Imran *et al. Hum. Cent. Comput. Inf. Sci.* (2019) 9:16
https://doi.org/10.1186/s13673-019-0176-7

Human-centric Computing and Information Sciences

RESEARCH Open Access

Reducing the effects of DoS attacks in software defined networks using parallel flow installation

Muhammad Imran[1], Muhammad Hanif Durad[1], Farrukh Aslam Khan[2*] and Abdelouahid Derhab[2]

*Correspondence:
fakhan@ksu.edu.sa
[2] Center of Excellence in Information Assurance (CoEIA), King Saud University, Riyadh, Saudi Arabia
Full list of author information is available at the end of the article

Abstract

Software defined networking (SDN) is becoming more and more popular due to its key features, such as monitoring, fine-grained control, flexibility and scalability. The centralized control of SDN makes it vulnerable to various types of attacks, e.g., flooding, spoofing, and denial of service (DoS). Among these attacks, DoS attack has the most severe impact because it degrades the performance of the SDN by overloading its different components, i.e., controller, switch, and control channel. This impact becomes more prominent in SDNs having fine-grained control over traffic for monitoring and management purposes, where large numbers of flow rules are installed. Existing approaches handle DoS attacks in SDN either by dropping malicious packets or by aggregating flow rules, resulting in a legitimate packet drop or loss of fine-grained control over network traffic. In this paper, a parallel flow installation approach is proposed to reduce the effects of DoS attacks, without losing the monitoring capability and fine-grained control over network traffic. The proposed approach installs flow rules in all switches along the path from the source to the destination on a single request from the source; resulting in a considerable reduction of control channel traffic and controller's utilization. The proposed approach is evaluated by comparing it with the basic SDN controller. The simulation results show that the proposed approach increases the SDN performance in terms of CPU utilization, response time, flow requests, and control channel bandwidth.

Keywords: Software defined networking, Denial of service attacks, DoS mitigation, Fine-grained control, Parallel flow installation

Introduction

Software defined networking (SDN) is becoming more and more popular among the network research community due to its simple configuration and easy management. SDN makes it easy to get fine-grained information about the transferred data and provides centralized control over network traffic. Therefore, it can manage all network traffic with different protocols, such as Internet Protocol version 6 (IPv6), Internet Protocol version 4 (IPv4), Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), etc., from different sources (MAC address, IP address, port number, etc.) in multiple ways [1, 2]. In addition, SDNs are more scalable and flexible as compared to legacy networks. In SDN,

the data-forwarding layer (switches and routers, etc.) is separated from the control layer (controller or operating system), which makes it ideal for large-scale and high-speed computing. The control layer has the operating system, which coordinates with the application layer, in order to perform different tasks such as intrusion detection, load balancing, routing, etc. [3, 4]. Mostly, the control layer is connected to the data forwarding layer through a standard protocol called OpenFlow (OF), which transfers the OF messages (Packet-In, Packet-Out, Flow-Add, etc.) between switches and the controller for routing and management purposes. In OF, each switch contains a flow table with flow entries added by the controller for a specific duration to forward packets. When a packet arrives at a switch, it matches the packet header with flow entries in the flow table to take the appropriate action. If there is no matching entry, then the packet header is encapsulated in a Packet-In message and is directed toward the controller for a suitable action. The controller adds a flow entry in the switch by sending a Flow-Add message containing the appropriate action. The remaining packets with the same packet header move according to the instructions in the matching flow entry [5, 6]. These days, the OF support is being provided in network switches by major vendors, such as Juniper, HP, and Cisco [7].

The SDN architecture allows a controller to manage a wide range of data plane resources and offers to simplify their configurations in a proper manner. Furthermore, the SDN architecture recommends that common models and mechanisms should be employed, wherever possible, to reduce the standardization, integration and validation efforts. In addition, it also implies utilizing existing standards or accepted best practices [2]. In SDN, security has become an important concern, as it is not a built-in feature yet in the SDN architecture. SDN is vulnerable to various kinds of security threats, such as spoofing, tampering, repudiation, information disclosure, denial of service (DoS) and elevation of privilege [8]. Among these threats, DoS has the most devastating effect as it degrades the SDN performance by increasing latency and dropping legitimate packets [9, 10]. An attacker can easily launch a DoS attack by massively sending useless packets having different source addresses so that the switch forwards each packet toward the controller in the form of a Packet-In message. This attack will overload the controller, control channel (the link between the controller and the switch), and the flow table [11]. An attacker can launch a DoS attack or its distributed version (i.e., DDoS attack) in order to choke the whole network by overloading the controller's resources (computing power and memory), or a segment of the network by overwhelming a switch's resources (flow agent, packet buffer and flow table) [12].

To defend SDN against DoS attacks, different techniques have been proposed during the past few years. Most of the techniques block suspicious traffic by installing rules to drop all packets, while other reduce the effects of an attack by load balancing, flow aggregation, priority scheduling, or using trust values. It is very difficult to judge in such a short time and with high accuracy that the data sent by a node is legitimate or fake; hence, there are chances of a loss of data sent by a legitimate node. Therefore, more reliable techniques are required that aim to prevent legitimate data drop; however, they may also suffer from problems such as the requirement of additional hardware, modification in the switch, loss of fine-grained information, additional delay in route establishment, extra control packets, or race conditions, etc.

This paper proposes a parallel flow installation model to reduce the effects of DoS attacks in SDN without compromising the fine-grained control over network traffic. The proposed model overcomes problems such as additional hardware requirement, switch modification, additional delay in route establishment, information loss, and additional control packets. The working of the proposed model is evaluated by comparing it with the basic SDN controller. The simulation results show that the proposed model increases the serving capacity of the existing SDN infrastructure by reducing the response time, lowering the CPU processing, and decreasing traffic on the control channel. In addition, the proposed model can be used with any existing mitigation approach to make it more efficient.

The remainder of the paper is organized as follows: "Working of SDN and DoS attacks" section presents the working of SDN and DoS attacks with their effects. "Related work" section reviews different methods to handle DoS attacks in SDN. Parallel flow installation model is explained in "Proposed parallel flow installation (PFI) model" section, while "Experiments and results" section evaluates the performance of the proposed model by comparing it with the default controller. Finally, "Conclusions and future work" section concludes this paper and presents some future directions.

Working of SDN and DoS attacks

In SDN, the controller being a centralized authority, keeps and maintains the updated information about the whole network. The routing process is also managed by the controller with the help of OF messages, such as Packet-In, Packet-Out, Flow-Add, etc. [2]. Switches contain flow tables to store flow rules for a limited time and forward data according to these rules. The controller keeps the network topology in its database to provide efficient and timely routing to the connected nodes. The routing applications/modules in well-known open-source SDN controllers, such as Floodlight (circuit pusher) [13], ONOS (fwd) [14], Ryu (simple_switch, simple_switch_12, simple_switch_13) [15], and POX (L2_learning, L3_learning) [16], etc., install the flow rules on OF switches in a linear fashion, where each switch in the path between the source and the destination sends a Packet-In message toward the controller in order to get the flow rule to forward newly arrived data. The network diagram shown in Fig. 1 explains the routing process between the source (Host-1) and the destination (Host-8) in SDN.

