry in switch 1 and is forwarded to the controller.
(3) The stateful firewall module in controller receives 'Packet_in' event. The following series of actions will be performed to examine the packet:

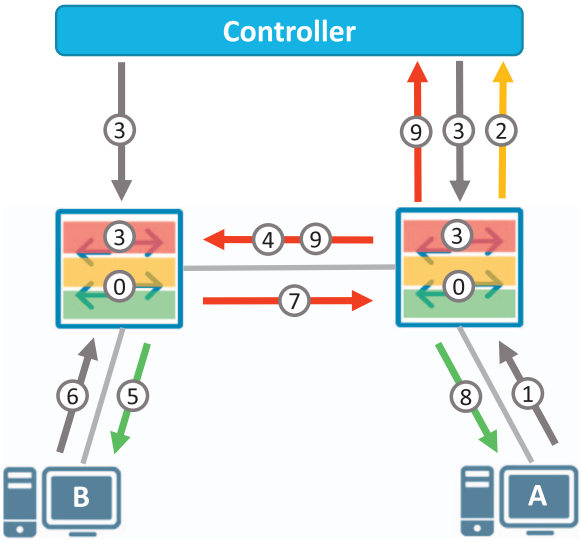

Fig. 2. Tracking process of FlowTracker.

Table 1
Illustrated switch flow table after FlowTracker setup phase.

Header fields				Actions	Timeout	Priority
mac A	*	*	...	CONTROLLER	–	100
mac A	mac B	*	...	PORT 1	–	90
mac B	mac A	*	...	PORT 2	–	90

– The packet header is compared with the existing connections information maintained by FlowTracker.
– If not matched, the packet is classified as a new connection initiating packet.
– The packet header is matched against the firewall rules
– If not matched with the violation rule, the connection is approved:

1. Connection information including all header fields data of the packet will be added to ongoing connection list of FlowTracker. Each connection is saved as an object in a connection list of FlowTracker. The object will have the attributes corresponding to the number of header fields from layer 2 to layer 4 of the connection packet. All the header fields are included to make sure the uniqueness of connection data.
2. By looking up the topology data table with the sender and receiver MAC addresses, FlowTracker can find out the two switches immediately next the sender and receiver for connection
3. FlowTracker sends 'Flow_mod' OF packet to both switches to add connection forwarding entries to the flow table. The flow entries will have matching fields specified from all the packet header fields. However, to adaptively track connection-oriented protocols (TCP) or connectionless protocol (UDP), there will be different configurations:

- For TCP, the connection termination can be detected via FIN TCP flag. So double actions will be specified so that FlowTracker can continuously track the TCP packets FIN flag, while the packets are sent to the destination at the same time
- For UDP, there is no definite sign to detect connection tear down. In the flow entries the timeout will be set. Thus, after a specific time, no UDP packets are sent then the flow entries will be removed and connection ended.

(4) After the connection forwarding entry is installed in switch 1, the packet matches this flow rule and is forwarded to switch 2
(5) In switch 2, the incoming packet will match switching flow entries and is forwarded to host B.
(6) The replied packet is sent from host B
(7) In switch 2, the replying packet matches the connection forwarding flow entries and is sent back to switch 1
(8) The reply packet matches switching flows on switch 1 and is redirected to host A.
(9) Now data can be exchanged between two hosts. In case of TCP connections, a copy of sending packet from host A will be sent to controller for detecting connection tear down.
(10) When the connection is ended:

- For TCP connections, FlowTracker detects the connection ended when the FIN TCP flag is set as 'True'. After that, FlowTracker will send 'Flow_mod' OF messages to both switches to remove the connection forwarding flow entries
- For UDP, after the connection is ended, connection forwarding flow entries will be removed automatically due to timeout. Controller will learn this when 'Flow_removed' event is sent from switches
- For both case, after detecting connection termination, FlowTracker will remove the connection information in its on-going connection list.

Our proposed FlowTracker utilizes new ideas to improve the performance of stateful firewall:

- To the best of our knowledge, this is the first time topology data is used in a SDN firewall to boost the efficiency of the SDN solution. The topology map is used extensively in order to concise the switch flow table. By selectively installing the control flow entries in each switch, the setup phase for firewall is simplified and the number of flow entries switch for maintenance decreases. Additionally, topology data helps FlowTracker to examine the two switches managing the new connection rapidly.
- The novel three-layer model for the switch flow table are proposed toward well-categorized incoming traffics to the switches. With hybrid approach of using reactive flows and proactive flows together, FlowTracker can deploy essential firewall control flows to the switch beforehand while the flows continue to deal with ongoing connections are updated dynamically.
- We explore two methods to continuously tracking the connection efficiently. After connection detection, FlowTracker can keep tracking the connection without any additional delay. For connection-oriented protocols, the dual-action flows allow tracking and forwarding at the same time. For connectionless protocols, timeout is a heuristic solution to minimize controller interaction and well-suit the continuous characteristic of these protocols.

3.2. FlowTracker DoS attack detection/mitigation module

FlowTracker constantly maintains the connection information. These connection statistics can be used to estimate the usual behavior of each host. Thus, in case of significant derivation from that routine, such as a sudden increase in new connection request in a short time, the controller can be alerted right away of the potential attack. Moreover, the originator of the attack can also be identified.FlowTracker monitors the peak number of on-going connection for each particular host. From the cumulative peak history in previous time slots, the baseline peak connection of each host is estimated. If the number of connection requests from the host exceeds a certain predefined degree from baseline, then it will be identified as an attacker. To mitigate the damage of controller DoS, FlowTracker maintains multiple message queues of different priorities. The incoming requests from the hosts with history of attacker or showing suspicious behaviors are put into 'warning queue' with lower priority and are processed after all requests in 'normal queue' are handled. DoS detection and mitigation is integrated into existing stateful firewall function of FlowTracker. The modified structure is described in Fig. 3.

