SDNShield: Towards More Comprehensive Defense against DDoS Attacks on SDN Control Plane

Kuan-yin Chen, Anudeep Reddy Junuthula, Ishant Kumar Siddhrau, Yang Xu, H. Jonathan Chao
Department of Electrical and Computer Engineering
NYU Tandon School of Engineering
Brooklyn, New York
{kyc257, arj331, iks229, yang, chao}@nyu.edu

***Abstract*—While the software-defined networking (SDN) paradigm is gaining much popularity, current SDN infrastructure has potential bottlenecks in the control plane, hindering the network's capability of handling on-demand, fine-grained flow level visibility and controllability. Adversaries can exploit these vulnerabilities to launch distributed denial-of-service (DDoS) attacks against the SDN infrastructure. Recently proposed solutions either scale up the SDN control plane or filter out forged traffic, but not both. We propose SDNShield, a combined solution towards more comprehensive defense against DDoS attacks on SDN control plane. SDNShield deploys specialized software boxes to improve the scalability of ingress SDN switches to accommodate control plane workload surges. It further incorporates a two-stage filtering scheme to protect the centralized controller. The first stage statistically distinguishes legitimate flows from forged ones, and the second stage recovers the false positives of the first stage with in-depth TCP handshake verification. Prototype tests and dataset-driven evaluation results show that SDNShield maintains higher resilience than existing solutions under varying attack intensity.**

Keywords*—*software-defined network (SDN), distributed denial-of-service (DDoS), scalability, security

I. Introduction

Facing increasing administrative complexity and demand for service innovation in production networks, the information technology industry is moving forward with *software-defined networking* (SDN) [1], a revolutionary networking paradigm that enables unprecedented level of controllability and automation. The core idea of SDN is to decouple the control plane logic from forwarding devices to a logically centralized controller. It breaks the ossified legacy architectures and enables the control plane and data plane to evolve independently: The simplified programmable data plane focuses on performance optimization, while the centralized controller possesses a global abstraction of the network that greatly facilitates network-wide coordination, automation and customization.

OpenFlow protocol [1], [2] is the first and most widely adopted standard interface between the decoupled control and data plane. OpenFlow implements *reactive* flow installation by default. That is, each new flow generates a flow request transaction upon its arrival to get routed in the network. When an OpenFlow switch receives a packet that doesn't have a matching entry in its flow tables, it treats the packet as the arrival of a new flow. The packet is then encapsulated in an OpenFlow `packet_in` message, which is sent to the OpenFlow controller for routing/policing decision via a secure TCP connection. In return, the controller sends out `flow_mod` messages to program a forwarding route into the data plane. We refer to the entire path traversed by a flow request transaction as the SDN *control path.*

Although the network administrator can proactively install policies to handle these flows in an aggregated manner, many innovative SDN-inspired network applications in fields like traffic engineering [3] and service chaining [4], [5] rely on reactive installation for per-flow level customization and control. While reactive flow installation achieves fine-grained controllability, several studies [6], [7] have shown that current SDN infrastructure have potential bottlenecks that could hinder the network's capability of handling large amounts of flow request transactions. Particularly, the hardware and software constraints along the SDN control path between edge OpenFlow switches and the centralized controller are the limiting factors [6]. The bottlenecks give rise to the threat of *distributed denial-of-service* (DDoS) attacks against the SDN infrastructure. We name this type of attacks *SDN-DDoS*: Malicious third parties can hire an army of compromised hosts, a.k.a. zombies, to flood the edge OpenFlow switches with large amounts of spoofed flow arrivals, with non-repetitive random header patterns. Without proper protection, this would in turn flood the SDN control plane with flow requests. Legitimate flow arrivals are crowded out by forged ones, leading to performance degradation and massive interruption of the entire SDN network. Unlike traditional DDoS attacks, most of which target only at an end host or service, SDN-DDoS attacks overwhelm the SDN network infrastructure, and have a much broader scope of damage.

The scalability issues of SDN control plane and potential threat of DDoS have gained much research interest recently [6]–[10]. There are two main approaches of defense against SDN-DDoS attacks. The first approach is to *scale up*: improve SDN control plane's capability to accommodate higher control workload. For example, Scotch [6] identifies that the software components of commodity OpenFlow switches are a major bottleneck of the SDN control plane. On the other hand, server-based software switches can handle higher amounts of control plane transactions. Scotch elastically scales up the SDN control plane capacity by using a software switch-based overlay network, and protects SDN network edges, i.e. the commodity OpenFlow switches, from anomalous control workload surges.

The scale-up approach focuses on providing higher capacity to SDN network edges, but forged flows are not scrubbed and make their way to the centralized controller. The second approach is to filter: check each flow's validity and block the forged ones from the SDN network. For example, Avant-Guard [9] implements TCP SYN cookie and proxy functions [11]–[13] in the data plane, and only admits those who can complete the TCP three-way handshake into the SDN control plane. Avant-Guard effectively eliminates the impact of adversarial behaviors, e.g. TCP SYN flooding, on the centralized controller.

We believe that a more comprehensive defense can be achieved by joining the strengths of both approaches. In this article, we propose *SDNShield*, a defense framework that protects both the SDN network edges and the centralized controller from SDN-DDoS attacks. SDNShield deploys an array of customized software switches named *attack mitigation units* (AMU) near the SDN network edges, and leverage their elastic processing power to overcome the bottlenecks at the SDN network edges. Furthermore, the AMUs implement a two-stage filtering scheme to protect the centralized controller. The first stage, *statistical differentiation* (SD), identifies legitimate flows with low complexity. While the flows rejected by the SD stage may contain some false positives, they are in turn inspected by the second stage, *TCP connection verification*, to ensure that all legitimate flows are accepted.

