AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks

Seungwon Shin[†] Vinod Yegneswaran[‡] Phillip Porras[‡] Guofei Gu[†]
[†]Texas A&M University [‡]SRI International
{swshin,guofei}@cse.tamu.edu {vinod, porras}@csl.sri.com

ABSTRACT

Among the leading reference implementations of the Software Defined Networking (SDN) paradigm is the OpenFlow framework, which decouples the control plane into a centralized application. In this paper, we consider two aspects of OpenFlow that pose security challenges, and we propose two solutions that could address these concerns. The first challenge is the inherent communication bottleneck that arises between the data plane and the control plane, which an adversary could exploit by mounting a *control plane saturation attack* that disrupts network operations. Indeed, even well-mined adversarial models, such as scanning or denial-of-service (DoS) activity, can produce more potent impacts on OpenFlow networks than traditional networks. To address this challenge, we introduce an extension to the OpenFlow data plane called *connection migration*, which dramatically reduces the amount of data-to-control-plane interactions that arise during such attacks. The second challenge is that of enabling the control plane to expedite both detection of, and responses to, the changing flow dynamics within the data plane. For this, we introduce *actuating triggers* over the data plane's existing statistics collection services. These triggers are inserted by control layer applications to both register for asynchronous call backs, and insert conditional flow rules that are only activated when a trigger condition is detected within the data plane's statistics module. We present AVANT-GUARD, an implementation of our two data plane extensions, evaluate the performance impact, and examine its use for developing more scalable and resilient SDN security services.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Security and Protection

Keywords

Software-defined network (SDN); OpenFlow; control plane saturation attack; security and resilience

1. INTRODUCTION

As enterprise networks and data centers expand in size and complexity, they pose greater administrative challenges and demand enhanced automation in orchestrating their computer and network resources. The network research community postulates that one approach to meeting these challenges lies within the tenet of software-defined networking (SDN) [19]. By decoupling the control logic from the closed and proprietary implementations of traditional network devices, SDN enables researchers and practitioners to design new innovative network functions and protocols in a much easier, flexible, and more powerful way. The OpenFlow framework [20] is an embodiment of the SDN concept. In recent years, *OpenFlow* (OF) has steadily matured from a research idea explored in academic milieus [8, 4, 5] to the current SDN standard-bearing reference implementation with considerable momentum in industry.

We believe that OpenFlow provides new research opportunities for the network security community [26, 3, 10]. For example, OF could offer a dramatic simplification in the design and integration of complex network security applications into large networks. Unfortunately, the potential for OpenFlow to provide meaningful advancements to the state of network defense must be tempered by the recognition that OpenFlow itself introduces serious security challenges. In this paper, we explore potential solutions to two such security challenges. First, OpenFlow networks lack *scalability* between the data and control planes. This enables targeted attacks by an external entity who crafts an inbound stream of flow requests to inundate communications between the controller and switch in an adversary model that we refer to as the *control plane saturation attack*. Second, OpenFlow offers very limited support for network monitoring applications that seek a fine-grained tracking of operations at the data plane, thus making difficult the support of many security applications that require *expeditious* access to critical changes in network-traffic patterns.

Scalability Challenge. The root cause of the first challenge, scalability, lies in the operation of the OpenFlow "southbound" protocol, which separates the control plane from the data plane to enable centralized and fine-grained control of network flows. When an OpenFlow switch receives a packet belonging to a new flow for which it has no matching handling rules, it forwards the packet to its OpenFlow controller. The controller responds with one or more *flow rules* that indicate how the switch should process this flow and future flows that satisfy the rule's match criteria. Here, the centralized controller, designed to mediate these flow requests, quickly becomes a scaling bottleneck, i.e., a potential *Achilles heel* of the network during anomalous traffic bursts such as flash crowds and denial-of-service attacks. Even worse, because an external input stream ultimately drives the data-to-control plane interactions, an adversary can produce a series of unique flow requests (e.g., using a set of distributed botclients) to quickly saturate the control

CCS'13, November 4–8, 2013, Berlin, Germany.
Copyright 2013 ACM 978-1-4503-2477-9/13/11 ...$15.00.
http://dx.doi.org/10.1145/2508859.2516684 .

plane with new flow requests. At the same time, the data plane also suffers from saturation attacks because switches have limited resources to buffer (TCP/UDP) flow-initiation until the controller issues a flow rule that states how the flow shall be handled. Hence, control plane saturation also has direct implications for the data plane's operational ability. Adversary models such as DDoS and network scanning, which have been thoroughly dealt with by the security community, pose potential *new threats* to the scalability of the centralized control layer model of OpenFlow (and more broadly to the general SDN paradigm).