In Fig. 1, the solid lines represent connectivity between switches and hosts whereas the dashed lines represent links between the controller and switches. In SDN terminology, the link between the controller and the switch is called a control channel. The arrows represent the movement of data from the source node (Host-1) to the destination node (Host-8) at different time intervals. At time *t1*, the source node sends a data packet to Switch-D, which searches its flow table to forward the data. A table-miss occurs when no matching flow rule is found in the flow table, as a result of which the switch generates a Packet-In message and sends it to the controller through the control channel at time *t2*. The controller, after receiving the Packet-In message, searches its routing table to find the route to the destination and sends a Flow-Add message containing instructions to forward data back to Switch-D at time *t3*. If there is no route to the destination in the controller's routing table (i.e., the destination node has not yet communicated with any other node in the network), then the controller sends a Packet-Out message to Switch-D

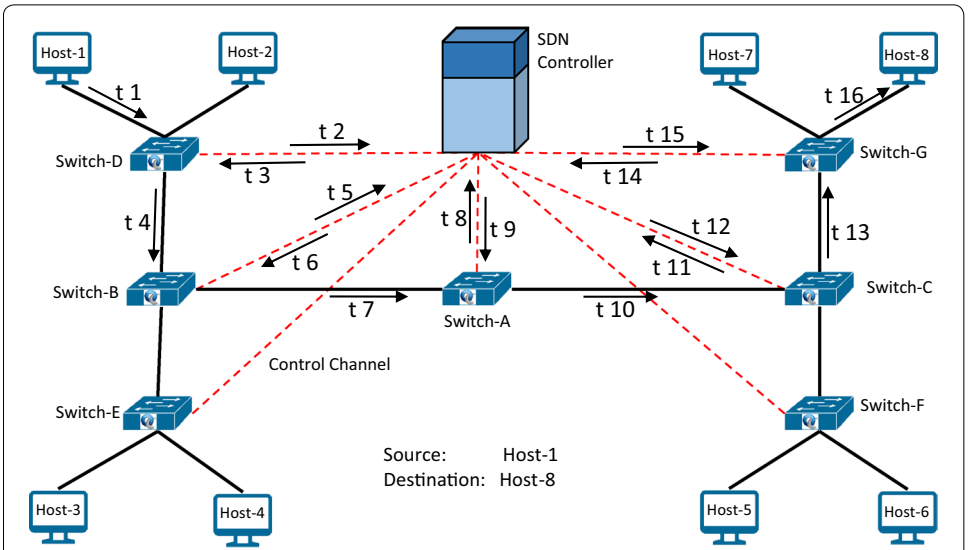

Fig. 1 Routing process in SDN

asking it to broadcast the message. Upon receiving the Flow-Add message from the controller, Switch-D forwards the data to Switch-B at time *t4*. Switch-B also sends the Packet-In message, receives the Flow-Add message, and forwards data to Switch-A at time *t5*, *t6*, and *t7* respectively. In a similar way, Switch-A, Switch-C, and Switch-G ask the controller for the path during time *t8* to *t15* and finally, Switch-G transfers data to the destination at time *t16*.

Now if the destination node also wants to communicate with the source node, then all these steps will be repeated in the reverse order. Furthermore, if these two nodes again communicate with each other after a specified time period to keep the flow rules in the flow table, then all the above-mentioned steps will be repeated again.

According to Fig. 1, if there are five switches between the source and the destination, then at least five Packet-In messages will be sent toward the controller; hence, five flow rules will be installed to transfer data toward the destination. So, the number of OpenFlow messages (Packet-In and Flow-Add) sent over the control channel by the routing apps/modules (circuitpusher, fwd, simple_switch, simple_switch_12, simple_switch_13, L2_learning, L3_learning, etc.) for a single path can be calculated using the following simple equation:

$$\text{OpenFlow messages} = S \times 2 \tag{1}$$

where "S" is the number of switches included in the path. Most of the controllers also send Packet-Out message toward the switch in addition to the Flow-Add message; in that case, "S" will be multiplied by 3. Therefore, when a host connected to the SDN communicates with another host located at 5 switches away, then the total number of OF messages will be 15 (3+3+3+3+3) and for a two-way communication between same hosts, this number will be doubled (2×15). The SDN controller and switches require some time and resources to generate and process each of these OF messages. So if t is the time required to install a single flow on a switch, then the response time (reply from destination) for a complete path can also be calculated by the following simple equation:

$$\text{Response Time} = (S \times t) \times 2 \tag{2}$$

where "t" is the time required to install a single flow. According to the above-mentioned calculations, there will not be any considerable load on the controller's processor and the control channel during normal traffic. However, when there is more traffic, then the efficiency of the network can be affected. This does not mean that SDN is unable to process heavy traffic. It has the ability to aggregate existing flow rules and can install wildcard flow rules to forward data based on the destination MAC address only.

Centralized monitoring and fine-grained control are the important features of SDN through which the data can be easily monitored up to the transport layer. The data sent to, or received from specific ports, can be blocked or re-routed. For monitoring and security purposes, a flow is mostly stored as a 12-tuple with fields that match against the incoming packets [17]. To monitor the data transferred over the network in detail (mac, network, and transport layers), a new rule is required each time when a host uses a different port to transfer the data. As a result, there will be hundreds or even thousands of flow rules on each switch during normal traffic. An attacker can exploit this feature of SDN by flooding traffic toward random destinations to keep the controller busy in useless route management.

In the SDN paradigm, it is easy for an attacker to launch a DoS attack by flooding packets with random destinations in such a way that the OF switch forwards each packet toward the controller to get a new flow rule. This attack will affect different components of the SDN infrastructure by consuming their resources such as switch's memory, control channel's bandwidth and controller's processing power. This overuse of resources may result in low throughput, high latency or a legitimate packet drop. In SDN, the DoS attack does not necessarily make the entire network or some of its segments unavailable; rather it may overload network resources in such a manner that the data is not delivered as desired [7].

Due to having different forms and the ability to target different components of the SDN infrastructure, DoS attacks are very difficult to detect and mitigate. An attacker can flood packets to different hosts in the network to consume resources of the switch and the controller (memory and processing power) or it can send large-sized packets to different hosts in the network to consume their bandwidth. A legitimate user can also have a requirement to scan the whole network repeatedly or to send a large amount of data to different nodes. Hence, it is difficult for a mitigation strategy to differentiate between an attacker and a legitimate user. Furthermore, a mitigation approach should not have a tradeoff at the cost of fine-grained flow conditions (source and destination's MAC address, IP address, and port numbers, etc.).

Related work

During the past few years, several authors have proposed different methods to counter the DoS attack in SDN. Kandoi et al. [7] presented an approach that used enhanced configuration parameters in order to decrease harmful effects of DoS attacks in SDN. They enforced the rate-limit to the number of packets sent toward the controller in a unit time. They also recommended flow aggregation and optimal idle timeout value to avoid overflow. The idle timeout has a large value in networks where hosts communicate more frequently with one another. More space is created in the flow table by the aggregation of two or more rules with a similar destination or by keeping the short idle timeout value of flows for their early

removal from the flow table. Due to flow aggregation, there will be a loss of packet statistics and other information from switches. Kloti et al. [12] performed the security analysis on the STRIDE model and proposed some solutions that could be helpful to mitigate DoS attacks in SDN. These solutions include rate limiting, flow timeout adjustments, and flow aggregation. Shin et al. [18] suggested an approach, called Avant-Guard, by adding connection migration and actuating triggers in OF switches to add some intelligence in the data forwarding plane. In this approach, upon receiving a new data packet, a switch verifies the source by the TCP handshake. If the handshake is successful, then it informs the controller, which permits connection migration to start the data transfer. Actuating triggers enable the data plane to report the network status and payload information to the control plane. Wang et al. [19] introduced OF-Guard to overcome the limitations of Avant-Guard by including the data plane cache. OF-Guard is triggered when the utilization of network resources crosses a predefined limit, after which all new packets seeking the destination are sent toward the data plane cache, which stores proactive flow rules, caches table-miss packets, and separates fake packets. Both the Avant-Guard and OF-Guard require modifications in OF switches and are limited to control the TCP traffic only.