Our work makes the following contributions:

- We analyze the bottlenecks in commodity forwarding devices and identify subsequent vulnerabilities of SDN network operation.
- We identify and demonstrate the impact of SDN-DDoS, a novel security threat for current SDN infrastructure, and state implications to the design of effective countermeasures.
- We propose SDNShield, a defense framework that joins the strengths of software switches and filtering algorithms to protect the SDN control plane against SDN-DDoS attacks.
- We perform extensive evaluation to demonstrate that SDNShield is capable of effectively defending the SDN control plane against SDN-DDoS attacks, with less penalty on flow setup time compared to alternative schemes.

The rest of this article are organized as follows: Section II discusses the threat of scalability issues of SDN operation and threats of SDN-DDoS. Section III illustrates the details of system design of SDNShield. The evaluation scheme and results are provided in Section IV. Section V discusses related works and our difference from them. Finally, Section VI draws the conclusion and indicates future work directions.

II. Scalability Issues and Security Threats

A. Scalability Issues of Commodity OpenFlow Devices

A fundamental difference of SDN network operation from legacy is that the control plane decisions are not made locally at forwarding devices, but entirely at the centralized controller. The controller and forwarding devices communicate via an SDN interface protocol to request resources, exchange states and configure the network to administrator's needs. While there are a number of competing candidates, OpenFlow is so far the most widely adopted standard for SDN interface protocol among the information technology industry. By default, OpenFlow enforces reactive flow instantiation. Each new flow arrival triggers a flow request transaction between on-path forwarding devices and the OpenFlow controller. Understanding the underlying scalability issues of these transactions requires a deeper look into the OpenFlow control path.

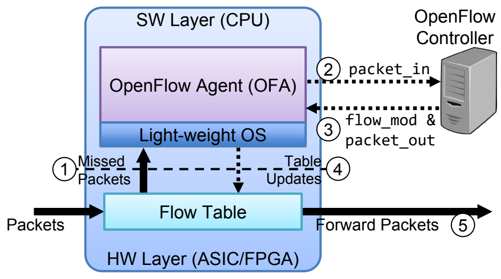


Fig. 1: OpenFlow control path and flow request transaction

Figure 1 illustrates the structure of the OpenFlow control path. An OpenFlow switch consists of two layers: the hardware layer and the software layer. The main component of the hardware layer is are programmable ASIC or FPGA modules optimized for tasks including table lookup, buffering and forwarding. The software layer runs an instance of light-weight operating system (OS), usually a miniature adaptation of Unix/Linux [14], [15], on top of a general-purpose CPU. OpenFlow protocol-related tasks such as message creation, parsing and measurement collection are handled by OpenFlow Agent (OFA), which runs as a user-space process on top of the OS.

Figure 1 also shows a walkthrough of the flow request transaction: When an ingress OpenFlow switch's hardware layer receives a packet that doesn't match any entry in its flow tables, the packet will be treated as the arrival of a new flow. The packet is by default sent to the software layer via the switch's internal interface (Step 1). The OFA buffers the new flow packet, encapsulates all or part of the packet's information in an OpenFlow `packet_in` message, and then sends it to the OpenFlow controller via a pre-configured secure channel (Step 2). When the controller receives the `packet_in` message, it makes routing decision based on the encapsulated information. In return, the controller will send out `flow_mod` messages to install flow table entries to switches on the route. Meanwhile, it will also send out a `packet_out` message to notify the ingress switch to release the buffered packet, if it is buffered (Step 3). The switches parse the return messages in the OFAs, and configure their flow tables according to the controller's instructions (Step 4). When transaction completes, the flow's packets will be forwarded along the configured route (Step 5).

A flow request transaction involves multiple resources and poses several scalability issues along the control path. We

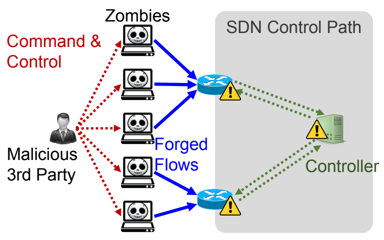


Fig. 2: Scenario of a SDN-DDoS attack

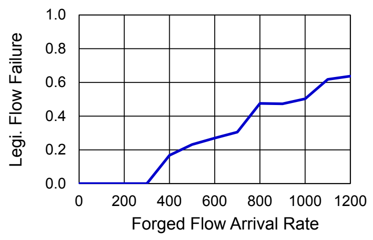


Fig. 3: Impact of SDN-DDoS on legitimate flows

define the switch-to-controller control path as inbound (Step 1 and 2), and its controller-to-switch counterpart as outbound (Step 3 and 4). Here we discuss the underlying issues of each control path component:

1. Inbound Control Path: The major bottleneck of the inbound control path is the commodity switch's OFA. For cost reasons, many commodity OpenFlow switches run their software layers on top of low-end, single-core or dual-core embedded CPUs. This limits the OFAs capacity of handling OpenFlow protocol events and messages. Wang et al. [7] examined two models of commodity top-of-rack OpenFlow switches, and found out they are only capable of creating an average of 500 to 1000 `packet_in` messages per second. The switch's protocol handling performance can also be hindered by other factors, such as the narrow interface between software and hardware planes, and the encryption requirements on the secure channel.

2. Outbound Control Path: The two limiting factors of the outbound control path are the commodity switch's limited flow table update speed and capacity. Commodity switches rely on ternary content addressable memory (TCAM) for line-speed lookup tables. However, TCAM comes in limited size, and is a scarce resource in the face of possible DDoS threats. On the other hand, a commodity switch's table update speed is limited by several factors: the slow OpenFlow `flow_mod` message parsing at the OFA, the narrow interface between the switch's software and hardware planes, and the priority-based rule insertion at the TCAM table.