Responsiveness Challenge. The second challenge (i.e., responsiveness) stems from the need for expeditious access to critical data plane activity patterns. Network-monitoring applications often need to collect network statistics for tasks such as tracking flow- and network-wide packet statistics or to measure the activity of various entities communicating through the switch (e.g., to identify DoS attacks, which impact the data plane). Current SDN technologies such as OpenFlow only allow applications to explicitly *pull/poll* such information from each switch. Unfortunately, such interfaces are not sufficient for monitoring applications that require the data plane statistics in order to track and respond to malicious or degenerate operating conditions. Aggressive polling degrades data plane performance and may still not provide the latency reduction desired by security services to react to detected attacks. In addition, though security applications often require an inspection of packet contents that match some criteria, OpenFlow offers no mechanism to facilitate such policies.

We investigate the viability of our security specific extensions to OpenFlow in the context of a new system framework that we call AVANT-GUARD (AG). There are several critical issues that we address through the development of this framework. The first issue is determining the type of intelligence to be added to the data plane, i.e., what sort of statistics should we capture at the switch layer? Second, we need to develop effective techniques to report network statistics to the control plane. Third, we need to develop new mechanisms that quickly react to identified attacks. Finally, our implementation should strive to minimize changes to the OpenFlow protocol and have negligible performance impact. In essence, we recognize the design objective to keep the OpenFlow data plane as simple as possible, but the current tradeoff imposed by this design is a serious adversary model in which remote entities can halt operations of the entire network with relatively minimal traffic flows.

To this end, this paper makes the following contributions:

- We propose a strategic and focused extension to the data plane called *connection migration* that we argue yields the significant benefit of halting the threats of the saturation attack. To the best of our knowledge, *connection migration* is the first attempt in this direction to be embedded into an SDN network.

- We propose a new technique called an *actuating trigger* that addresses the responsiveness challenge by providing condition-triggered push capability in SDN devices.

- We design and implement AVANT-GUARD to integrate both *connection migration* and *actuating triggers* in a reference SDN (OpenFlow) software switch. We implement several exemplar OpenFlow security applications that demonstrate how AVANT-GUARD enhances the flexibility and robustness with which these applications can be implemented. Our performance evaluation also indicates that AVANT-GUARD incurs a very small overhead.

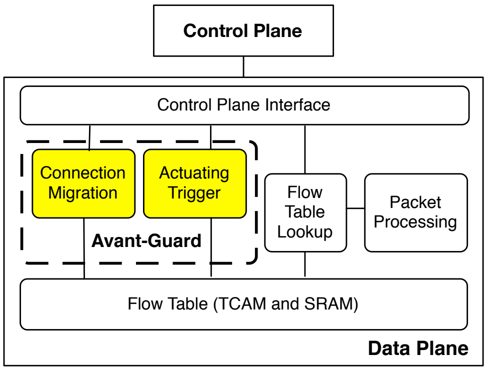


Figure 1: Conceptual architecture of AVANT-GUARD

2. PROBLEM STATEMENT

In this paper, we plan to investigate and answer the following research questions:

- Can we add (hopefully minimal) intelligence to the data plane to increase the resilience of the data-plane-to-control-plane interaction to anomalous control-plane floods?

- Is there an alternative to the existing polling strategy that may enable the control plane to acquire lower-latency knowledge of the traffic dynamics flowing through the data plane?

- Can OpenFlow applications leverage this information to detect and react more efficiently to anomalous traffic patterns?

Specifically, the key design objectives of our AVANT-GUARD (AG) framework include the following:

- Scalability and Resilience: AG must improve resilience of the OF network with minimal impact on overall scalability.

- Transparency: AG must require no changes to software running on end hosts.

- Incremental Deployment: AG must minimize changes to the OpenFlow network and enable incremental deployment. Though AG switches would require an AG-aware controller, both should be able to interoperate with other OF switches in the network.

- Minimal Collateral Impact: AG should introduce minimal additional delay to legitimate connections on the network.

3. SYSTEM DESIGN

To address the problems discussed in the previous section, we present AVANT-GUARD as a security extension to the OpenFlow data plane. In this section, we present the AVANT-GUARD design.

3.1 Overall Architecture

AVANT-GUARD extends the existing OpenFlow data plane with the addition of two new modules: 1) a *connection migration module* and 2) an *actuating trigger module*. AVANT-GUARD also slightly modifies existing data plane modules to support our target features. The conceptual diagram for AVANT-GUARD in the data plane is shown in Figure 1.