From single stateful firewall thread, FlowTracker is separated into three independent threads with different tasks:

- Enqueue thread: putting the incoming request from data plane in an appropriate queue or dropping the requests of the attacker.
- Stateful Firewall thread has three modules:

1. Stateful firewall module: tracking, monitoring and matching connections against firewall policies
2. Statistic updating module: updating peak connection number and comparing with baseline to detect attacker
3. DoS mitigation module: activating the DoS damage mitigation right after the attacker is identified

- Statistic processing thread: being run at the end of each timeslot, updating host reputation, recalculating baseline and updating peak history

Table 2
Algorithm parameters.

Notation	Meaning
N_h	Max length of host peak history
h_{ij}	Peak of ongoing connection of host i at j previous timeslot
p_i	Peak number of ongoing connections in this timeslot for host i
c_i	Current number of ongoing connections for host i
b_i	Mean baseline of ongoing connections for host i
s_i	Suspend cap for host i
w_i	Warning cap for host i
SL	Suspending deviation level
WL	Warning deviation level
B	Global baseline
S	Global absolute suspend cap

The parameters for the DoS detection and mitigation are defined as in Table 2.

The details of three threads are discussed in the next subsections.

3.2.1. En-queue thread

The enqueue thread keeps listening to the 'Packet_in' event to catch the incoming packets from data plane. This thread initiates two request queues: a normal queue and a warning queue. Each queue is actually a hash list of message queue with host MAC address as hash input.

Each host in the network, identified by its MAC address, is assigned one from three reputation based on its behavior.

- 'Attacker' reputation for hosts that violate in this timeslot.
- 'Suspicious' reputation for hosts that violated in the previous timeslot or reaches 'warning cap' in this timeslot.
- 'Normal' reputation for non-violating hosts.

Examining the reputation of the host, the enqueue thread can make the decision to handle:

- Requests from 'attacker' will be dropped.
- Requests from 'suspicious' will be put into Warning queue.
- Request from 'normal' host will be put into Normal queue.

The process of this process is illustrated in Fig. 4.

Two queue lists with host MAC address hashing are used for DoS mitigation:

- MAC address hashing enables FlowTracker to classify the incoming traffics from different hosts. This results in a significant benefit that the controller can discard the whole group of malicious packets from

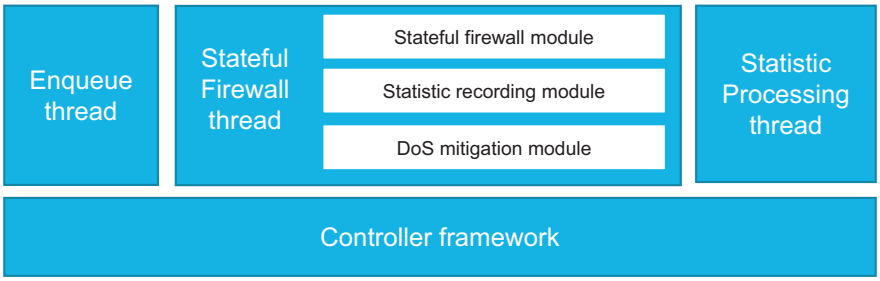

Fig. 3. Overall FlowTracker system.

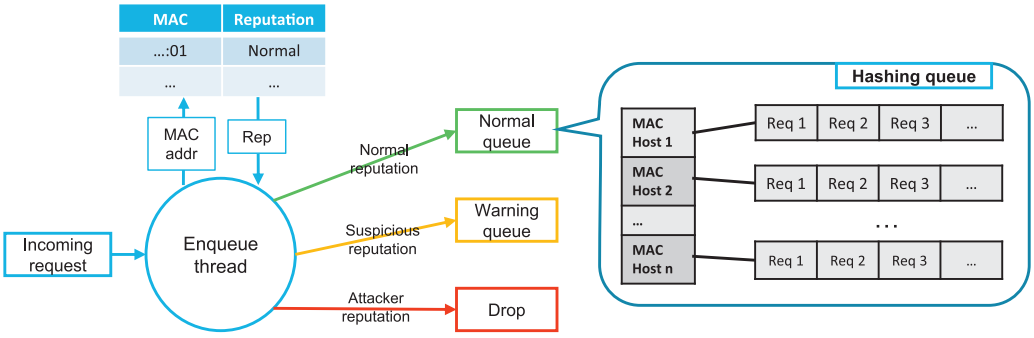

Fig. 4. FlowTracker multiple hashing queues structure.

attackers at once and does not needs to check packet by packet. It is our belief that this practice is effective in protecting the controller processing and memory resource from the large amounts of DoS packets.

- Two message queues with different priorities ensures that the legit requests from clean hosts with no attack history are prioritized and processed first. On the other hand, the requests from suspicious host who has the history of being attacked or currently increasing its number of ongoing connections over the baseline are only handled after the normal queue is completely empty. Without the priority queue mechanism, legit packets on the waiting line would get dropped because of many malicious packets filling up the queue.

3.2.2. Statistic processing thread

The statistics collecting thread is activated at the end of each timeslot to update the parameters including host peak history, host baseline and limits, host reputation and global baseline.

```
STATISTIC PROCESSING THREAD
for each host in topo list:
    if reputation[host_i_MAC] == "attacker"
        reputation[host_i_MAC] = "suspicious"
    else if reputation[host_i_MAC] == "suspicious"
        reputation[host_i_MAC] = "normal"
    if length(h_i[]) == N_h:
        remove the oldest entry from h_i[]
    add p_i to h_i[]
    b_i = avg( sum_{k=1}^{length(h_i)} h_i[k])
    s_i = b_i*SL
    w_i = b_i*WL
B = avg( sum b_i)
```

First Statistic Processing thread updates the reputation for each host:

- The attacker in current timeslot will have "suspicious" reputation in the next timeslot.
- The "suspicious" host with no violation in this timeslot will have its reputation reset to "normal".

As can be seen in Fig. 5, the peak number of ongoing connection for each host in previous N_h timeslot will be about average to produce the baseline b_i of this host in next timeslot.

- Warning cap w_i is the product of baseline b_i and a predefined warning derivation level WL.
- Suspend cap s_i is the product of baseline b_i and a predefined suspend derivation level SL.

When the number of connection requests from host exceeds warning cap or suspend cap, host reputation will be marked as "suspicious" or "attacker" respectively. The global baseline B is about average from the baseline sum of all hosts. The global baseline will be assigned to the newly join to the network host with no connection history.