Oktian et al. [20] designed an application to mitigate the DoS attack in the Beacon controller by using a table to save the related information about switches and connected hosts in the network. The controller collects the IP and MAC addresses of connected hosts by inspecting packets at the network startup. In addition to IP and MAC addresses, the status of ports and other switch information is also saved. By having this information, the controller has full knowledge of the network topology, attached hosts, and the status of switch's ports. Upon leaving or joining of each host, the controller updates this information, which helps it to block all traffic generated by an attacker as near as possible to its source to reduce its effects. This application only deals with MAC/IP spoofing. Belyaev et al. [21] suggested a technique to make SDN more resistive during the DoS attack with load balancing and is combined with an intrusion detection and mitigation system termed Callophrys. It first collects the topology and load information of the network, and then overrules the network with static information collected by the Bellman-Ford path finding algorithm. It splits the path having high traffic load with alternative routes. This technique shifts the load to alternate routes, which is only possible when there exist multiple paths between the source and the destination. Wang et al. [22] presented a flexible control structure to secure cloud computing and SDN with fast and specific attack reaction. It consists of two modules; one for anomaly-based detection and the other for mitigation. The detection module contains attack patterns in a database, which are matched with new flow packets. If an attack is detected, then the mitigation module takes an appropriate action after notifying the controller. The malicious packet is further investigated; if it belongs to a new type of attack, then the database is updated accordingly.

Dao et al. [23] proposed a solution assuming that abnormal users transmit less number of packets than a pre-defined value 'n' during each connection. If the transmitted packets are less than 'n', then a flow rule is installed to block traffic from that user. Additionally, flow rules installed for new packets contain lower values of the soft and hard timeouts to decrease their lifetime. Due to the low hard timeout value, the load will be more on the controller and the control channel. Kuerban et al. [24] proposed a simple method to control bandwidth, called FlowSec, where the controller constantly monitors

bandwidth of each port of the switch. If it crosses a predefined limit, then it is reduced to half in order to decrease the number of packets sent toward the controller. Padmaja et al. [25] presented a solution, in which a new entry is introduced into the flow table to keep packets of a flow in the packet buffer, to avoid the race condition. In a race condition, redundant flow requests sent to the controller cause it to calculate more flow rules for the same flow. Dridi et al. [26] suggested SDN-Guard, which coordinates with an intrusion detection system to reduce the effect of DoS attack on the SDN infrastructure. It has three modules: flow management, monitoring, and aggregation. The flow management module assigns those routes to malicious packets that have low bandwidth consumption. The high timeout values are also assigned to suspicious flows to decrease the control channel communication. To avoid table overflow, malicious flow entries are aggregated by the aggregation module. Flow management requires multiple routes and flow aggregation will result in the loss of information from switches.

Zhang et al. [27] suggested Multi-Layer Fair Queuing (MLFQ) to counter DoS attacks by maintaining queues of Packet-In messages in the controller, where messages received from each switch are placed in the corresponding queues. The controller applies the Weighted Round Robin (WRR) algorithm to pool requests from these queues. When the number of packets in a queue crosses a predefined threshold, then that queue is expanded into a per-switch queue. If the size of a queue is still larger, then the queue is further expanded into the per-port queue. On the other hand, if the size of the sub-queue drops below the threshold, then the queues are aggregated into a single queue again. This approach requires additional computation for multi-layer queue management, so the legitimate hosts connected to the switch will face more delay when the attacker exists. Wang et al. [28] presented an approach, called FloodGuard, which introduced a proactive flow rule analyzer in the controller to install proactive flow rules during the attack to reduce burden on the controller. The controller also has a migration agent, which is responsible to direct all table-miss packets toward the data plane cache, which temporarily stores packets to guard the data plane and sends them toward the controller in a rate-limited fashion using Round Robin (RR) scheduling. This solution requires additional hardware (data plane cache), modification in OF switches, and introduces a delay due to the rate limitation.

Wang et al. [29] proposed a rounding-based load balancing solution to reduce control channel congestion and response time for SDNs having multiple controllers. This algorithm performs link load balancing and controller load balancing. The authors also suggested an efficient mechanism for network status maintenance to improve their approach. Ma et al. [30] proposed a load balancing mechanism for multiple controllers by implementing a hierarchical control with a meta-control plane and local control plane. The meta-control plane analyzes the resource utilization of the local control plane to optimize the performance, whereas the local control plane optimizes the performance of the data plane and eliminates the bottleneck effect. Ganesh et al. [31] proposed a load balancing mechanism by implementing a dynamic load balancing algorithm to distribute different traffic flows via different parallel paths.

As a result of the above study, the DoS mitigation approaches can be divided into two categories. The first category contains approaches that block malicious traffic by installing rules to drop all packets from malicious hosts. The second category contains

approaches that do not drop malicious traffic. These approaches reduce the effects of DoS attacks by load balancing, priority scheduling, or using trust values, as it is very difficult to accurately figure out in such a short time that the data sent by a node is fake or legitimate. Therefore, approaches in the second category are more reliable because using these approaches, there are no chances of a legitimate data drop. The proposed system belongs to the second category, which reduces the effects of the DoS attack by optimizing control traffic toward the SDN controller in order to reduce its load.

Proposed parallel flow installation model

It is necessary to install flow rules with more and more matching fields for fine-grained management and monitoring of data statistics. For this purpose, the OF specification defines a default set of 45 matching fields [2]. Moreover, as the number of matching fields for the flows increases, more traffic will flow toward the controller in the form of Packet-In messages and the controller will utilize more resources to process them. During normal traffic, SDN works fine with more matching fields; however, when there is a DoS attack in progress, then the number of packets sent over the control channel will increase exponentially. This attack will affect SDN in the following ways:

- Response time from the destination will increase
- Controller's CPU utilization will increase
- Control channel traffic will increase.

To launch a DoS attack, the attacker generates fake route requests by sending fake data packets toward different destinations. Upon receiving these data packets, the OF switches that lie within the path send Packet-In messages to the SDN controller. In response, the controller sends Flow-Add messages to switches to forward data toward the destination. The number of control messages (Packet-In and Flow-Add) produced as a result of each fake request can be calculated using Eq. (1). More fake requests will result in the generation of more control packets due to which the controller's resources will saturate and its capacity to handle requests will decrease. Now, to reduce the impact of the DoS attack, it is essential to reduce the number of control messages, as the number of control messages is directly proportional to the controller's CPU utilization.