Inspired by the SYN proxy, which handles TCP connections in a middle box, we propose *connection migration* to sift failed TCP sessions at the data plane prior to any notification to the control plane. It collaborates with an *access table* and maintains TCP session information at the data plane to provide session details to the control plane. The *actuating trigger* enables collection of network status information and packet payload information more efficiently than existing data planes. Additionally, it offers *conditional flow rule activation*, i.e., the ability to activate flow rules (or actions) when some events happen.

3.2 Connection Migration

The objective of connection migration is to add intelligence to the data plane to differentiate those sources that will complete TCP connections from sources that will not. To do this, we extend the data plane to proxy the TCP handshake, and we only expose those flow requests to the control plane that complete the handshake. We present the operation of connection migrations in a stage diagram consisting of four stages: (i) classification, (ii) report, (iii) migration, and (iv) relay. Each stage and transitions between them are shown in Figure 2. When a source initiates a connection, the connection migration (CM) module engages the source in the stateless TCP handshake using SYN cookies [2]. The connection is assigned the classification stage. On handshake completion, CM notifies the control plane of the flow request, transitioning the connection to the report stage. If the control plane allows migration, CM initiates the real target host with the TCP handshake, which transitions the connection to the migration stage. Then, if the target accepts the handshake, CM notifies the control plane, and the connection enters the report stage. Finally, if the control plane allows the data plane to relay packets, CM completes the connection between the source and target, and the connection is migrated to the relay stage. Next, we examine the details of each stage.

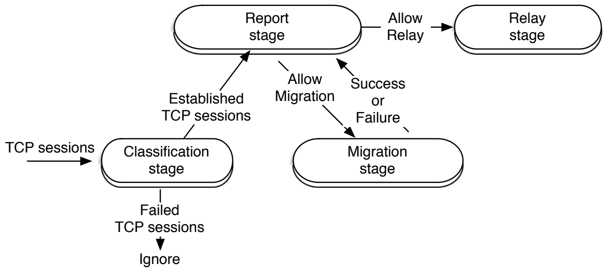


Figure 2: Stage diagram of connection migration

Classification Stage: In this stage, connection migration classifies useful TCP sessions (i.e., established TCP sessions) from connections that would result in client-side timeout (i.e., failed TCP sessions). Inspired by the SYN cookie algorithm, connection migration shields the control plan from client-side failed connection floods (e.g., which arise from DoS and reconnaissance activities), as shown in Figure 3 and 4. When the data plane receives a TCP SYN/RST/FIN packet (Figure 3), the data plane first checks if a matched flow rule exists in a flow table. If so, the data plane immediately forwards the packet. Otherwise, the data plane first updates the access table that contains information on all TCP connection attempts, by increasing the connection attempt counter for an IP address (i.e., the access table collects TCP session information). The data plane then checks whether this packet is a TCP SYN packet, and if so generates a sequence number for this packet with a hash function[1] (i.e., SYN cookie) and returns a TCP SYN/ACK packet to a peer who sends the TCP SYN packet with the generated sequence number. If the packet is not a TCP SYN packet (i.e., TCP FIN or TCP RST), it is rejected and the data plane can optionally return a TCP RST package or simply ignore the source.

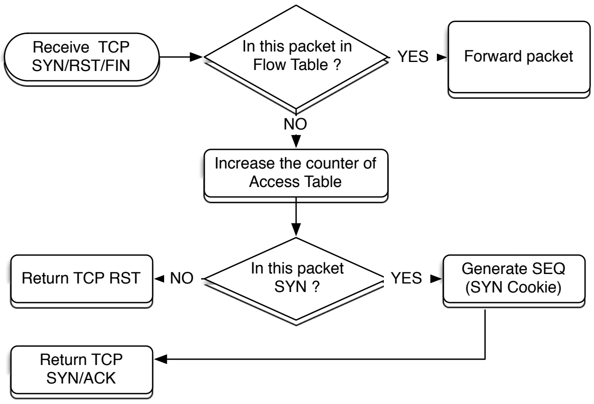


Figure 3: Flowchart for handling TCP SYN/RST/FIN packets

If the peer sends a TCP ACK packet to the data plane (Figure 4), the data plane follows the handling method shown in Figure 3. In this case, the data plane first checks the flow table to determine whether there exists a matched flow corresponding to the ACK packet. If so, the device forwards the packet. Otherwise, it validates the SYN cookie to determine whether this packet completes a TCP session or was sent unsolicited. If this ACK packet contains an appropriate SYN cookie, the TCP handshake is established. Upon completion of the handshake, the data plane reports the flow request to the control plane (i.e., step 4 in Figure 5). Otherwise, the connection request is considered an incomplete probe; a RST is sent, and the access table counters are adjusted.