3. Centralized OpenFlow Controller: The centralized controller easily becomes a bottleneck and single point of failure of the entire SDN network. Although the controller can be dynamically scaled up with server clusters and protected by hot redundancy, still we consider the case in which the controller cannot be quickly provisioned, and desire to shield the controller from the attack of forged flows for maximum security.

B. *Threat of SDN-DDoS*

With several potential bottlenecks along the control path, current SDN infrastructure falls short of supporting high-frequency flow request transactions. This weakness can be exploited by adversaries to initiate SDN-DDoS attacks that cause massive interruption of normal SDN operation. Figure 2 shows an example attack scenario. The attacker can employ an army of zombies to simultaneosly send large amounts of forged flows, with randomly permuted flow signatures. This will deplete SDN control path resources and crowd out legitimate flows.

We demonstrate the impact of SDN-DDoS on legitimate traffic with the following experiment: We inject to Pica8 P-3297 [16], a commodity OpenFlow switch, legitimate flows at a fixed arrival rate of 200 flows/sec, and forged flows at a variable arrival rate. We ensure the OpenFlow controller is powerful enough such that it doesn't become the bottleneck of the control path. As shown in Figure 3, at an attack intensity over 300 flows/sec, some legitimate flow requests get dropped. More than 60% of legitimate flows fail to install at an attack intensity of 1200 flows/sec.

Unlike traditional DDoS attack scenarios, in most of which only a specific host network or service is targeted, SDN-DDoS attacks the network infrastructure itself and hence is a much more serious threat. Also, it is worth noticing that one forged packet is sufficient to trigger a flow request and all together the attack does not consume much bandwidth, making it easier for malicious third parties to launch and deliver the attacks.

III. SDNShield System Design

The essential idea of SDNShield is to deploy a defense line of NFV-based *Attack Mitigation Units* (AMU) to scalably and intelligently protect the SDN control path from SDN-DDoS bombardment. There are several benefits in developing SDNShield under the NFV paradigm: First, AMUs run on general-purpose servers, which can be purchased at low costs thanks to economy of scale. Second, software implementation allows higher level of customization at a faster development cycle than hardware. Third, virtualization allows SDNShield to elastically scale up to meet defense requirements.

Before going through the design of SDNShield, Section III-A discusses how SDNShield is motivated by recent trends in NFV deployment and software switch technologies:

A. *Trends in NFV and Software Switches*

There is a trend among major Internet service providers (ISP) of transitioning from traditional vender hardware-oriented networking into the NFV paradigm. A notable example is AT&T's vision of Domain 2.0 [17]: ISPs are deploying NFV infrastructure (NFVI) clouds near their backbone network edges, aka. provider edges (PE), and moving the complexity of service customization into the cloud environment. This helps ISPs to achieve reduced vendor hardware costs, elastic scalability, and faster innovation. Following this trend, we expect that computational resources will become increasingly available at network edges.

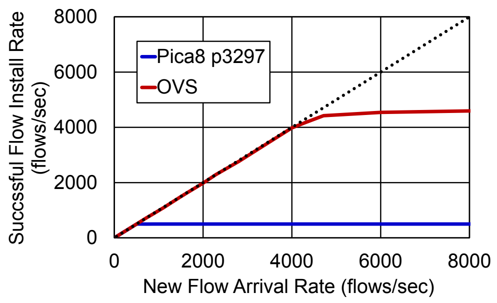


Fig. 4: Performance of OVS vs. commodity OpenFlow switch under SDN-DDoS attack

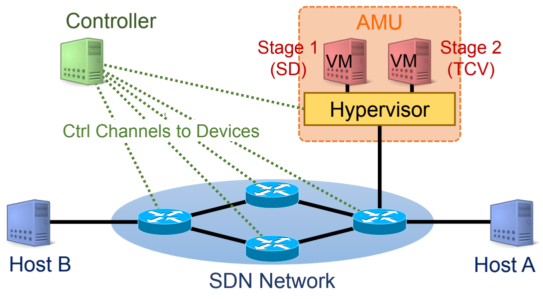


Fig. 5: SDNShield System Architecture

Meanwhile, software switch solutions like Open vSwitch (OVS) [18] potentially achieve better control path scalability than commodity hardware ones. As was discussed in Section II-A, a commodity OpenFlow switch's capability of handling flow arrivals is limited by its weak OFA in the software layer, and limited table size in the hardware layer. Software switches could outperform commodity OpenFlow switches in this part for several reasons: The high-end CPUs equipped in general-purpose servers grant OFA more computing power for OpenFlow protocol processing, and the abundance of memory allows much larger lookup tables, mitigating the table size limit problem.

Figure 4 shows a comparison between an OVS instance and a physical OpenFlow switch (Pica8 P-3297 [16]) under massive overload situations. We compare the commodity switch against an OVS instance running on a virtual machine (VM) equipped with a dedicated Intel Xeon 2.60 GHz core and 8 gigabytes memory. While the physical switch can only endure a workload of around 500 flows/sec, the OVS handles up to 4000 flows/sec without dropping any flow. This demonstrates our argument that software switches could better handle flow request transactions than commodity switches. We also test the flow table update speed of physical and software switches by sending out `flow_mod` messages (with no rule priority) from the OpenFlow controller. Our measurement results show that the physical switch can only support up to 2000 updates/sec while the software switch can achieve more than 10000 updates/sec.

However, software switches suffer from the drawbacks that they have lower forwarding throughput as well as higher packet latency. These are due to the facts that general-purpose servers are not optimized for data plane forwarding, and the buffer management mechanisms in many existing software switch implementations bring undesired overhead to packet forwarding. Although recent solutions mitigate these inefficiencies [19], as a guideline, it is still desired to differentiate legitimate flows from attacking flows, and have them installed on physical switch paths as much as possible to minimize the impact.