There is a trickier situation when the attacker deliberately increases the number of peak connections gradually over several timeslots in order to reach a high baseline number eventually. To prevent this we introduce Global absolute suspend cap S applied for all hosts in the system. Regardless of individual suspended limit, when a host reaches this number of peak connections, it will be treated as attacker.

The baseline calculation only takes a summation and average of collected peak connection values so the calculation complexity is $O(n^2)$. However, the number and saved values are small and the Statistic Processing thread runs independently with the main firewall thread so there is no significant performance degradation from statistic calculation.

3.2.3. Stateful firewall thread

This thread bears the main function of stateful firewall described in the previous section. In addition, Stateful firewall also continuously updates and evaluates the peak connection record for each host.

```
STATEFUl FIREWALL THREAD
while (true)
    if normal queue is not empty:
        current_packet = pop a packet from normal queue
    else if warning queue is not empty:
        current_packet = pop a packet from warning queue
    else
        continue
    if packet violates firewall policies:
        continue; // Drop
    else if packet is for initiating new connection:
        c_i+=1
        if c_i > p_i:
            p_i = c_i
        if c_i ≥ S or c_i ≥ s_i:
            reputation[host_i_MAC] = "attacker"
            b_i = B
            s_i = b_i*SL
            w_i = b_i*WL
            drop all further packets from host i in normal
and warning queue
        else if c_i > w_i:
            reputation[host_i_MAC] = "suspicious"
        record new connection
        install forwarding rules
    continue
```

Stateful firewall thread will update the current number of connections c_i of the particular host as soon as the new connection is detected

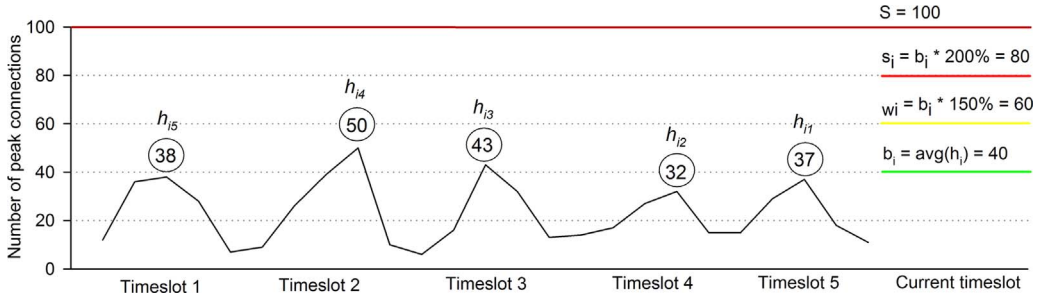

Fig. 5. Connection baseline estimation.

and approved. Peak connection value is also updated accordingly. It is then compared with Global Absolute Suspend cap S, host suspend cap s_i and host warning cap w_i. If p_i exceeds S or s_i, the host is ruled out as attacker. FlowTracker performs the following countermeasures:

– Change host reputation to "attacker" so further incoming packets from the host in this timeslot will be discarded.

– Discard the rest of incoming packets from host i in normal queue and warning queue.

Reset the connection baseline/current peak p_i of host i to global baseline B to reduce the connection cap of host i.

4. Experiment result

The experiment is set up and performed on GENI test bed [15]. The linear topology contains five OVS switches [17], each switch connecting with four XEN VM hosts. Links between switches and hosts have bandwidth of 100Mbps. FlowTracker runs as an extension on top of POX controller framework [16] which supports OpenFlow protocol version 1.0. Hping3 software [26] was used to generate TCP and UDP packets.

We focus on examining two aspects: the effectiveness of DoS detection and mitigation function measured by a number of attack packets processed/rejected and time to discard all malicious packets in controller, the performance of FlowTracker and its additional controller processing overhead including connection latency and controller service rate. DoS mitigation result and service rate are compared with those of single controller request queue as in normal case while latency result is measured against the latency of LearningSwitch module.

4.1. DoS prevention testing

We set up an attack host and a normal host. The attacker and normal host send a number of TCP SYN requests in each timeslot. The number of requests per time slot of normal host is static throughout all timeslots while that of attacker suddenly increases and decreases to simulate bursting TCP SYN flood attack. The attack is in pulse pattern, repeating five normal timeslots, then followed by five attacking timeslots. This attack pattern is shown in Fig. 6 and the parameters for this experiment is listed in Table 3.

4.1.1. Attack detection and prevention

Fig. 7 shows the number of processed packets and rejected packets from attacker host by controller in each timeslot.

It can be observed that in the timeslot when the attacker host follows normal behavior with the number of new connection request under host suspend cap, all the requests are accepted. On the other hand, during the attack wave timeslots, FlowTracker is able to reject all of the requests exceeding the suspending limit. Because SDN firewall design is still in infancy stage, there is yet a de facto solution to be used as a baseline to evaluating how effective FlowTracker is in rejecting attacking packets. However the result in Fig. 7 could indicate the feasibility and practicality of FlowTracker in protecting the controller from harmful incoming requests.

4.1.2. DoS mitigation effect

The DoS mitigation function of FlowTracker utilizes multiple hashing queues with different priorities to rapidly discard malicious packets and reduce processing delay for legit users. To test this feature, we compares the performance of FlowTracker running multiple hashing message queues against FlowTracker with a single message queue. The mitigation time is defined as the duration from the moment DoS is detected until all the malicious packets are discarded from controller queue(s). The mitigation time is recorded for all attack waves.

With multiple hashing queues, FlowTracker only has to check the queues corresponding to MAC address of attacker in normal and warning list and removed all of them at once. In contract, when using a single queue, FlowTracker has to check each individual packet to match the attacker info. The longer these unwanted requests presenting in the queue, the more overwhelming processing and memory resources controller has to spend. Moreover, the high number of these malicious packets could have negative effects upon legit packets such as additional delay and higher chance of getting dropped due to overflow queues. As can be seen in Fig. 8, mitigation times when using multiple hashing queue range only from 0.3 to 0.85 ms. Even when the number of attack requests goes as high as 600, the mitigation time is still less than 1 ms. For the single queue, the mitigate time is 10 to 25 times higher, from 2.5 ms in case of 100 attack requests to 21 ms when the number of attack requests increases to 600. It can be seen that using multiple hashing queues is more effective in DoS mitigation than using single message queue.