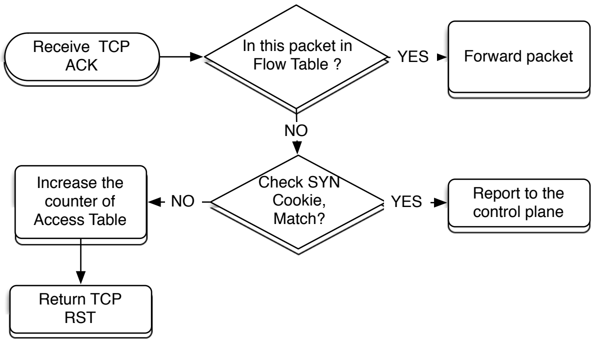

Receive TCP ACK
In this packet in Flow Table ?
YES
Forward packet
NO
Increase the counter of Access Table
NO
Check SYN Cookie, Match?
YES
Report to the control plane
Return TCP RST

Figure 4: Flowchart for handling TCP ACK packets

Report Stage: For each connection validated by the classification stage, the report stage first determines if there is an existing flow rule to handle the session. If not, the data plane reports this flow request to the control plane. The data plane extracts the header information of a flow representing the TCP session, and sends this

[1] We use 4-tuple information as inputs for this hash function.

information to the control plane with a specific command. The control plane then decides whether to allow migration of this session. If so, the connection is transitioned to the migration stage.

Migration Stage: During the migration stage, the CM module initiates a TCP connection handshake with the connection's destination host. If the host responds with a TCP SYN/ACK packet, the data plane finalizes this session by sending a TCP ACK packet to the host. The data plane also reports this information (i.e., establishment of a TCP session with a real target host) to the control plane. If the data plane fails to establish the TCP session with destination hosts (due to an unavailable host or closed port), this result will also be reported to the control plane.

Relay Stage: After the data plane successfully establishes a TCP session with a real target host, it enters the relay stage where it relays all TCP data packets between a connection source and destination as occurs during normal TCP sessions.

Example Connection Migration Scenario: To illustrate connection migration, consider the interaction shown in Figure 5. If the data plane receives a TCP SYN packet from the host A (1) and this packet does not match an existing flow rule in the device, it automatically responds with a TCP SYN/ACK packet to host A (2). Then, if host A sends a TCP ACK packet to complete this TCP session (3), the switch knows that a TCP session is successfully established, and it reports this event to the control plane. Then, the control plane decides whether to allow the connection to migrate on to the real destination host (i.e., host B). Assuming the connection is allowed, the control plane activates a flow rule with what we propose as the *Migrate* action. When a migrate action rule is received by the data plane it initiates a TCP connection to host B (6) and completes the connection (7, 8). If the migration is successful, the device notifies the control plane of this event (9). Finally, the control plane inserts a *Relay* action into the data plane, causing it to relay all packets between host A and B. At this time, the device need not add a new rule; rather, it only needs to change the action field of the existing flow rule. Hence, the rule will be changed from (A-1) to (A-2). Operations 1-3 represent the classification stage; 4-5 and 9-10 denote the reporting stages, 6-8 refer to the migration stage, and 11-12 refer to the relay stage.

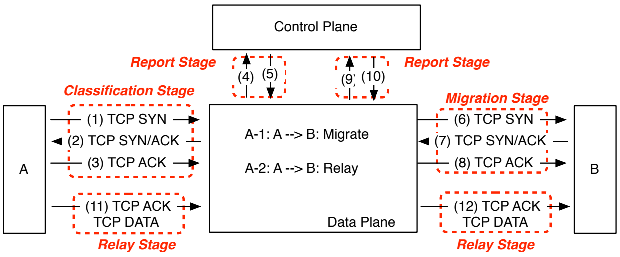


Figure 5: Example connection migration scenario

Impact on Control Plane Saturation: Connection migration offers an immediate benefit for maintaining control operations in the presence of well-known adversarial models that engage in both spoofed and non-spoofed attacks against an OpenFlow network. In the context of spoofed flooding attacks (e.g., spoofed TCP SYN floods that may saturate the control plane with bogus connection requests), all such flow requests are nullified at the classification stage. For non-spoofed connection floods (e.g., those that may arise from an aggressive scanner), connection migration converts the OpenFlow network into a *whitehole* network [9]. From the source's perspective, all probes to the ports and IP address ranges of the OpenFlow network appear to produce a TCP handshake response, hindering the source from knowing which IP and port combinations are actually alive.