B. SDNShield System Overview

Figure 5 shows the overall architecture of SDNShield. SDNShield coordinates a defense line of AMUs, which are deployed on-demand in NFVI clouds at network edges. An AMU is a specialized cluster of VMs which can either run on a single server or a server cluster, and is connected to the protected edge switch via a data port. For ease of management, we assume an one-to-one mapping between the edge switches and AMUs.

SDNShield incorporates two defense stages. The first stage is *statistical differentiation* (SD). The design of the SD stage is to quickly identify the legitimate flows, send their flow requests via the usual control path, and route them via high-throughput physical switch routes. In this way, most legitimate flows achieve higher throughput and low overhead on their flow setup time.

However, the statistical approach inevitably yields false positives. That is, there are some legitimate flows falsely classified as suspicious ones. Knowing that most Internet traffic and critical connections are TCP-based, to address the false positive issue, we add a second stage, *TCP connection verification* (TCV), to double-check the rejected flows. This stage ensures that legitimate TCP flows can get routed with some extra verification.

The details of our two-stage defense scheme are provided in Section III-C and III-D, respectively.

C. Statistical Differentiation (SD) Stage

The Statistical Differentiation (SD) stage is inspired by PacketScore [20], [21] and its serial works. The key idea is to distinguish legitimate flows from forged ones by estimating each flow's likelihood of being legitimate. This can be done by comparing recently measured traffic characteristics to a reference model of normal network behavior. The flows with high likelihood of being legitimate are referred to as *in-profile* flows, and are directly requested via the usual SDN control path for minimized overhead on flow setup time. Flows with low likelihood of being legitimate are referred to as *out-of-profile* flows, and requires further check which will be discussed in Section III-D.

We discuss the details of the SD stage in three phases: detection, differentiation and divert.

1. Detection Phase: The controller continuously monitors each edge switch's performance metrics, particularly the arrival rate of `packet_in` messages. A potential SDN-DDoS attack is detected when the `packet_in` arrival rate of a switch

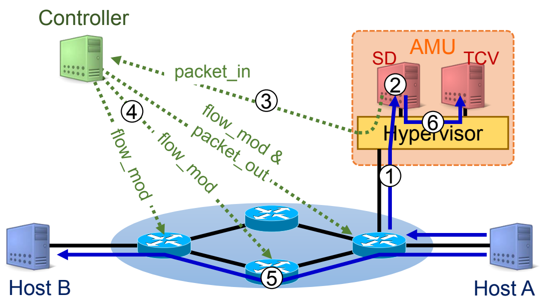
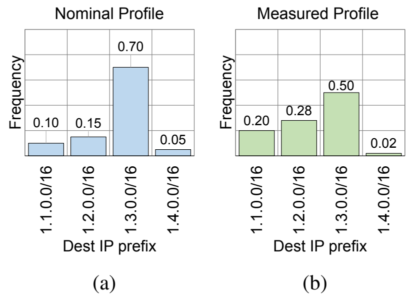


Fig. 6: Defense flow of statistical differentiation stage



dst IP	Score
1.1.0.0/16	-0.301
1.2.0.0/16	-0.222
1.3.0.0/16	+0.121
1.4.0.0/16	+0.398

Workload Score
= -0.301

(c)

Fig. 7: A miniature example of (a) nominal and (b) measured profiles and (c) the calculated scoreboard

exceeds a certain threshold. In response, the controller will activate SDNShield defense by modifying the switch's default forwarding rule (supported by OpenFlow version 1.3 and above), such that table-missing packets are forwarded to the corresponding AMU via a data port. In this way, the packets will circumvent potential bottlenecks in the weak software layer of the switch, as was discussed in Section II-A. In our case we use a threshold of 80% of the switch's maximum admissible rate, which is measured offline.

2. Differentiation Phase: To differentiate legitimate flows from forged ones, the system keeps two types of profiles: (1) The *nominal profile*, a predictive model that characterizes the expected normal behavior, calculated from historical measurements; (2) The *measured profile*, which is a snapshot of current statistical footprints of flow arrivals. A scoreboard system is then calculated from the two profiles to evaluate "how likely an incoming flow request is legitimate".

Building the profiles: Each of the two profiles inspects the five attributes that define a transport layer flow, i.e. the 5-tuple (source IP, destination IP, source port, destination port, protocol identifier). For each attribute, the profile keeps a histogram of frequency distribution in terms flow counts. Figure 7a and 7b give a miniature example of the two profiles, showing only the histograms of destination IP.

Evaluating likelihood of legitimacy: The statistical differentiation stage uses a *conditional legitimate probability* (CLP), a Bayesian-theoretic metric ,as the core metric to evaluate the "how likely a flow is legitimate". The definition of CLP is described in the following: We denote measured and nominal profiles by subscripts m and n respectively. ρ_n and ρ_m denote the total workload in nominal and measured profiles, in terms of flows/sec. We also denote the 5-tuple attribute values by A, B, C, D and E. Suppose the SD stage receives a flow f with 5-tuple attribute values $A = f_a, B = f_b, C = f_c, D = f_d$ and $E = f_e$. then the CLP of f is:

$$\begin{aligned} CLP(f) &= Prob(f \text{ is legitimate} | A = f_a, B = f_b, C = f_c, ...) \\ &= \frac{\rho_n}{\rho_m} \frac{JP_n(A = f_a, B = f_b, C = f_c, ...)}{JP_m(A = f_a, B = f_b, C = f_c, ...)} \end{aligned} \tag{1}$$

Here JP_n and JP_m denotes joint probability in nominal and measured traffic, respectively. Assuming that the five tuples are mutually independent, the joint probability equals to the multiplication of individual attribute value's marginal probabilities. We denote the marginal probability of $X = f_x$ in profile y by $P_y(X = f_x)$. Then, Equation 1 can be rewritten as:

$$\begin{aligned} CLP(f) &= \frac{\rho_n}{\rho_m} \times \frac{JP_n(A = f_a, B = f_b, C = f_c, ...)}{JP_m(A = f_a, B = f_b, C = f_c, ...)} \\ &= \frac{\rho_n}{\rho_m} \times \prod_{X=A,B,C,D,E} \frac{P_n(X = f_x)}{P_m(X = f_x)} \end{aligned} \tag{2}$$