In the next evaluation, we focus on the impact of different priority queues on the processing of legit requests in case that DoS happens. The total processing time for requests coming from normal host is requested. The total processing time marks the duration it takes FlowTracker to process all requests from normal host in each timeslot.

From Fig. 9, it can be observed that the processing duration for normal user using one queue is mostly stable and only slightly increases toward the ends align with the increment of total request number. It can be noticed that the incoming packets arrives in chunk and all the requesting packets from one host come at once, followed by those of other hosts. This is different from our presumption of mixing arrival between two hosts and can be explained due to slightly difference in delay from packet generating, link traffic and very fast arrival rate etc. Naturally, one queue FlowTracker itself will process all the requests from one host to another.

On the contrary, using multiple hashing queues with round-robin queue lookup allows FlowTracker to process requests from both hosts in turn. Hence, the processing duration for normal user during the time slots with no attacks is double those of single queue. A repeating pattern from the second timeslot of attacking period to the first timeslot of cool-down period can be observed. The processing durations for normal user greatly decrease and match the duration of single queue. During the second attacking timeslot (6, 16, 26, 36 ,46), the

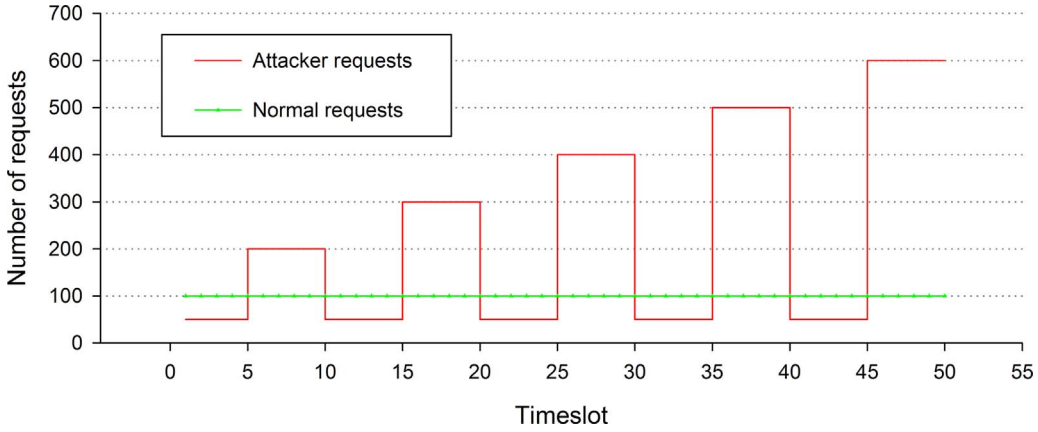

Fig. 6. SYN Flood attack pattern.

Table 3
Experiment parameters.

Parameters	Value
Simulation duration	50 timeslots
Singe attack wave duration	5 timeslots
Packet arrival rate	100,000 packets/s
N_h	5
WL	125%
SL	150%
B	100
S	200

attacker is identified and marked as suspicious host at the first attacking timeslot and all the requesting packets from attacker are put in warning queue. Hence, the requests from normal host in normal queue are processed first, which results in halving the processing duration. On the other hand, at the first timeslot of normal period (11, 21, 31, 41), the processing duration for normal request does not pick up because the attacker host still have suspicious reputation and its requesting packets still have lower priority.

After that, in the next timeslot the attacker reputation is reset as normal on the next timeslot because of no violation, thus, the processing duration is doubled with the in-turn packet lookup. It can be concluded that FlowTracker is able to lower the processing priority of violated hosts to reduce the latency and dropping probability for legit request. Moreover, the penalty is automatically removed after the attacker host shows no suspicious behavior in the next timeslot. In this experiment, the penalty period is only one timeslot; however, if there is demand for higher security level, the operator can dynamically change this period to keep the attacker in the suspicious list for a longer time.

4.2. Performance testing

Stateful firewall performance has previously been evaluated in [6]. Jake et al. interests in examining the latency increment when the packet arrive rate and number of firewall rules varies. In this work, the aim is to explore a different aspect: the connection latency and controller service rate fluctuation due to the increasing number of simultaneous requests. We compare the performance of SDN network running FlowTracker with the network running solely LearningSwitch module. Due to the simple learning switch logic of LearningSwitch module, after the set up face to populate switching rules in each switch, LearningSwitch bears no controller communication overhead. Comparing FlowTracker with LearningSwitch performance eliminates the similar factors that affect the connection latency such as link traffic and bandwidth, switching packet processing time etc. to give us the clear idea of the connection latency caused by additional stateful firewall logic.

Because of different handling methods in FlowTracker, we performed separate tests for connection-oriented protocol – TCP and connectionless protocol – UDP. Ten hosts were set up as servers and the other ten hosts as clients. To generate traffics, the clients send multiple requests to corresponding servers simultaneously.

For TCP, we measured the total download time for a small file via HTTP, for UDP the round trip time of sending and replying a UDP packet is recorded. These figures reflect the latency increment when the number of concurrent connections goes up.