In the case of the flow-rule-flooding problem in the data plane, connection migration addresses this concern through its adoption of stateless TCP handshaking with SYN cookies. Because the SYN cookie algorithm does not require any state management, a device does not need to store any flow rules for failed or malicious TCP connection attempts. It can reduce the effect of flow-rule-flooding problem. Because of this, connection migration enhances an OpenFlow network's *resilience* and *scalability* to network flooding attacks.

Collecting TCP Session Information: Based on information from access tables in the data plane, the control plane acquires two important attributes from each source that contacts the network: (i) the number of all connection attempts, captured in the access table (defined as A_1) and (ii) the number of established connections recorded within the connection migration report (defined as A_2). Analysis of the ratio of failed TCP connections of a peer (A_1 - A_2) and the number of established TCP connections (A_2) can often be used to detect various flooding and probing behavior.

3.2.1 Delayed Connection Migration

Knowledgeable adversaries may infer the use of connection migration and attempt to produce flooding packets by establishing many real TCP sessions. They can use multiple processes, threads, or many zombie PCs to generate fake TCP connections. However, for some protocols, such as HTTP, in which the client is expected to send the first data packet, we can extend connection migration to incorporate *delayed connection migration*. Here, we operate a variant of connection migration in which the key difference is that the classifying stage will delay the transition to the reporting stage until it receives the client's TCP data packet. This scenario is shown in Figure 6. As shown in Figure 6, the data plane delays the reporting time (5) until it receives more evidence (i.e., data packet) from a TCP session initiator (4).

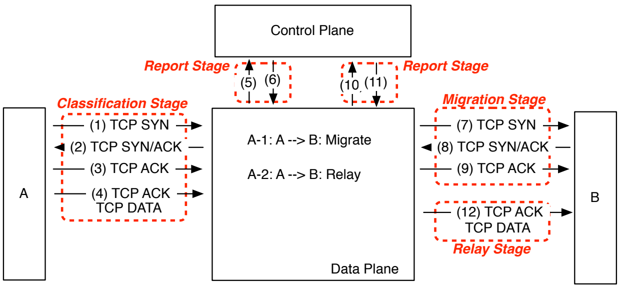


Figure 6: Example delayed connection migration scenario

3.3 Actuating Triggers

We propose to extend OpenFlow with *actuating triggers* which enable the data plane to asynchronously report network status and payload information to the control plane. In addition, actuating triggers can be used to activate a flow rule under some predefined conditions to help the control plane manage network flows without delays. The actuating trigger consists of four main operations. First, the control plane needs to define a traffic statistic condition under which notification is warranted. Second, the control plane registers this condition to the data plane. Third, the data plane checks the condition against its current locally collected packet and flow statistics to determine if the condition is satisfied. Fourth, when the data plane determines that the condition is satisfied by its current statistics, it may 1) trigger a call-back event to the control plane to indicate that the condition is met, or 2) insert a flow rule into a specified flow table.

Next, we discuss this operation in detail. The conceptual diagram of event triggering and its operation sequence are shown in Figure 7.

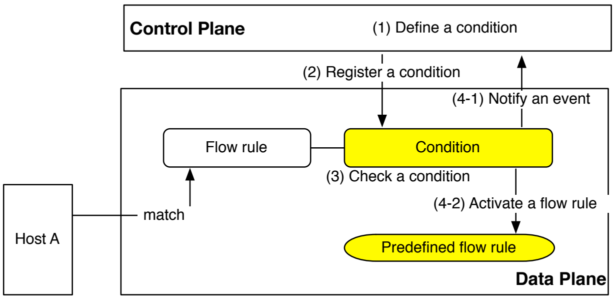


Figure 7: Example event triggering scenario

Defining a Condition: AVANT-GUARD supports three types of actuating trigger conditions: (i) payload-based, (ii) traffic-rate-based, and (iii) rule-activation. Conditions can be extended in the future. To define each type of condition, the control plane uses the following format.

```
{type: condition: pointer}
```

The type field is a 2-bit data structure representing three condition types: 00 = payload, 01 = traffic rate, and 10 = rule activation. The condition field varies for each type. For payload, the condition field is a 1-bit Boolean, indicating 1 for payload investigation required and 0 for no investigation required. For both traffic-rate and rule-activation conditions, the control plane uses a 22-bit data structure, which consists of a 2-bit, 4-bit, and 16-bit field. The 2-bit field specifies whether the control plane wishes to register for time-related metrics (e.g., packets per second (PPS) and bytes per seconds (BPS)), where 10 represents PPS, and 01 represents the BPS. 00 indicates the control plane has registered for raw counts. 8. The 4-bit data structure represents comparator options and covers five different cases. The first three are simple: (i) 0001 for *equal*, (ii) 0010 for *greater than*, and (iii) 0100 for *less than*. If the control plane wants to define compound comparators, it can simply combine (i) and (ii) (i.e., 0011) for *greater than or equal to*, and (i) and (iii) for *less than or equal to*. If the control plane sets the highest bit as 1 (i.e., 1000), the data plane needs to check PPS or BPS. The latter 16-bit structure represents the trigger value to be matched by the current statistics and enables the control plane to employ trigger ranges from 0 and 65,535.