To reduce the number of control messages during the DoS attack, a Parallel Flow Installation (PFI) model is proposed by modifying the controller's behavior from linear to parallel for flow rule installation. A traditional SDN controller only replies to the switch from which it receives the Packet-In message as described in Algorithm 1; whereas, in the proposed model, the controller sends Flow-Add messages to all switches that lie on the path between the source and the destination. This is because it knows that these switches that are included will demand the path toward the destination in the future. To convert the behavior of the controller from linear to parallel, the function handling the Packet-In message is modified in such a way that it makes a list of switches that lie on the path from the source to the destination, and sends Flow-Add messages to them without receiving requests to install flow rules from them.

The proposed model is explained in Fig. 2 with an example, which shows the time intervals to transfer data from the source (Host-1) to the destination (Host-8). At time

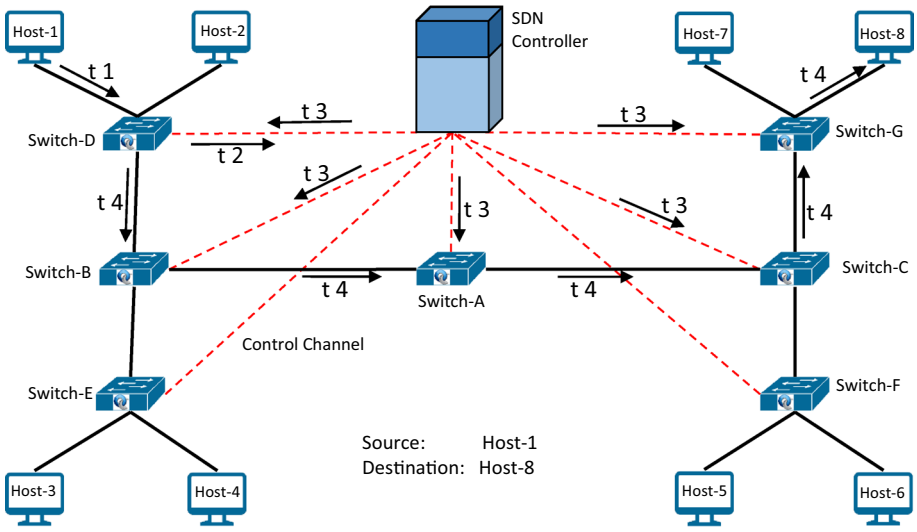

Fig. 2 Working of the proposed parallel flow installation system

t1, Host-1 sends data to Switch-D, which searches its flow table for possible existence of a matching flow rule to forward data toward the destination. Upon failure in finding the flow rule, Switch-D sends a Packet-In message toward the controller at time *t2*. The controller calculates the complete path from the source to the destination and sends flow rules to all switches (D, B, A, C, and G) present in the path at time *t3*. At time *t4*, the data moves from Switch-D to Switch-G through Switches B, A and C, and finally to Host-8, as the flow rules to forward data are already installed on them. An important thing to note in the proposed approach is that the controller sends flow rules to those switches only that are included in the path between the source and destination hosts. So, the controller will not forward any flow rule to Switch-E and Switch-F because they are not included in the path.

Algorithm 1 Existing Linear Flow Installation

// This algorithm describes the default flow installation process of FloodLight, ONOS, RYU and POX.

1: $Src \rightarrow$ **new**Pkt;
2: $Send(Pkt) \rightarrow InPort$;
3: **for all** $Switches \in Path\ from\ Src\ to\ Dst$ **do**
4: $InPort \leftarrow Pkt$;
5: **if** $Pkt.Header \in Flow\_Table$ **then**
6: $Update.Counters()$;
7: $Send(Pkt) \rightarrow OutPort$;
8: **else**
9: $Packet\_In(Pkt.Header)$;
10: $Send(Packet\_In) \rightarrow Controller$;
// Lines (11-14) will be executed on controller
11: $identify\ next\_hop$;
12: **new** $Flow\_Add$;
13: $Flow\_Add.OutPort \leftarrow next\_hop$;
14: $Send(Flow\_Add) \rightarrow Switch$;
15: $Flow\_Table \leftarrow Flow\_Add$;
16: $Update.Counters()$;
17: $Send(Pkt) \rightarrow OutPort$;
18: **end if**
19: **end for**
20: $OutPort \rightarrow Dst$;

According to the proposed model, as described in Algorithm 2, a single Packet-In message is sufficient to install flow rules on all switches included in the path from the source to the destination, so the number of OF messages over the control channel for a path can be calculated from the modified form of Eq. (1) as:

$$\text{OpenFlow messages} = \text{S} + 1 \tag{3}$$

where "S" is the number of switches included in the path.

Consider another scenario, which has 8 switches in the path from the source to the destination. As in SDN, each switch along the path will send a Packet-In message toward the controller in order to get the flow rule to forward data from the source to the destination. In response to each Packet-In message, the controller will send a Flow-Add message containing instructions to forward data toward the destination. So according to Eq. (1), there will be a total of 16 OF messages (8 Packet-In and 8 Flow-Add messages) and if the processing time of each packet is considered to be 1 ms, then the path establishment time will be 16 ms. Whereas in the proposed PFI model, only one Packet-In message will be sent toward the controller from the first switch along the path, in response to which the controller will send Flow-Add messages to all 8 switches that lie in the path from the source to the destination. Therefore, according to Eq. (3), there will be 9 OF messages (1 Packet-In and 8 Flow-Add messages) and the path establishment time will be 9 ms.

Algorithm 2 Proposed Parallel Flow Installation

```
// This algorithm describes the proposed parallel flow installation process.
Src → newPkt;
Send(Pkt) → InPort;
if Pkt.Header ∈ Flow_Table then
    Update.Counters();
    Send(Pkt) → OutPort;
else
    Packet_In(Pkt.Header);
    Send(Packet_In) → Controller;
    for all Switches ∈ Path from Src to Dst do
        identify next_hop;
        // Lines (10-13) will be executed on controller
        new Flow_Add;
        Flow_Add.OutPort ← next_hop;
        Send(Flow_Add) → Switch;
        Flow_Table ← Flow_Add;
        Update.Counters();
        Send(Pkt) → OutPort;
    end for
end if
OutPort → Dst;
```

Table 1 shows the OF messages for different numbers of switches in the path from the source to the destination.

In a similar way, the response time from the destination after installing two-way flows will be according to the following equation:

$$\text{Response Time} = (\text{t} + \alpha) \times 2 \tag{4}$$

where "t" is the time required to install a single flow and "α" is the additional processing time required by the controller to install parallel flows.

The above example logically proves that the proposed model just sends one Packet-In message to the controller for a complete path and comparatively takes less time to

Table 1 OpenFlow messages for a complete path from source to destination

No. of switch(es) in path	Linear flow installation			Proposed parallel flow installation		
	Packet-In	Flow-Add	Total	Packet-In	Flow-Add	Total
1	1	1	2	1	1	2
2	2	2	4	1	2	3
3	3	3	6	1	3	4
4	4	4	8	1	4	5
5	5	5	10	1	5	6
6	6	6	12	1	6	7
7	7	7	14	1	7	8
8	8	8	16	1	8	9

establish a path from the source to the destination. So, there will be less traffic toward the controller, as a result of which the controller will process fewer numbers of packets. In this way, the proposed model reduces the response time and saves the control channel bandwidth and the controller's CPU resources. The flowcharts for both linear and parallel flow installation mechanisms are shown in Fig. 3. The dotted part is optional and is executed only when there are multiple switches along the path.