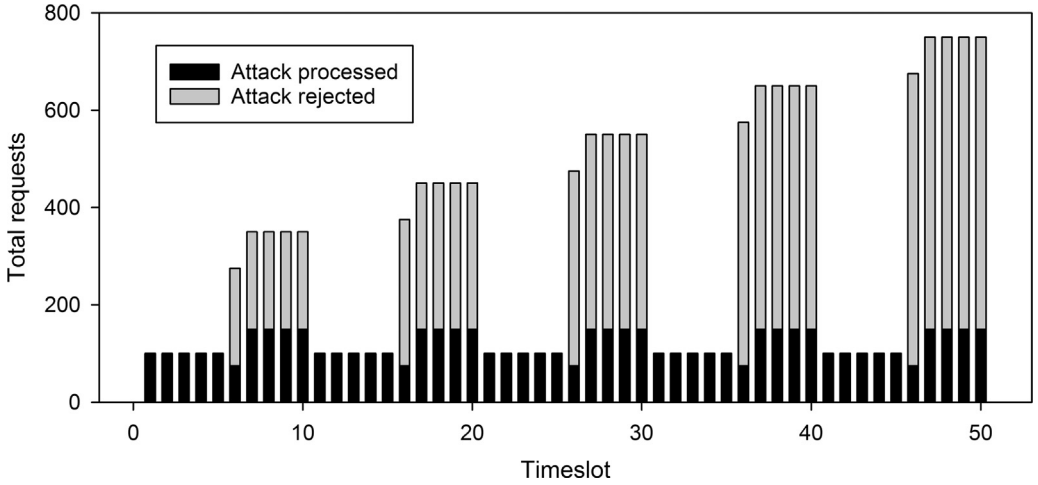

Fig. 7. Number of accepted and number of rejected attack packets.

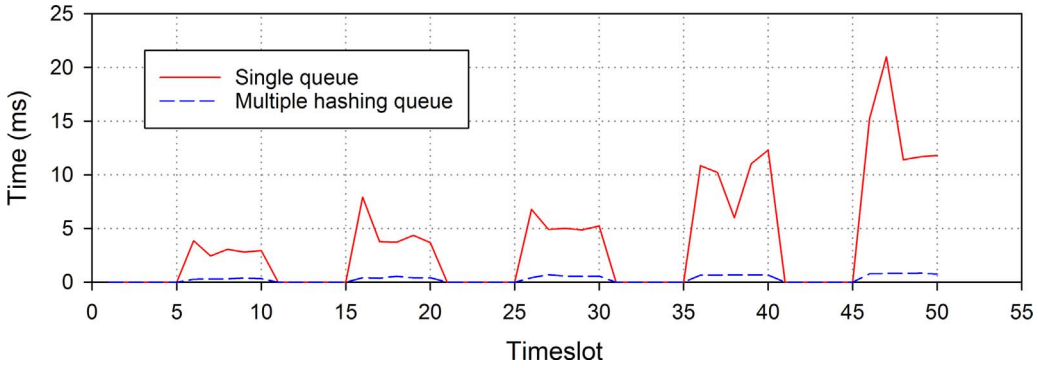

Fig. 8. Mitigation time comparison.

Beside the controller communication overhead, the service rate of controller is also measured during each request periods in case of using a single queue and using multiple hashing queue to get a better idea of how much using multiple message queues affects controller performance.

4.2.1. Controller communication overhead

Fig. 10 shows that the average TCP connection duration of FlowTracker is considerably higher than those of LearningSwitch module. By subtracting LearningSwitch number to FlowTracker number, we obtain the net latency caused by controller communication overhead. During the connection increasing period from 10 to 200, the controller communication overhead rises from 24.1 ms to 152.6 ms. From the range of 10 to 130 simultaneous connections, the difference is quite stationary in the range of 20–30 ms. However, after that time, the latency increases rapidly with the connection number from 140–200; especially with 190 and 200 connections, the latency is 3–5 times larger than the like of connections in the range of 10–130.

The difference of the FlowTracker-single queue and FlowTracker-multiple queues is significant. Although FlowTracker-multiple has a higher peak, the sum average latency of both are similar. It can be explained that the additional memory and processing for operating multiple queues is not significant within this range of connection number. The same symptoms can be seen with the Fig. 11 of UDP connection latency comparison. One observation which can be made from Fig. 11 is that after the number of on-going requests surpasses 110, the mean connection times are highly fluctuated, and even the maximum latency reaches the peak of 160 connections, greater than that of 200 connections.

4.2.2. System loads penalty of multiple queues

Regarding the possible system load of multiple queues, the service rate of FlowTracker 20-queue and single queue are compared. All 20 hosts are scheduled to send multiple TCP SYN packets simultaneously, the service rate of FlowTracker is measured in the process of handling all the requests. The service rate for FlowTracker using single queue is recorded for comparison.

Fig. 12 describes the service rate variation when FlowTracker processes from 100 to 800 pending requests. The service rates of multiple queues and single queue does not show significant difference. The results imply that using multiple hashing queues in FlowTracker provides DoS mitigation and requests balancing at a small cost of additional memory and no significant performance tradeoff. However, Fig. 12 also shows a significant drop in service rate at the high number of request. After checking the performance of each atom process, it can be observed that the most time-consuming operation in FlowTracker process is one that checks if the new request belongs to existing connection. Because of multiple header fields extracting and comparing, this action will take considerable processing time, especially when there is a high number of on-going connections in the connection data of FlowTracker.

4.3. Discussion and research limitations

4.3.1. Accuracy of estimated routine

FlowTracker uses the peak number of on-going connections in one timeslot to track host behavior. However, to avoid false alarms, the network characteristics need to be taken into consideration to decide the sampling frequency. For example, timeslot length for the servers that runs continuously could be set to one day to reduce data collection. On the other hand, with common office computer timeslot length can be flexibly changed between office hours and standby hours. Further speaking, host peak connection tracking can be used to profile the host behavior in the specific hours of the day. Generally speaking, the advantage of centralized high-level network control in SDN greatly supports network operators to dynamically configure the tracking period.

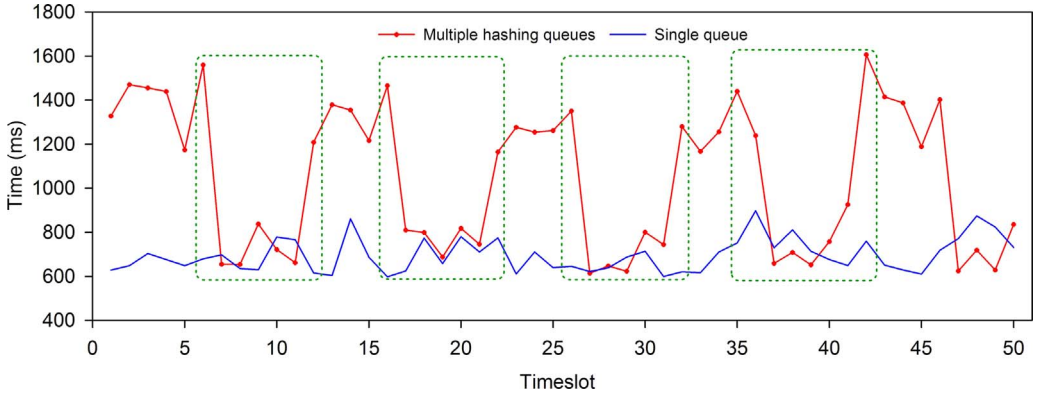

Fig. 9. Processing time for normal host.