The pointer part is used when the control plane wants to activate a predefined flow rule in which the pointer indicates where the flow rule is stored. If the data plane finds that a condition defined by the control plane is satisfied and there is a pointer attached to the condition, the data plane follows the pointer and activates the flow rule into the flow table. We extend the data plane to support a function that installs the predefined flow rules, which we can implement via an extension to the `dpctl` command.

To clarify this idea, we provide an example scenario. We assume that the control plane wants to define a conditional flow-rule insertion that will activate when a flow exceeds 32 bytes per second. To do this, the type field is set to 10 to indicate this is a network-status-based trigger. The 22-bit condition field is set as follows: the 2-bit field is set to 10 to indicate a time metric, the 4-bit comparator field is 0010 (for equality), and the 16-bit trigger value field is 32.

Condition Registration: When the control plane creates a conditional flow rule, it will be delivered to the data plane through an existing communication channel (e.g., the OpenFlow network interface). When the data plane receives this condition, it gets installed into its flow table.

Traffic Rate Monitoring: Whenever the data plane receives a packet, it updates statistical information for related fields (e.g., packet count of a flow rule). This is standard functionality in the implementation of existing OpenFlow switches which we utilized for our trigger implementation. We augment the data plane logic by adding a trigger-evaluation function which incorporates its own counter management logic within the data plane. This counter is mainly used for our network-status trigger evaluation.

In addition, we add a 16-bit data structure to store time information, which we use in our PPS and BPS calculations. These triggers are particularly useful in security applications for monitoring traffic and flow rate anomalies. PPS, BPS, and counts, can be computed on packet arrival or calculated independently based on the internal clock. The advantage of a clocked-based calculation strategy is that one can define *less-than*-based trigger evaluates (e.g., trigger when a rate falls below 10 packets per second). Packet-based calculations support *equality* and *greater-than* triggers and are computed when a trigger interval has been exceeded. For AVANT-GUARD we have implemented traffic-based rate computation.

Event Notification: When the data plane detects a signal satisfying a pre-defined condition, it notifies the control plane using a new "trigger" message extension that we added to OpenFlow.

Selective Packet Payload Delivery to the Control Plane: Packet delivery to the control plane is controlled by a flag bit (i.e., 1 bit) in the flow table of the data plane. A set flag bit implies that the control plane wants to receive packet payloads that match this flow rule condition.

The running scenario for this function is described in Figure 8. In this scenario, we assume that the control plane wants to investigate packet payloads being delivered from a host at 10.0.0.1. First, the control plane simply asks the data plane to deliver packet payloads when the source IP address of packets is 10.0.0.1 (1). Second, the data plane sets the condition field for payload of a matched flow rule (i.e., flow rule whose source IP address field is 10.0.0.1) (2). If the data plane receives a packet from a host whose IP address is 10.0.0.1 (3), it forwards the packet (with payload) to the control plane (4). In this case, we just need to add a 1-bit flag to each flow rule entry in the data plane.

Conditional Flow Rule Activation: In addition to asynchronous event notifications, we also employ triggers to introduce conditional flow rule activation. This is a powerful feature that enables a security application to predefine a course of action strategy for handling flows when the network encounters certain operating conditions that can be expressed through switch statistics. For example, when a DDoS targets a server, the control plane can find this based on the event delivered by the data plane. Then, the data plane will enforce a flow rule to stop the attack. However, this process will delay the reaction to the attack because it requires a transaction

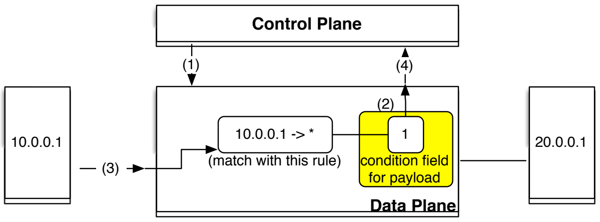


Figure 8: Scenario illustrating packet payload delivery to the control plane

between the data plane and the control plane. If the control plane already installed a flow rule for stopping this attack, the data plane need not notify the control plane to get a flow rule; instead it simply activates the installed rule.