Deriving a scoreboard system: Equation 2 can be transformed from the multiplicative/divisive form into additive/subtractive form, which is more friendly to general-purpose CPUs, by taking logarithm of both sides:

$$\begin{aligned} log(CLP(f)) = \; & (log(\rho_n) - log(\rho_m)) \\ & + \sum_{X=A\cdots E} (P_n(X = f_x) - P_m(X = f_x)) \end{aligned} \tag{3}$$

In this way, a simple additive scoring system can be derived from Equation 3. We define each attribute value's score as $score(X = f_x) = log(P_n(X = f_x)) - log(P_m(X = f_x)$, the workload score as $score(\rho) = log(\rho_n) - log(\rho_m)$. Then, the total score of flow f as $score(f) = CLP(f)$ is:

$$\begin{aligned} score(f) = \; & score(\rho) \\ & + \sum_{X=A,B,C,D,E} score(X = f_x) \end{aligned} \tag{4}$$

That is, a flow's score is the sum of its workload score and each attribute's score, so for 5-tuple, there are five scoreboards, and a workload score. Figure 7c shows the destination IP scoreboard calculated from profiles shown in Figure 7a and 7b. It is easy to see that when the measured workload is higher than nominal, the workload score will be negative. On the other hand, if the marginal probabilities $P_m(X = f_x) < P_n(X = f_x)$, $score(X = f_x)$ will be positive, which indicates that flows with $X = f_x$ have higher probabilities to be legitimate. As a result, legitimate flows generally receive higher scores than forged ones.

3. Divert Phase: Equation 4 defines a scoreboard system where anomalous flow requests have lower CLPs, and therefore receive lower scores. SDNShield compares each flow's score against a threshold. Flows with higher scores are ruled as in-profile and directly requested via the usual SDN control path, and installed to the physical switches, as was shown in Step 3 to 5 in Figure 6. Flows with lower scores will be ruled as

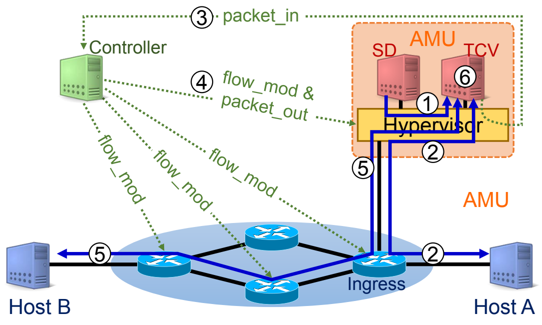


Fig. 8: Defense flow of TCP connection verification stage

out-of-profile, and forwarded to the second stage, TCV, for in-depth TCP connection verification. The threshold is dynamic, and periodically adjusted by the controller to maintain an overload control goal, i.e., at any time, the rate of passed flows is approximately a pre-specified target workload, ρ_{target}, regardless of the attack intensity.

Discussion: It is worth noticing that the scoreboard system can be incorporated into a multi-table pipeline, which is supported by OpenFlow standard version 1.3 and above. The multi-table pipeline structure readily implemented in OVS with performance optimization and requires only minor modifications to meet our needs.

D. TCP Connection Verification (TCV) Stage

The SD stage is efficient in differentiating legitimate and forged flows. However, it also intrinsically generates false positives, i.e. legitimate flows falsely classified as out-of-profile. Though false positive flows are relatively rare, in an SDN environment, dropping these flows directly is not recommended, as this may cause some legitimate users blocked out indefinitely.

Knowing that most of the Internet traffic and a majority of critical connections are based on TCP, we develop a second stage of defense, TCP connection verification (TCV), to further check the validity of these flows. The general idea of TCV stage is to locally verify that a new TCP flow can complete a three-way handshake, before allowing it to inject a flow request to the SDN control path. In this way, the control path as well as the centralized controller are protected from TCP SYN flooding attacks, since forged TCP flows with spoofed source IP can not complete the three-way handshakes,

The TCV stage is inspired by *TCP SYN cookie* and *proxy* [9], [11], a popular DDoS countermeasure adopted by commercial systems like Cisco TCP Intercept [12], Juniper SYN Cookie Protection [13] and latest Linux releases. Figure 8 shows TCV stage's defense flow, where Host A is initiating a TCP connection to Host B. The TCV Stage contains the following phases:

1. Handshake verification phase: When the first packet of a TCP flow, i.e. the SYN packet, is classified as out-of-profile in the SD stage, it is not directly dropped by the AMU but redirected to the TCV stage (Step 1 of Figure 8). The TCV stage plays the role of *SYN proxy*, sending back a SYN+ACK packet on behalf of Host B. The initial sequence number is elaborately chosen by SYN cookie encode algorithm [11]. When a returning ACK packet from Host A is received, the TCV stage can verify the connection's validity using the corresponding decode algorithm (Step 2). Instead of keeping track of each handshake's state, SYN cookie decoding is done in a *stateless* manner, reducing the risk of memory exhaustion at the AMU.

The AMU must deliver the SYN+ACK packet to Host A via the ingress switch. However, mapping each host IP to its ingress port at the physical switch would create exactly the same scalability problem we discussed in Section II. To solve this problem, we can encapsulate the ingress port information in a VLAN tag for each incoming SYN packet. The returning SYN+ACK packet also encapsulate this information, so the physical switch can correctly deliver the SYN+ACK packet to Host A, without keeping individual host-to-port mapping.