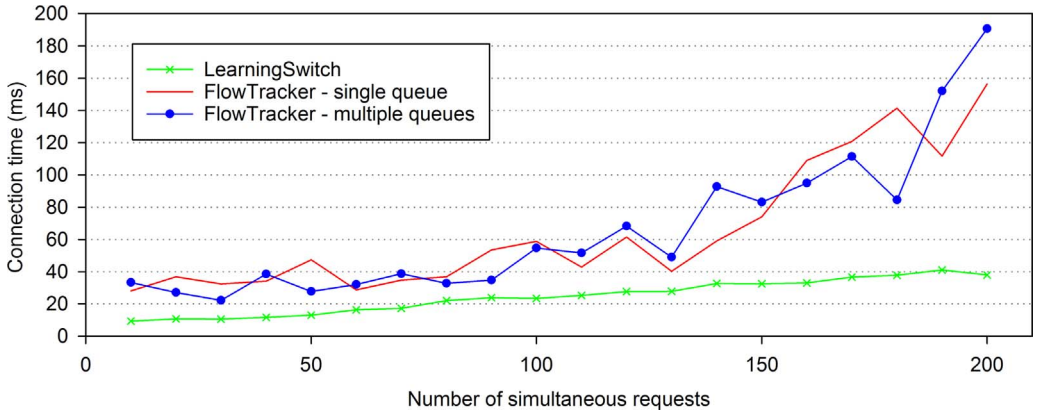

Fig. 10. Connection latency comparison (TCP).

4.3.2. Tracking scope and impact on network performance

There are concerns about the practicality of stateful firewall and its impact on network performance when all the connections are reported to the controller [7]. However, the author argues that it is not always necessary to report all the connections. In FlowTracker solution the author picks two common protocols TCP and UDP for tracking in general explanation. Nonetheless, the tracking scope can be narrowed or extended to fit the types of connection the network operator interests in. For example, FlowTracker can specifically track HTTP requests initiating from subnet 10.0.0.x/24 only. Operating stateful does not need to be an excessive delay. In other words, the tradeoff between security level and network performance can be well adjusted.

4.3.3. TCP tracking overhead

Current handling of FlowTracker still requires continuous forward traffics from one side to the controller to detect connection tear down. Although this does not cause connection delay, additional controller communication and processing overhead is inescapable. The reason for this inefficient practice is that the FlowTracker is built upon POX controller framework which only support for OpenFlow 1.0. This particular version does not support TCP flags in the list of matching fields. To the best of our knowledge, the current on-going draft OpenFlow 1.5 specification has added the support for matching TCP flags. However, at the moment this work is conducted only Ryu controller [18] and OpenvSwitch [17] just partially support for OF 1.5, and the hardware switches capable of handling OF 1.5 are still in the development program. The author would like to revisit this issue when the OpenFlow 1.5 is mature and supported well enough.

4.3.4. False positive case

Of course there is possible case when the host legit changes its routine and the number of ongoing connection increases significantly. With the DoS detection logic of FlowTracker, this host will be marked as attacker. However as explained in the 3.2.3 section, the mitigation steps performed by FlowTracker only reduces the connection number the attacker host can maintain and lower the new connection request priority from the host while the previous approved connections from the host are still carried on. So even in the false positive case, the host network communication is not completely shut down. Moreover, since the attacker record and mitigation policy are on the controller side, the network operator can remove the attacker status and/or update the limited connection baseline of the host easily and rapidly. The changes are made on the software side so it's much simpler than manually configuration on the network hardware devices.

4.3.5. TCAM over populated

In a switch, TCAM (Ternary Content Addressable Memory) is the most expensive component of the commodity switches [19] and TCAM lookup is very power and memory consuming. TCAM table has a finite capability of flow entries [20]. FlowTracker needs a separate flow entry in the switch for one connection forwarding, so that the connection forwarding entries could fill up the switch TCAM table as the number of ongoing connections in the network increase. Considering an example of an OpenFlow switch NEC PF5820 with maximum 4000 12-tuple flows, when a host maintains around 200 simultaneous connections, each switch can manage up to 20 hosts (given that the connection detection and switching flow entries are only layer 2 entries that take much less space [21]). This can be sufficient for medium size networks, however, the scalability of FlowTracker given various network sizes and data plane configurations still needs more investigation. Moreover, to solve this problem thoroughly, we have to find a solution for how to concise and group the connection forwarding entries in FlowTracker.

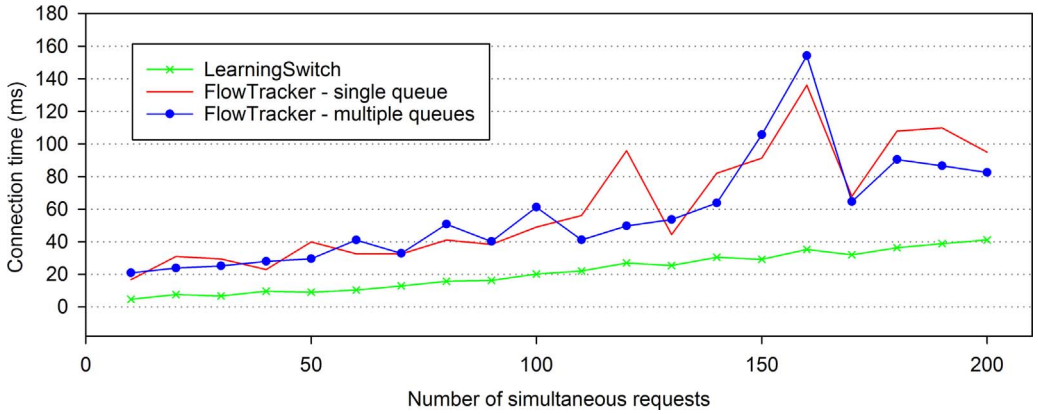

Fig. 11. Connection latency comparison (UDP).