To efficiently implement this function, we add new entries to store flow rules into the data plane. The format of these new entries is the same as a normal flow-rule entry in the data plane. The only difference is that the rules stored in these entries can only be activated by the condition field for the flow rule. The condition for activating a flow rule is the same as the condition field for status, which we explain below.

This idea is realized by adding two components to the data plane: (i) memory to store flow rules, and (ii) pointers to find installed flow rules. The pointer is an 8-bit data structure attached to a condition (i.e., the data plane manages up to 256 predefined flow rules).

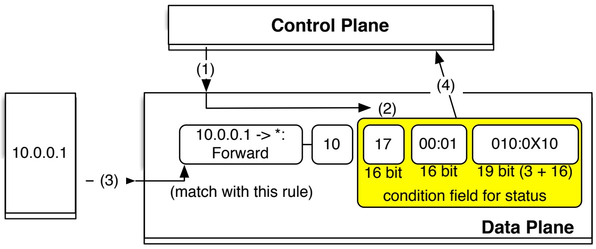


Figure 9: Scenario illustrating trigger-based notification to the control-plane

To illustrate how a flow rule is activated, we show an operational scenario in Figure 10. The condition in this Figure is the same as the condition presented in Figure 9. In this scenario, we assume that the control plane wants to block a flow if the flow generates more than 16 packets per second. In this case, the control plane can define this condition as we do above (1-3), and request that the data plane notify the control plane of this event (4). In addition, it will install a predefined flow rule (5) to BLOCK this traffic.

4. SYSTEM IMPLEMENTATION

We implemented AVANT-GUARD into the software-based OpenFlow reference switch (we call this the software OF switch) [21]. This reference implementation covers OpenFlow specification 1.0.0 [20], and it functions as the data plane. We modified the source code of this implementation to support connection migration and actuating triggers.

Specifically, we modified the packet_receive routine in the software OF switch to respond to new connection attempts with SYN/ACKs. The SYN cookie algorithm generates the SEQ number of this packet.

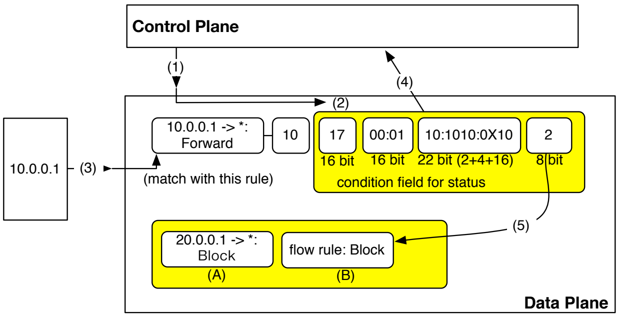


Figure 10: Scenario illustrating trigger-based activation of flow rules

If the packet-receiving routine subsequently receives a TCP ACK (i.e., matching the previously generated SYN cookie), it requests permission from the control plane to migrate the connection. Upon receiving permission, the modified OF switch will initiate a TCP connection to the real target host. To relay subsequent TCP packets through a migrated channel, we also add functions to carefully modify the corresponding ACK or SEQ numbers of each TCP packet.

We added three new data structures into the software OF switch to support actuating triggers. We modified the switch to check whenever it updates the counter for each flow (or other variables). If a counter value satisfies a condition that is defined by the control plane, the switch generates a signal back to the control plane. To implement flow rule activations, we created a data structure that can hold predefined flow rules. The data structure's format is the same as the existing flow-rule data structure (i.e., hash table and linked list) in the software OF switch.

Although most of the implementation involves extensions to the data plane (i.e., the software OF switch), minimal modification is also required to the control plane to support the aforementioned new features. Hence, we extended the source code of the POX controller [23] to support these capabilities. To support the novel functionality of AVANT-GUARD, we have added ten new OpenFlow commands as listed in Table 1. These commands are implemented in both the software OF switch and the POX controller.

4.1 Hardware Implementation Strategies

Below, we consider how our proposed OpenFlow switch extensions can be realized in a hardware-based implementation.