Experiments and results

The proposed PFI model is implemented by modifying routing mechanisms of POX's L2_learning [16] and RYU's Simple_switch_13 [15] modules from linear to parallel for OpenFlow [2] version 1.1 and 1.3 respectively. To evaluate the proposed model, all experiments are performed by emulating a network topology, as shown in Fig. 2, and by using Mininet (version 2.3.0) [32] virtual network that is installed on a virtual machine and connecting it with remote POX and RYU controllers running on another virtual machine. To generate normal traffic, a script is run on Host-2 and Host-4 that send TCP traffic to Host-7 and Host-5 respectively, which changes the destination port after every 5 s in order to request new flow rule from the controller. A DoS attack is launched by executing a script on Host-3, which picks a host from a randomly generated list of IP addresses within the network range and sends a data packet to its random port after a 0.01 s interval so that each packet demands a new rule from the controller. The performance of the proposed PFI model is compared with default routing algorithms of POX and RYU (L2_learning and simple_switch_13 described in "Working of SDN and DoS attacks" section) on different benchmarks such as response time, controller's CPU utilization, control channel bandwidth, and flow requests sent to the controller. The results for each benchmark are discussed below for both the controllers (POX and RYU) with and without a DoS attack.

Response time

It is the time duration between sending a packet and receiving its reply from the destination. Response time is calculated by sending a ping message from "Host-1" to all other hosts. The experiment was repeated multiple times to calculate the average response time.

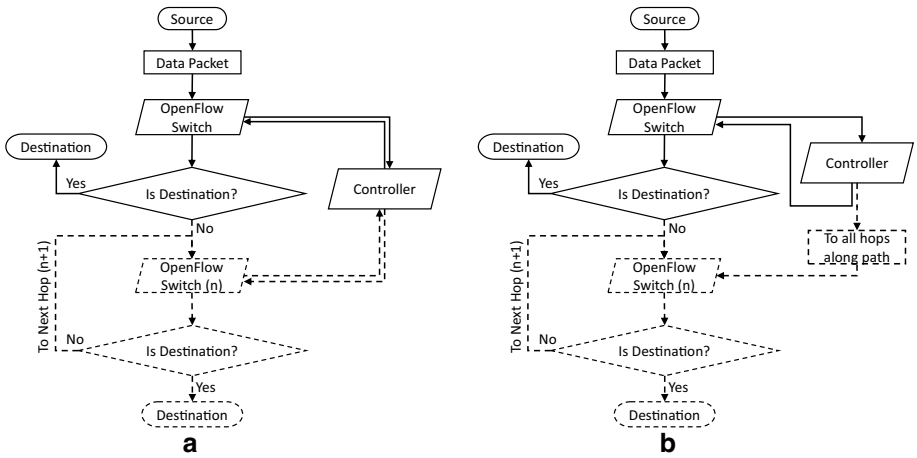

Fig. 3 Flowchart comparison **a** linear flow installation, **b** proposed parallel flow installation

Figure 4 shows the average response time for each host for both POX and PFI controllers without a DoS attack (dashed bar and wave bar respectively) and with a DoS attack (diagonal line bar and crossed line bar respectively). The shorter lengths of bars for PFI and PFI-DoS can be observed as compared to the bars for POX and POX-DoS respectively, which shows that the PFI model has a shorter response time as compared to POX for almost all hosts. The average response time of POX and PFI without a DoS attack is 38.26 ms and 26.29 ms respectively for all hosts, while the average response time of POX and PFI with a DoS attack is 52.14 ms and 39.96 ms respectively. So, the proposed system has a 31.29% less response time in the absence of a DoS attack, while it has a 23.36% less response time in the presence of a DoS attack.

Similarly, Fig. 5 shows the average response time for each host for both RYU and PFI controllers without a DoS attack (dashed bar and wave bar respectively) and with a DoS attack (diagonal line bar and crossed line bar respectively). Here, it can again be observed that the bars for PFI and PFI-DoS are shorter as compared to the bars for RYU and RYU-DoS respectively, which clearly indicates that the PFI model has a shorter response time as compared to RYU. Here, the average response time of RYU and PFI without a DoS attack is 9.42 ms and 6.79 ms respectively for all hosts, while the average response time of POX and PFI with a DoS attack is 15.09 ms and 12.31 ms respectively. So, the proposed system has a 27.88% less response time in the absence of a DoS attack, while it has 18.43% less response time in the presence of a DoS attack. The reason behind this reduction in response time is the parallel flow installation, in which the controller processes only one Packet-In message for a complete path from the source to the destination and installs flow rules on next switches in the path before the arrival of data on those switches.

CPU utilization

It is the percentage of workload on controllers during the simulation period. To calculate the CPU utilization, its per-second usage was recorded for a period of 300 s.

Figure 6 clearly shows that the CPU usage of PFI-DoS (solid line with spheres) is lower than that of POX-DoS (solid line with squares) during the DoS attack, while during

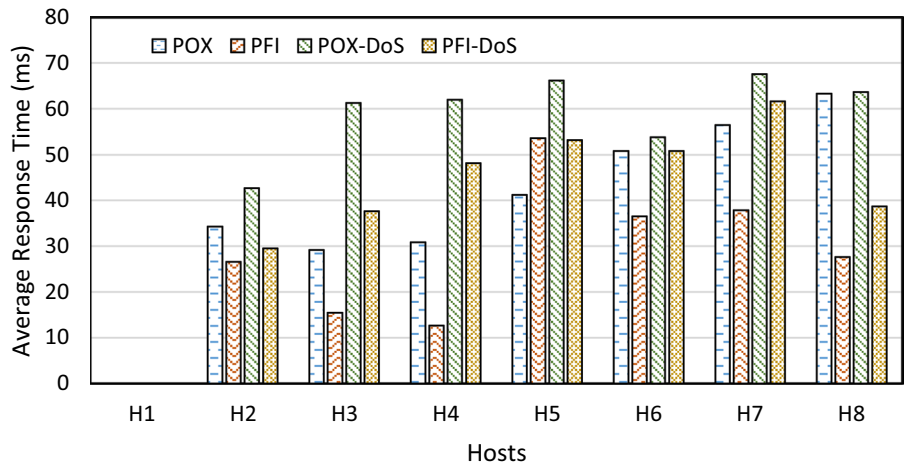

Fig. 4 Response time comparison with POX controller

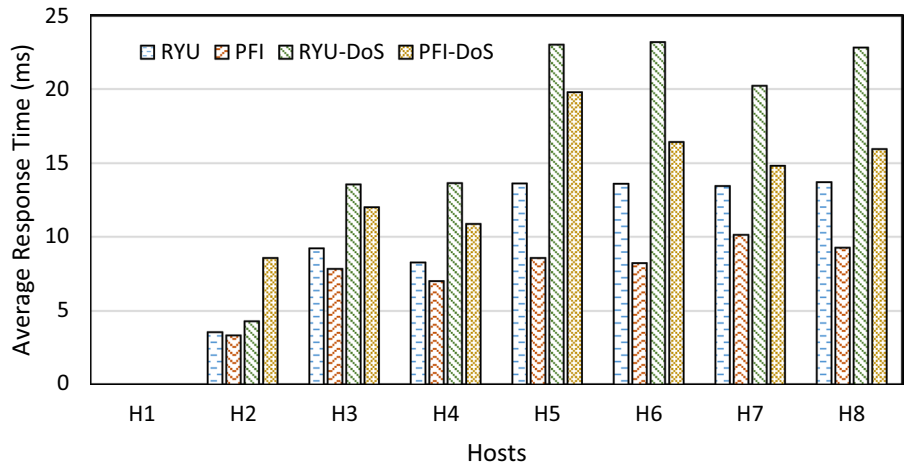

Fig. 5 Response time comparison with RYU controller

normal traffic, the CPU usage of both PFI (dotted line with spheres) and POX (dotted line with squares) are almost equal. The average CPU utilization for POX and PFI without a DoS attack is 5.87% and 6.11% respectively. On the other hand, the average CPU utilization for POX and PFI with a DoS attack is 20.47% and 12.82% respectively. So the proposed model has about 37.39% less CPU utilization during the DoS attack.