2. Flow request and connection completion phase: The AMU completes the three-way handshake with Host on behalf of Host B, but Host B has no knowledge about this connection yet; hence the AMU must launch the second-round handshake with Host B, on behalf of Host A. To establish the connection, the AMU sends a `packet_in` message to request a route from Host A to Host B (Step 3). The `packet_in` message encapsulates the SYN packet of the second-round handshake, which is later injected by the controller with a `packet_out` message (Step 4). The SYN packet reuses the initial sequence number chosen by Host A in the first-round handshake for TCP protocol conformity.

3. Connection relay phase: Host B will complete the handshake via the established route (Step 5). In the second-round SYN+ACK packet, Host B may choose an initial sequence number different from that chosen by AMU in the first-round handshake. To ensure that the SYN proxy is transparent to both sides of the TCP connection, the route between Host A and B must traverse the AMU, and the AMU must relay both sides of the TCP connection by performing *sequence/acknowledgement number translation* (Step 6). There exists a constant offset between the initial sequence numbers on both sides, so the translation is a constant add/subtraction operation to align the sequence/acknowledgement numbers. In cases that the second-round handshake fails due to invalid destination IP or rejection by Host B, the AMU will notify the controller and remove the route.

Discussion: The TCV stage has higher computational complexity on each processed packet than the SD stage, as it involves more complicated operations such as hashing and L4 header modifications. The TCV stage ensures that all legitimate TCP flows are verified and routed in the SDN network. It does not apply to non-TCP flows. However, the SD stage takes the 5-tuple into consideration, and therefore can detect and reject forged UDP flows with high accuracy.

Metric	Definition
True positives (TP)	# of Forged flows ruled as out-of-profile
True negatives (TN)	# Legitimate flows ruled as in-profile
False positives (FP)	# Legitimate flows ruled as out-of-profile
False negatives (FN)	# Forged flows ruled as in-profile
False positive rate (FPR)	FP/(FP + TN)
False negative rate (FNR)	FN/(FN + TP)

TABLE I: Definition of SD Stage accuracy metrics

Attack intensity (flows/sec)	TP	TN	FP	FN	FPR (%)	FNR (%)
100	14762	92432	831	15331	0.891	50.945
200	46605	92810	453	14397	0.486	23.601
300	77801	93096	167	14052	0.179	15.298
600	166472	93039	224	15615	0.240	8.349
1200	343987	93127	136	15517	0.146	4.316
2400	706175	93148	115	14294	0.123	1.984
4800	1430598	93135	128	14816	0.137	1.025

TABLE II: Accuracy measurements of the SD stage

IV. Evaluation

A. Prototype Tests

We prototype the SD and TCV stages on two separate VMs running on top of a KVM hypervisor. Each VM is assigned a dedicated Intel Xeon core running at 2.60 GHz, and 8 gigabytes of memory.

SD Stage Scalability: We integrate SD stage's score lookup tables (see Section III-C) into OVS's multi-table pipeline structure. Our experiment results show that our prototyped SD stage can handle an average of 330000 5-tuple score lookups per second.

A concern is that whether using a finer-grained nominal profile, which installs more entries at the each stage at the multi-table pipeline, will lead to decreased score lookup throughput. Our experiment results shows that when we use 4096 buckets in the source/destination IP stages, the lookup throughput is 91.7% of that yielded by the case of 1024 buckets. This indicates that fine-grained nominal profile may have slightly negative impact on the SD stage's processing capacity.

TCV Stage Scalability: We implement the TCP cookie functions with PCAP library and socket programming. Experiment results show that our TCV stage can handle an average of 27690 handshakes per second.

B. Data-driven Evaluation

We further evaluate the performance of SDNShield with simulations that incorporate real packet traces to create a realistic baseline traffic pattern. Besides the baseline scenario in which no defense is implemented, we compare our scheme in particular to adaptations of two major related works: (1) Scotch [6] and (2) Avant-Guard [9].

Simulation Setup: We use a single OpenFlow switch topology in the simulator. Network links are modeled as fixed propagation delays (1 millisecond each) and processing units, e.g. switch's OFA and OpenFlow controller's routing application, are modeled as $m/m/1$ queues. We draw the parameters from the measurements from Section III-A as well as prototype results from Section IV-A.

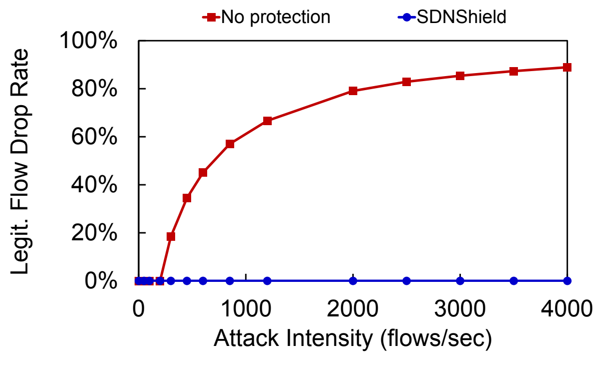
Data set and SD Stage parameters: We choose WIDE-MAWI Working Group's DITL 2009 data set [22] as the basis of our data-driven evaluation. DITL 2009 is a set of packet trace collected from a transit link of WIDE backbone in Tokyo, Japan. Its 96-hour duration is sufficiently long to cover daily periodic pattern of Internet traffic. We conduct our study over a 300-second period starting from April 1, 2009 12:00 pm (Japan local time), and use traffic from both the immediate previous time window and exactly one day before as the training data for nominal profiles. We Extract TCP flows from the packet traces, and downsample to match our simulation settings. As a result, there are a total of 93263 legitimate TCP flows over the 300-second test period, which is 311 flows/sec in average. These legitimate flows are then mixed with forged TCP flows with randomly generated source/destination IP addresses and port numbers, to create SDN-DDoS scenarios.