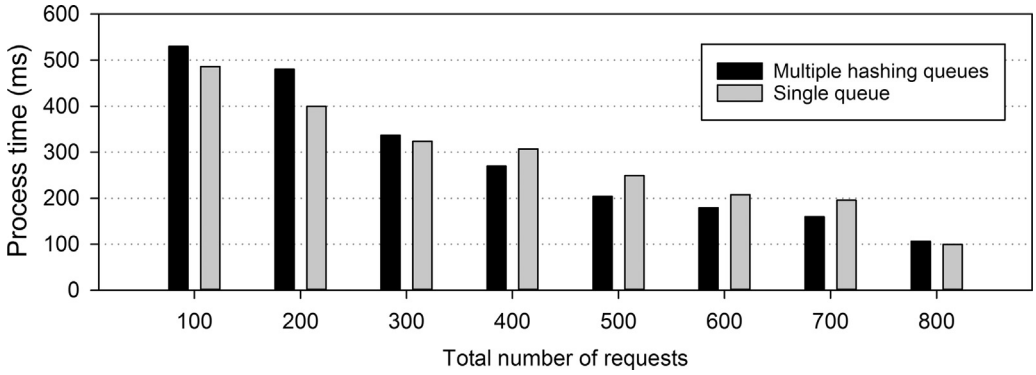

Fig. 12. Service rate comparison.

4.3.6. Multiple attackers – multiple protecting targets

FlowTracker demonstrates its effectiveness in DoS detection and migration, i.e., a single attacker case. However, in DDoS attack, where the hacker orchestrates a large number of hosts to attack the target system, is more complicated because of multiple attacker involvement. As a quick solution, by summing up the peak connection statistics of all hosts in the network, we could get the baseline requests activity for the controller and detect the abnormal increment. However, since there are multiple hosts participating in the attack, it is harder to detect if a host shows significant excessiveness in connection requests. Therefore, FlowTracker could detect when the controller under DDoS with little additional computation overhead. However, to pinpoint exactly the hosts take part in DDoS attack is still an unsolved issue.

In this work, FlowTracker only focused on protecting the controller from DoS attacks. However in many situations, the hackers might target a server in the network. To add the support for host DoS detection, FlowTracker not only needs to maintain the peak requests from one host but also the peak requests to that host in its statistic. Similarly, if there is an abnormal surge in the number of requests toward the server, FlowTracker can be alerted and perform actions. Then again this feature would require further statistics recording and processing in the controller.

5. Conclusion

In this paper, we studied the challenges of and solution to stateful SDN firewall. The delay and overload for dynamic control of stateful firewall are inherent and serious bottlenecks for stateful control. It has been shown that the challenge can be overcome in a significant level by the proposed algorithms, topology-based selective filtering rules for setup and maintenance stage, three-layer rule structure for in-switch flow tables, and FlowTracker.

Importantly, FlowTracker is able to set up the firewall flows efficiently, eliminate redundant flows and duplicate new connection reports, monitor the states of both connection-oriented and connectionless protocols. The novel DoS detection and mitigation solution incorporates well with FlowTracker while no additional network components or changes in data plane side are required. Our simulation results show that FlowTracker can achieve adaptive connection tracking in real time, complete profile host normal routine and immediate detection of suspicious behavior, and 5–20 times faster in DoS mitigating than the current controller setup with single message queue.

The new OpenFlow version 1.5 adds TCP flag matching fields in flow rules, which could be utilized to reduce the number of communication requests between controller and data plane for connection tracking. In this work, we focused on protecting the DoS attacks towards controller, but traditional DoS and DDoS attack towards an application server, such as a web server [29,30] is important issue to study. Thanks to the centralized architecture of SDN network, DDoS attack detection become much easier than in non-SDN network.

Acknowledgment

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the Global IT Talent support program (IITP-2016-R0134-16-1030) supervised by the IITP (Institute for Information and Communication Technology Promotion).