Similarly, in Fig. 7, it can be observed that the average CPU utilization of RYU (dotted line with squares) is almost equal to that of PFI (dotted line with spheres) during the normal traffic, which is 7.04% and 6.96% respectively. On the other hand, during a DoS attack, PFI-DoS (solid line with spheres) has much lower average CPU utilization as compared to RYU-DoS (solid line with squares), which is 20.70% and 30% respectively. So the proposed model has about 31.01% less CPU utilization during the DoS attack. The reason for the reduction in CPU utilization is the parallel flow installation where the controller has to process less number of Packet-In messages for the whole path from the source to the destination. As a result, its CPU utilization will be reduced.

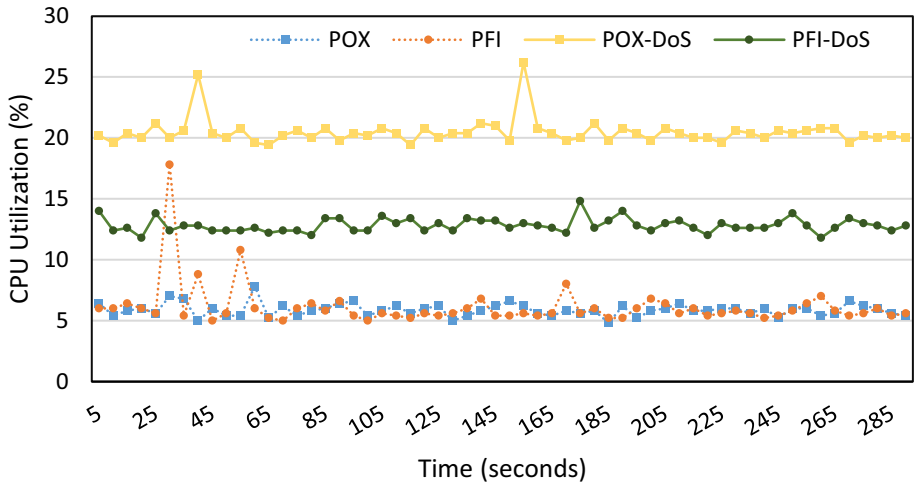

Fig. 6 CPU utilization of POX controller

Control channel bandwidth

It is the amount of data transferred over the control channel between the controller and the switches. As discussed in the previous section, switches in the proposed approach send less number of Packet-In messages to the controller, so the amount of data toward the controller is also low. To calculate the amount of data sent toward the controller, the per-second network statistics were recorded for 300 s duration.

Figure 8 shows the average amount of data transferred between the controller and switches for POX (dotted line with squares) and PFI (dotted line with spheres) controllers with the normal traffic is 15.08 kb/s and 6.61 kb/s respectively. On the other hand, during the DoS attack, the average amount of data transferred for PFI-DoS (solid line with spheres) is 83.40 kb/s, which is much smaller than POX-DoS (solid line with squares) having a data rate of 269.43 kb/s. So, there is about 56.17% and 69.04% decrease in the data rate for PFI and PFI-DoS respectively.

In the same way, Fig. 9 shows that the average amount of data transferred between the controller and switches for RYU (dotted line with squares) and PFI (dotted line with spheres) controllers with the normal traffic is 7.0 kb/s and 4.31 kb/s respectively. On the other hand, during the DoS attack, the average amount of data transferred for PFI-DoS (solid line with spheres) is 134 kb/s, which is much smaller than the RYU-DoS (solid line with squares) having a data rate of 301 kb/s, which again shows that the proposed model has 38.46% and 55.62% less data transfer rate as compared to RYU and RYU-DoS respectively. This reduction in the use of control channel bandwidth is due to the decrease in the number of Packet-In messages sent toward the controller from the switches.

Flow requests

It is the request, which is sent to the controller via the control channel from an OF switch to get a flow rule to transfer the data. It is sent in the form of a special packet called a Packet-In message.