Accuracy of SD Stage: We define the SD stage accuracy metrics in Table I, and show the results in Table II. We set the target workload of the SD stage slightly higher than the legitimate flow arrival rate, which results in very low FPR but also higher FNR at low attack intensities (100 and 200 flows/sec). Since the design of SDNShield focuses on protecting the SDN infrastructure, we allow some forged flows to be ruled as in-profile as long as the SDN control path is not overwhelmed.

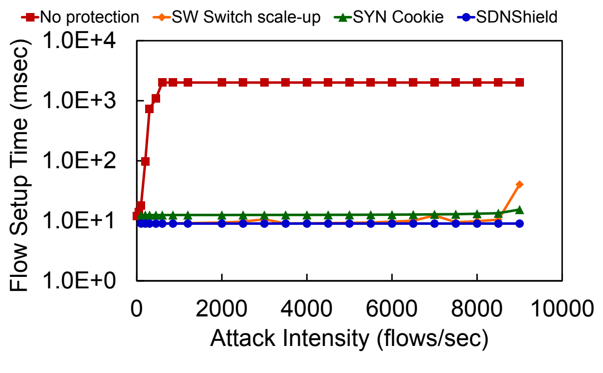
Assuming that attackers cannot accurately mimic the normal traffic footprints, on the other hand, we deliberately put the SD stage to test with a "flat" attack pattern, i.e. the forged flows are uniform randomly spread over the entire 5-tuple space. Table II shows that the SD stage can still achieve high accuracy under such an attack pattern. It is also worth noting that as the attack intensity increases, the accuracy metrics of the SD stage improve in general. This is because a more intense SDN-DDoS attack will lead to more difference between the nominal and measured profiles, making it easier for the SD stage to statistically distinguish legitimate flows from forged ones. A low FPR is significant since it allows more legitimate flows to be requested via a shorter control path shown in Figure 6, and installed onto the high-throughput physical switch route.

Performance Evaluation under Varying Attack Intensities: In this section we discuss SDNShield's performance under varying SDN-DDoS attack intensities. Here we compare four schemes: (1) ***No protection***, in which the flow arrivals are directly handled by physical switches; (2) ***Software Switch scale-up***, which is equivalent to is equivalent to Scotch [6] but without large flow migration. In this scheme, software switches elastically scale-up inbound and outbound control plane capacity, but no filtering is performed. (3) ***SYN cookie***, which is equivalent to Avant-Guard [9] implemented in AMUs. All flows are verified by the TCV stage, but no SD stage filtering is performed. (4) ***SDNShield***.

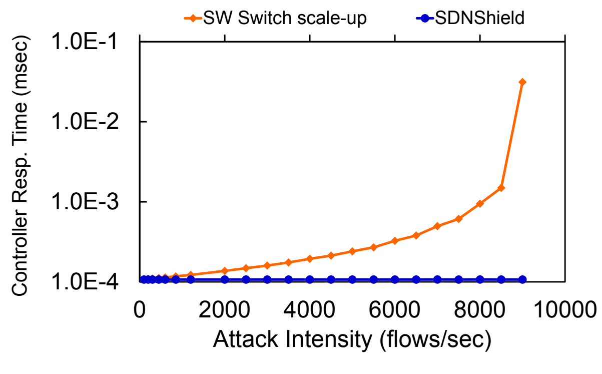
Data-driven Evaluation Results: Figure 9a shows that when attack intensity increases over 200 flows/sec, the non-protected SDN network quickly gets overwhelmed and starts to drop



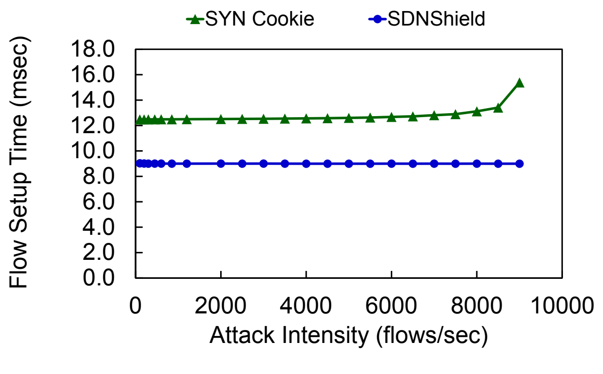
(a)



(b)



(c)



(d)

Fig. 9: Performance evaluation under varying attack intensities: (a) Legitimate flow dropping rate (b) Legitimate flow setup time (c) Controller response time (d) A closer look into flow setup time between the SYN cookie scheme and SDNShield

legitimate flows. SDNShield does not drop any legitimate flow, as the AMUs provide extra control plane capacity, and the two-stage SD+TCV filter stops forged flows from flooding the control paths.

Figure 9b shows the average setup time of those legitimate flows accepted by the network, under varying attack intensities, for the four schemes. Here we define the flow setup time as the time from the sending of SYN packet by the TCP client (Host A in Figure 5), until the handshake process completes and data packets start to transfer. As was mentioned above, the non-protected case gets quickly overwhelmed and starts to drop legitimate flows. For those legitimate flows accepted by the SDN network, the flow setup time tops out due to the long inbound queuing delay at the edge switch's OFA. SDNShield, software switch scale-up and SYN cookie are highly resilient to increasing attack intensity. However, for the software switch scale-up case, there is a significant increase in flow setup time when attack intensity is over 9000 flows/sec. This is due to that unfiltered workload causes congestion at the centralized controller, as is shown in Figure 9c.

Figure 9d takes a closer look into the comparison between SDNShield and the SYN cookie scheme. The SYN cookie scheme constantly yields longer flow setup time than SDNShield, as it requires two rounds of handshakes to establish a TCP connection. Also, the flow setup time slightly increases as the attack intensifies, due to congestion at the TCV stage module of the AMU. It is worth noting that although the TCV stage introduces considerable delay to flow setup, in SDNShield, as shown in Table II, most of the legitimate flows can be identified by the SD stage and requested via the faster path (Step 3 in Figure 6); only false-positive TCP flows (less than 1% overall as shown in Table II) are forwarded to the TCV stage. Therefore, the SDNShield schemes still enjoys lower flow setup delay in average.