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, in: Proceedings of the ACM SIGCOMM Computer Communication Review, vol. 38, pp. 67–74, 2008.
[2] G.J.V. Pena, W.E. Yu, Development of a Distributed Firewall Using Software Defined Networking Technology, Information Science and Technology (ICIST), in: Proceedings of the 4th IEEE International Conference, April, pp. 449–452, 2014.
[3] T. Javid, T. Riaz, A. Rasheed, A layer 2 Firewall for Software Defined Network, in: Proceedings of the Information Assurance and Cyber Security (CIACS) Conference, June, pp. 39–42, 2014.
[4] K. Kaur, J. Singh, N.S. Ghumman, Programmable firewall using Software Defined Networking, in: Proceedings of the Computing for Sustainable Global Development (INDIACom) International Conference, March, pp. 2125–2129, 2015.
[5] T.V. Tran, H. Ahn, A network topology-aware selectively distributed firewall control in SDN, in: Proceedings of the ICTC International conference, Oct, 2015.
[6] C. Jake, L. Jun, An OpenFlow-based Prototype of SDN-Oriented Stateful Hardware Firewalls, in: Proceedings of the IEEE 22nd International Conference on Network Protocols, Aug, pp. 525–528, 2014.
[7] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, Z. Zhang, Enabling security functions with SDN: A feasibility study, in: Computer Networks, vol. 85, pp. 19–35
[8] L. MacVittie, SDN Prerequisite: Stateful versus Stateless, DevCentral, ⟨https://devcentral.f5.com/articles/sdn-prerequisite-stateful-versus-stateless⟩, April 2014 (accessed 14.04.16).
[9] A.B. Astuto, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A Survey of Software-Defined Networking Past, Present, and Future of Programmable Networks, in: Proceedings of the IEEE communications surveys & tutorials, vol. 16, no. 3, 2014.
[10] S. Hayward, S. Natarajan, S. Sezer, A Survey of Security in Software Defined Networks, in: Proceedings of the IEEE Communications Surveys & Tutorials, vol. 18. No. 1, July, pp. 623–654, 2015.
[11] M. Dabbagh, B. Hamdaoui, M. Guizani, A. Rayes, Software-defined networking security: pros and cons, in: Proceedings of the IEEE Communications Magazine, vol. 53, no. 6, June, pp. 73–79, 2015.
[12] S. Sezer, S. Scott-Hayward, P.K. Chouhan, Are We Ready for SDN? Implementation Challenges for Software-Defined Networks, in: Proceedings of the IEEE Communications Magazine, vol. 51, no. 7, pp. 36–43, 2013.
[13] D. Kreutz, F.M.V. Ramos, P.E. Verissimo, C.E. Rothenberg, Software-Defined Networking: A Comprehensive Survey, in: Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, 2015.
[14] OpenFlow.org, OpenFlow specification v1.5.1, April 2015, ⟨https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf⟩.
[15] GENI Experimental testbed, ⟨http://www.geni.net/⟩.
[16] NOXRepo.org, POX documentation, ⟨http://www.noxrepo.org/pox/documentation⟩.
[17] Open vSwitch, ⟨http://openvswitch.org⟩.
[18] Ryu controller, ⟨https://osrg.github.io/ryu/⟩.
[19] B. Salisbury, TCAMs and OpenFlow – What Every SDN Practitioner Must Know, ⟨https://www.sdxcentral.com/articles/contributed/sdn-openflow-tcam-need-to-know/2012/07/⟩, July 2012 (accessed 14.04.16).
[20] V. Anilkumar, P. Rishabh, M. Vijay, B. Suparna, Effective Switch Memory Management in OpenFlow Networks, in: Proceedings of ACM International Conference on Distributed Event-Based Systems, May, pp. 177–188, 2014.
[21] B. Owens, OpenFlow Switching Performance: Not All TCAM Is Created Equal,

⟨http://packetpushers.net/openflow-switching-performance-not-all-tcam-is-created-equal/⟩, Feb 2013 (accessed 14.04.16).
[22] E. Eugene Schultz, A framework for understanding and predicting insider attacks, Computers and Security journal, Vol. 21, 6, October, 2012.
[23] M. Suh, S. Park, B. Lee, S. Yang, Building firewall over the software-defined network controller, in: Proceedings of Advanced Communication Technology (ICACT) International Conference, Feb, pp. 744–748, 2014.
[24] K. Karamjeet, S. Japinder, Building Stateful Firewall Over Software Defined Networking, in: Proceedings of the Third International Conference INDIA 2016, vol. 2, Springer, Feb 2016.
[25] Big Switch Networks, Floodlight Open SDN Controller, ⟨http://www.projectfloodlight.org/floodlight/⟩.
[26] Hping3, ⟨http://www.hping.org/hping3.html⟩.
[27] G. Paul, B. Chuck, Software Defined Networks A Comprehensive Approach, Chapter 4 – How SDN Works, pp. 59–79, Elsevier, June, 2014.
[28] L. Wei, C. Fung, FlowRanger: A request prioritizing algorithm for controller DoS attacks in Software Defined Networks, in: Proceedings of the IEEE International Conference on Communications (ICC), June, pp. 5254–5259, 2015.
[29] S. Lim, J. Ha, H. Kim, Y. Kim, S. Yang, S., A SDN-oriented DDoS blocking scheme for botnet-based attacks. in: Proceedings of the 2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN), pp. 63–68, 2014.
[30] R. Braga, E. Mota, A. Passito, Lightweight DDoS flooding attack detection using NOX/OpenFlow. in: Proceeding of IEEE Proceedings of the 35th Conference on Local Computer Networks (LCN), pp. 408–415, 2010.
[31] J. Jeong, H. Kim, J. Park, T. Ahn, S. Lee, Software-Defined Networking Based Security Services using Interface to Network Security Functions, Internet-Draft: Standards Track, IETF, July 5, 2016.
[32] S. Zerkane, D. Espes, P. Le Parc, F. Cuppens, Software Defined Networking Reactive Stateful Firewall, in: Proceedings of the 31st IFIP International Information Security and Privacy Conference (SEC), May 2016, Ghent, Belgium. Springer, IFIP Advances in Information and Communication Technology, AICT-471, pp. 119–132, 2016.
[33] Belden Inc., 3 Future Trends for ICS Security, ⟨http://www.belden.com/blog/industrialsecurity/3-Future-Trends-for-ICS-Security.cfm⟩, November 2015
[34] S. Hogg, Is an SDN Switch A New Form of a Firewall?, ⟨http://www.networkworld.com/article/2905257/sdn/is-an-sdn-switch-a-new-form-of-a-firewall.html⟩, April, 2015.
[35] M. Pasccuci, Back in Time and Back to the Future: Looking at the Evolution of the Firewall, January, 2013.

Thuy Vinh Tran is a M.S. student in Internet Computing Laboratory, the Electricals and Information Engineering Department at Seoul National University of Science and Technology. He received his B.S. degree in Data Communication and Computer Networking from the Computer Science Department of Hanoi University of Science and Technology in 2012. After that, he worked for Samsung Vietnam Mobile R & D Center as a software engineer until 2014. His research interests are in the area of network security, including designing robust SDN firewalls and SDN IDS systems.

Heejune Ahn is a professor in the Electrical and Information Engineering Department at Seoul National University of Science and Technology laboratory. He received his B.S., M.S, Ph.D. degree in KAIST, South Korea, 1993, 1995, and 2000 respectively. During his Ph.D. period, he worked for MPEG and DAVIC standards. After working as Postdoc in Telecommunication lab, University of Erlangen-Nuremberg, Germany, he joined LG electronics and worked GSM/GPRS/WCDMA protocol standardization and development from 2000 to 2002, and worked J2EE WAS contributor in Tmax Soft Inc. for 2002 to 2003. His research interests are in Internet protocol standard and implementation, Data mining, and embedded software design.