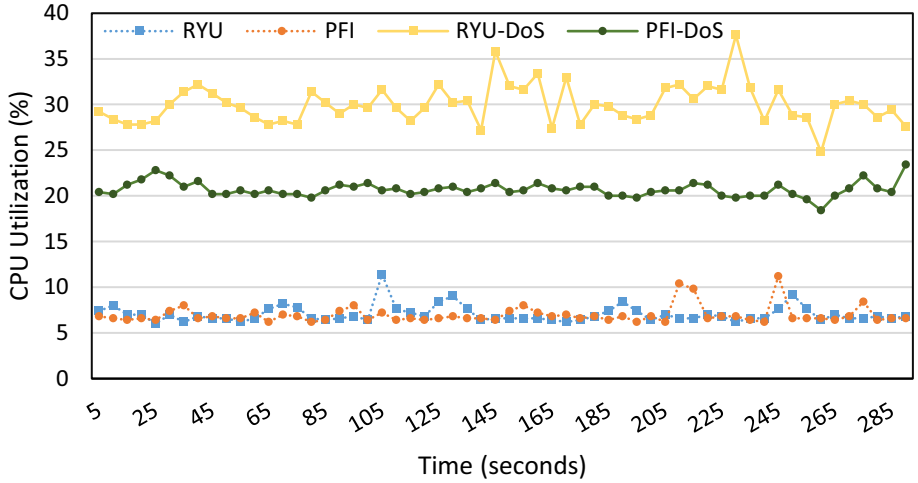

Fig. 7 CPU utilization of RYU controller

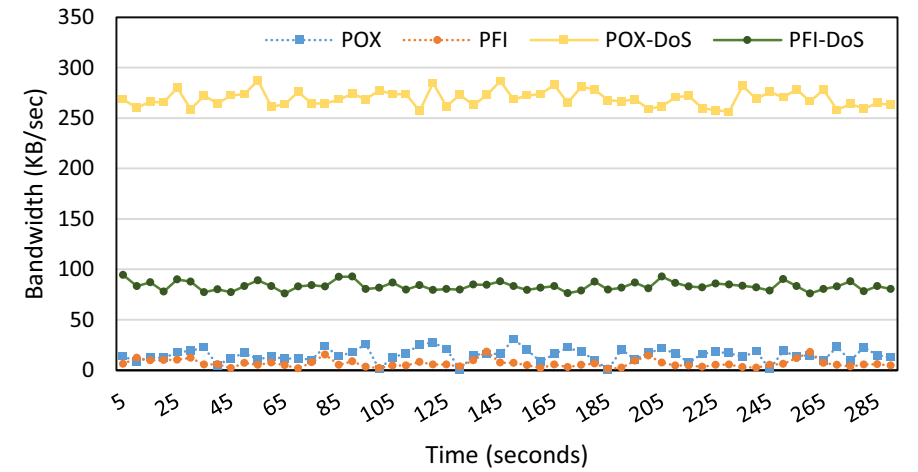

Fig. 8 Data sent over control channel for POX controller

In Fig. 10, the bars show that the Packet-In messages sent toward the controller for POX (dashed bar) and PFI (wave bar) with the normal traffic are almost equal, whereas those for PFI-DoS (crossed lines bar) are much less than POX-DoS (diagonal lines bar). The average Packet-In messages sent toward the controller by each switch for POX-DoS and PFI-DoS are 7730 and 1901 respectively. So, the proposed approach sends about 75.40% less Packet-In messages, as represented in Eq. (3).

Similarly in Fig. 11, the bars show that the average Packet-In messages sent toward the controller by each switch for RYU (dashed bar) and PFI (wave bar) with normal traffic are 287 and 96 respectively, whereas for RYU-DoS (diagonal lines bar) and PFI-DoS (crossed lines bar), these messages are 8353 and 2369 respectively. So, the proposed model sends 66.53% and 71.63% fewer packets with normal traffic and with a DoS attack respectively. This reduction in flow requests is directly related to the parallel flow installation.

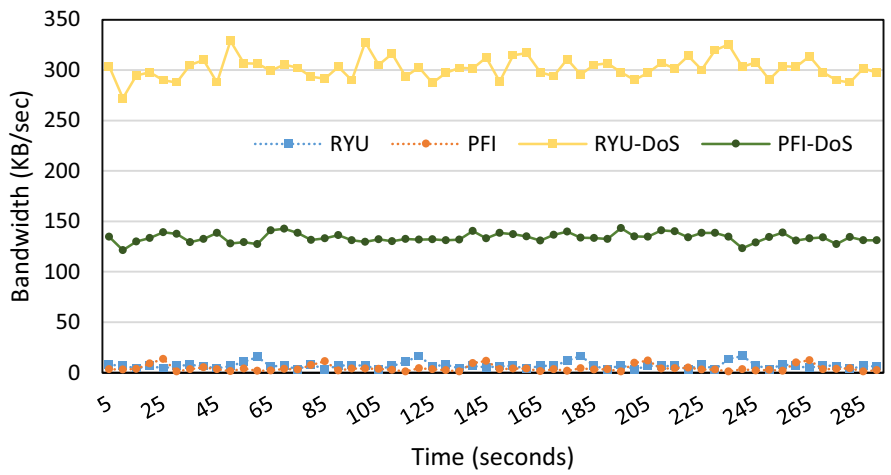

Fig. 9 Data sent over control channel for RYU controller

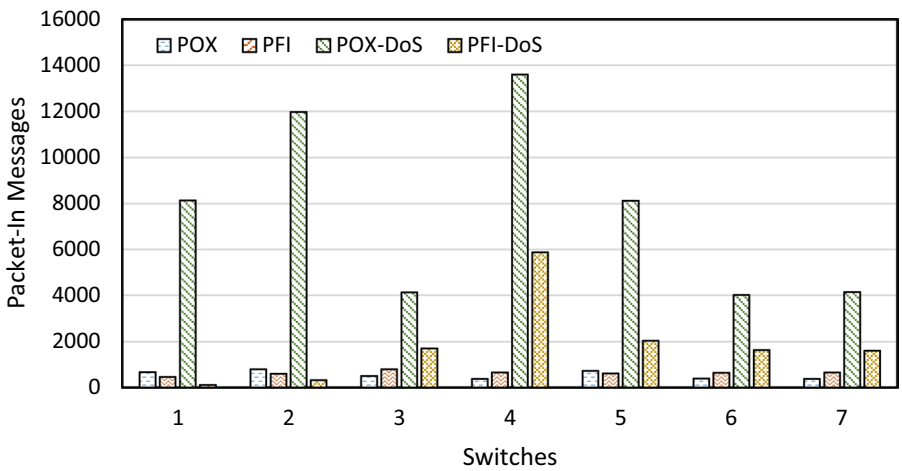

Fig. 10 Packet-In messages sent to POX controller

Comparison with other approaches

Table 2 shows the comparison of the proposed model with other state-of-the-art approaches on the basis of various parameters, such as additional hardware requirement, switch modification, loss of fine-grained flow information and additional delay. The comparison shows that the proposed model does not have these kinds of drawbacks.

Conclusions and future work

The centralized nature of software defined networking (SDN) makes it vulnerable to DoS attacks, which can disable the whole network or a component of the network and can degrade its performance. The study of existing techniques to counter DoS attacks in SDN shows that some of these techniques use complex methods, require additional equipment, or need modified switches. In addition, these techniques may also introduce additional delay in the routing process and can add more traffic to the control channel

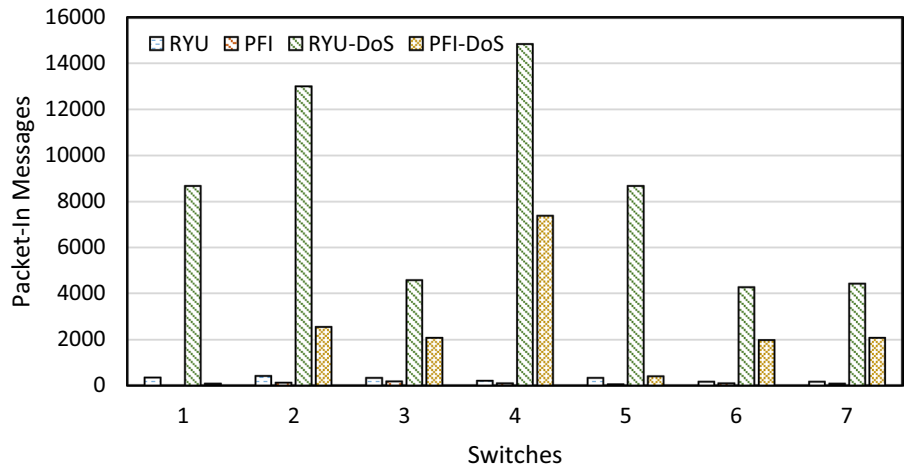

Fig. 11 Packet-In messages sent to RYU controller

Table 2 Comparison with other state-of-the-art approaches

Approach/ Reference	Mechanism used	Additional hardware	Switch modification	Loss of information	Additional delay
Kandoi et al. [7]	Flow aggregation Idle timeout management	No	No	Yes	Yes
Kloti et al. [12]	Rate limiting Flow aggregation	No	No	Yes	Yes
Belyaev et al. [21]	Load balancing	Yes	No	No	Yes
FlowSec [24]	Rate limiting	No	No	Yes	Yes
Padmaja et al. [25]	Flow rule for buffer action	No	Yes	No	Yes
SDN-Guard [26]	Flow aggregation Timeout management	No	No	Yes	Yes
Zhang et al. [27]	Multi-layer queue management	No	No	No	Yes
FloodGuard [28]	Packet migration Proactive flow rule calculation	Yes	Yes	Yes	Yes
Wang et al. [29]	Controller load management	Yes	No	No	Yes
Proposed model	Parallel flow installation (PFI)	No	No	No	No

for verification purposes. These techniques may drop legitimate packets or are unable to detect intelligent DoS attacks. So it is very difficult to completely mitigate DoS attacks without these tradeoffs. Considering all the above-mentioned problems, a parallel flow installation (PFI) model is proposed to make SDN (especially those that are configured for fine-grained control over network traffic) more tolerant to DoS attacks. Although the proposed model does not stop the DoS attack directly, it efficiently saves the control channel bandwidth and the controller's processing power to make it available for legitimate users. Furthermore, the proposed model can be used with any existing method to make it more efficient as well as to increase the serving capacity of the existing SDN infrastructure. The results of our experiments show that the proposed PFI model reduces the time to establish a path, lowers controller's processing, and decreases control channel traffic. These results will encourage the SDN research community to make