V. Related Works

Security has been an increasingly important area of research of SDN. Kreutz et al. analyzed the vulnerabilities of SDN, highlighting DDoS as a critical threat to SDN [8]. There have been many works investigating robustness and scalability issues of SDN infrastructure. On the controller side, [23] used multi-thread techniques to boost up processing capacity. Difane [24] improves SDN scalability by devolving workload from centralized controller to auxiliary devices. On the switch side, Tango [8] analyzed commodity switch performance and diversity.

The threat of DDoS on SDN control plane have been recently addressed by [6] and [9]. Scotch [6] leverages the scalability of OVS-based overlay network to accommodate more flow request transactions. Avant-guard [9] uses SYN cookie/proxy technique to block out forged TCP flows from depleting SDN resources. SDNShield takes a combined defense approach, and is further inspired by previous works on traditional DDoS defense. Particularly, SDNShield bases its first stage defense mechanism on statistical filtering, such as PacketScore [20], [21], and the second stage on TCP SYN cookie/proxy techniques [11]–[13]. Our defense architecture also draws inspiration from the two-stage, coordinated defense architecture in LADS [25].

VI. Conclusion and Future Work

In this article, we propose SDNShield, a NFV-based defense framework against SDN-DDoS attacks. SDNShield leverages the scalability and ease of customization of software switches to protect the SDN network edges, and incorporates a two-stage filtering scheme to protect the centralized controller. Most of the legitimate flows are identified early in the SD stage, requested via the usual SDN control path, and installed to physical switch routes for best throughput and minimum overhead on flow setup time. As the SD stage inevitably generates false positive flows, we add the TCV stage to double-check these flows and ensure all legitimate flows are accepted by the network. Evaluation results show that SDNShield achieves high resilience to increasing attack intensities.

We shall extend our work in the following directions: (1) Prototype SDNShield on advanced processing platforms, such as *Data Plane Development Kit* (DPDK) [26], which achieves higher throughput and low latency. (2) Develop a control scheme to coordinate the virtualized AMUs across the network, for swift defense and dynamic scalability. (3) Measure extensively the performance of switch software layers and the central controller, to provide more accurate characterization and models.

References

[1] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, accessed: April 2016.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. of USENIX NSDI '10*, 2010.

[4] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula, "StEERING: A software-defined networking for inline service chaining," in *Proc. of IEEE ICNP '13*, 2013.

[5] M. Arumaithurai, J. Chen, E. Monticelli, X. Fu, and K. K. Ramakrishnan, "Exploiting ICN for Flexible Management of Software-defined Networks," in *Proc. of ACM ICN '14*, 2014.

[6] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically scaling up sdn control-plane using vswitch based overlay," in *Proc. of ACM CoNEXT '14*, 2014.

[7] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "Devoflow: Cost-effective flow management for high performance enterprise networks," in *Proc. of ACM SIGCOMM Hotnets-IX*, 2010.

[8] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proc. of ACM SIGCOMM HotSDN '13*, 2013.

[9] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," in *Proc. of ACM CCS '13*, 2013.

[10] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "Lineswitch: Efficiently managing switch flow in software-defined networking while effectively tackling dos attacks," in *Proc. of ACM ASIA CCS '15*, 2015.

[11] D. J. Bernstein, "SYN cookies," http://cr.yp.to/syncookies.html, accessed: April 2016.

[12] Cisco, Inc., "TCP intercept Feature on the ASA device," https://supportforums.cisco.com/document/12021641/tcp-intercept-feature-asa-device, accessed: April 2016.

[13] Juniper, Inc., "Understanding SYN Cookie Protection," https://www.juniper.net/techpubs/software/junos-es/junos-es92/junos-es-swconfig-security/understanding-syn-cookie-protection.html, accessed: April 2016.

[14] Pica8, Inc., "PicOS," http://www.pica8.com/products/picos, accessed: April 2016.

[15] "Open Network Linux," https://opennetlinux.org/, accessed: April 2016.

[16] Pica8, Inc., "Pica8 P-3297 Data Sheet," http://www.pica8.com/wp-content/uploads/2015/09/pica8-datasheet-48x1gbe-p3297.pdf, accessed: April 2016.

[17] AT&T, "AT&T Domain 2.0 Vision White Paper," accessed: April 2016.

[18] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *Proc. of USENIX NSDI '15*, 2015.

[19] J. Hwang, K. K. Ramakrishnan, and T. Wood, "Netvm: High performance and flexible networking using virtualization on commodity platforms," in *Proc. of USENIX NSDI '14*, 2014.

[20] Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao, "Packetscore: statistics-based overload control against distributed denial-of-service attacks," in *Proc. of IEEE INFOCOM 2004*, 2004.

[21] M. C. Chuah, W. C. Lau, Y. Kim, and H. J. Chao, "Transient performance of packetscore for blocking ddos attacks," in *Proc. of IEEE ICC 2004*, 2004.

[22] WIDE-MAWI Working Group, http://mawi.wide.ad.jp/mawi/ditl/ditl2009/, accessed: April 2016.

[23] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. of USENIX Hot-ICE '12*, 2012.

[24] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," in *Proc. of ACM SIGCOMM 2010*, 2010.

[25] V. Sekar, N. Duffield, O. Spatscheck, J. van der Merwe, and H. Zhang, "Lads: Large-scale automated ddos detection system," in *Proc. of the Annual Conference on USENIX '06 Annual Technical Conference*, 2006.

[26] Intel Inc., "Packet processing on intel architecture," http://intel.com/go/dpdk/, accessed: July 2016.