SDN more efficient and reliable. In the future, we plan to test the proposed model for a complex topology to check its scalability and design a simple, efficient and lightweight DoS attack detection and mitigation system for SDN.

Authors' contributions
MI, MHD, FAK and AD were involved in proposing the system idea, performing the experiments, and writing the manuscript. All authors read and approved the final manuscript.

Author details
[1] Department of Computer and Information Sciences, Pakistan Institute of Engineering and Applied Sciences (PIEAS), Islamabad, Pakistan. [2] Center of Excellence in Information Assurance (CoEIA), King Saud University, Riyadh, Saudi Arabia.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
Please contact authors for data requests.

Funding
The authors would like to extend their sincere appreciation to the Deanship of Scientific Research at King Saud University, Saudi Arabia, for its funding of this research through the Research Group Project No. RGP-214. The research was supported in part by the Higher Education Commission of Pakistan through PIN no. 315-7318-2EG3-116.

Publisher's Note

Received: 15 December 2018 Accepted: 25 March 2019
Published online: 01 May 2019

References

1. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J (2008) Openow: enabling innovation in campus networks. SIGCOMM Comput Commun Rev 38:69–74
2. OpenFlow Switch Specification, Version 1.5.1.; 2015. https://www.opennetworking.org/software-defined-standards/specifications/. Accessed 04 Sept 2018
3. Kreutz D, Ramos FMV, Verissimo PE, Rothenberg CE, Azodolmolky S, Uhlig S (2015) Software-defined networking: a comprehensive survey. Proc IEEE 103:14–76
4. Schaller S, Hood D (2017) Software defined networking architecture standardization. Comput Stand Interfaces 54:197–202
5. Karakus M, Durresi A (2018) Economic viability of software defined networking (SDN). Comput Netw 135:81–95
6. Hu F, Hao Q, Bao K (2014) A survey on software-defined network and OpenFlow: from concept to implementation. IEEE Commun Surv Tutorials 16:2181–2206
7. Kandoi R, Antikainen M (2015) Denial-of-service attacks in OpenFlow SDN networks. In: 2015 IFIP/IEEE international symposium on integrated network management (IM). IEEE, New York
8. Ahmad I, Namal S, Ylianttila M, Gurtov A (2015) Security in software defined networks: a survey. IEEE Commun Surv Tutorials 17:2317–2346
9. Alsmadi Izzat, Dianxiang Xu (2015) Security of software defined networks: a survey. Comput Secur 53:79–108
10. Imran M, Durad MH, Khan FA, Derhab A (2019) Toward an optimal solution against denial of service attacks in software defined networks. Future Generat Comput Syst 92:444–453
11. Anand N, Babu S, Manoj B (2018) On detecting compromised controller in software defined networks. Comput Netw 137:107–118
12. Kloti R, Kotronis V, Smith P (2013) Openflow: a security analysis. In: 21st IEEE international conference on network protocols (ICNP). IEEE, New York
13. Floodlight OpenFlow Controller. http://www.projectoodlight.org/floodlight/. Accessed 04 Sept 2018
14. Open Network Operating System (ONOS). https://onosproject.org/. Accessed 04 Sept 2018
15. Ryu SDN Framework. https://osrg.github.io/ryu/. Accessed 04 Sept 2018
16. The POX Controller. https://github.com/noxrepo/pox. Accessed 04 Sept 2018
17. Dillon C, Berkelaar M (2014) OpenFlow (D) DoS Mitigation. Technical report. http://www.delaat.net/rp/2013-2014/p42/report.pdf. Accessed 04 Sept 2018
18. Shin S, Yegneswaran V, Porras P, Gu G (2013) AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In: 2013 ACM SIGSAC conference on computer & communications security (CCS 2013). ACM, New York
19. Wang H, Xu L, Gu G (2014) OF-GUARD: a DoS attack prevention extension in software-defined networks. In: The Open Network Summit (ONS), Santa Clara, CA, 3–5 March 2014
20. Oktian YE, Lee S, Lee H (2014) Mitigating denial of service (DoS) attacks in openflow networks. In: International conference on information and communication technology convergence (ICTC). IEEE, New York
21. Belyaev M, Gaivoronski S (2014) Towards load balancing in SDN-networks during DDoS-attacks. In: First international science and technology conference (Modern Networking Technologies) (MoNeTeC). IEEE, New York

22. Wang B, Zheng Y, Lou W, Hou YT (2015) DDoS attack protection in the era of cloud computing and software-defined networking. Comput Netw 81:308–319
23. Dao NN, Park J, Park M, Cho S (2015) A feasible method to combat against DDoS attack in SDN network. In: 2015 International conference on information networking (ICOIN). IEEE, New York
24. Kuerban M, Tian Y, Yang Q, Jia Y, Huebert B, Poss D (2016) FlowSec: DOS attack mitigation strategy on SDN controller. In: 2016 IEEE international conference on networking, architecture and storage (NAS). IEEE, New York
25. Padmaja S, Vetriselvi V (2016) Mitigation of switch-DoS in software defined network. In: 2016 international conference on information communication and embedded systems (ICICES). IEEE, New York
26. Dridi L, Zhani MF (2016) SDN-Guard: DoS attacks mitigation in SDN networks. In: 5th IEEE international conference on cloud networking (Cloudnet). IEEE, New York
27. Zhang Peng, Wang Huanzhao, Chengchen Hu, Lin Chuang (2016) On denial of service attacks in software defined networks. IEEE Netw 30:28–33
28. Wang H, Xu L, Gu G (2015) Floodguard: a dos attack prevention extension in software-defined networks. In: 45th annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, New York
29. Wang H, Xu H, Huang L, Wang J, Yang X (2018) Load-balancing routing in software defined networks with multiple controllers. Comput Netw 141:82–91
30. Ma YW, Chen JL, Tsai YH, Cheng KH, Hung WC (2017) Load-balancing multiple controllers mechanism for software-defined networking. Wireless Pers Commun 94:3549–3574
31. Ganesh S, Ranjani S (2015) Dynamic load balancing using software defined networks. In: International conference on current trends in advanced computing (ICCTAC)
32. Mininet: an instant virtual network on your laptop (or other PC). http://mininet.org/. Accessed 31 July 2018