Traditional SDN Data Plane: First, we review the traditional SDN data plane architecture, which is illustrated in Figure 11 (A). This architecture is based on the NetFPGA implementation of the OpenFlow switch by the OpenFlow inventors [18]. Our focus here is the ASIC implementation used to conduct packet handling operations inside the switch. This implementation consists of six main modules: (i) the input arbiter, which forwards a packet to following logic; (ii) header parse, which parses a packet header; (iii) exact match lookup, which finds a flow rule (w/o wildcards) for a packet; (iv) wildcard lookup, which finds a flow rule (with wildcard) for a packet; (v) the arbiter, which decides operations of a packet (forward or drop); and (vi) the packet editor, which forwards or modifies a packet. In addition, flow rules are stored in a TCAM or SRAM (outside of the ASIC), and a counter storing statistical values for each flow rule is attached to the TCAM or SRAM.

We illustrate the operation of this hardware switch implementation using the following scenario. First, if the data plane receives

Command	*Direction*	*Explanation*
OFPFC_MIGRATE	C → D	allow connection migration
OFPFC_RELAY	C → D	allow data relay
OFPFC_REG_PAYLOAD	C → D	register payload condition
OFPFC_REG_STATUS	C → D	register network status condition
OFPFC_REG_RULE	C → D	register a new flow rule
OFPR_NEWFLOW	D → C	report a new flow
OFPR_MIGRATE_SUCCESS	D → C	report a migration result (SUCCESS)
OFPR_MIGRATE_FAIL	D → C	report a migration result (FAIL)
OFPR_PAYLOAD	D → C	deliver payload
OFPR_STATUS	D → C	report a detected event

Table 1: New OpenFlow commands implemented by AVANT-GUARD (*C* denotes control plane, and *D* denotes data plane)

a packet, the lookup component checks the TCAM or SRAM to see if a flow rule handling this packet exists. If so, it forwards the packet to the arbiter. Otherwise, it asks the control plane through an interface running on the CPU.

Implementation of Connection Migration: To implement connection migration in hardware, we need to modify three components in the data plane and add two new data structures into the data plane. The new data-plane architecture with connection migration is presented in Figure 11(B). The header parser is modified to extract TCP flags, and the arbiter is modified to force the packet editor to initiate connection migration or to reply with a TCP SYN/ACK packet. We add a connection-handler module to the packet editor. This module can initiate connection migration or answer a connection request by sending the SYN/ACK.

We also add two new data structures to support the relay stage of connection migration. Because our data plane needs to manage two TCP connections as a single connection, it should change the ACK or SEQ number of each packet. We only need to track the difference between the SEQ number of SYN packets (SYN from a connection initiator to the data plane) and the inside connection (SYN from the data plane to the destination of our migration). This difference value will be stored in an ACK/SEQ delta structure, and the number of this value is the same as the number of migrated connections.

TCP connections also come with certain TCP options, such as timestamp, and our data plane should handle this value as we do for ACK or SEQ change. To support this, we have added an optional structure into the data plane to track the TCP timestamps between external and internal connections. However, this is optional, because the data plane could also simply discard such options during TCP negotiation.

Implementation of Actuating Triggers: To implement our actuating trigger in hardware, we add two data structures for storage into the data plane. This architecture is shown in Figure 11(B). All condition fields for the actuating trigger are collectively labeled as Condition in this Figure, and they are attached to counters in the data plane. Also, predefined flow rules can be implemented by adding the same components for flow rules (TCAM and SRAM). For implementations that are cost sensitive, we may share existing TCAM or SRAM storage for these flow rules (not denoted in the Figure).

Off-ASIC Implementation of AVANT-GUARD: The architecture described above involves adding new components to the ASIC in the data plane which is both costly and complex. Here, we are inspired by Mogul's research [11], which suggests we may move some components out of the ASIC, and could potentially leverage the switch CPU and DRAM to implement certain functionality. In this case, we cannot avoid modifying existing components (i.e., modified header parser), but we can offload some storage requirements from the ASIC by moving data structures into DRAM. This architecture is presented in Figure 11(C). As shown in this figure, we place all storage into DRAM. Logic that resides in the ASIC could access DRAM content through the CPU (e.g., via a PCI interface). This approach trades off some performance for simplified development cost.

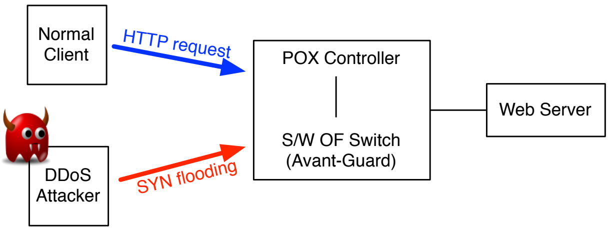


Figure 12: Environment for network saturation attack scenario

5. EVALUATION

In this section, we present how SDN security researchers and practitioners can leverage the benefit of AVANT-GUARD to develop simpler and stronger network security applications.

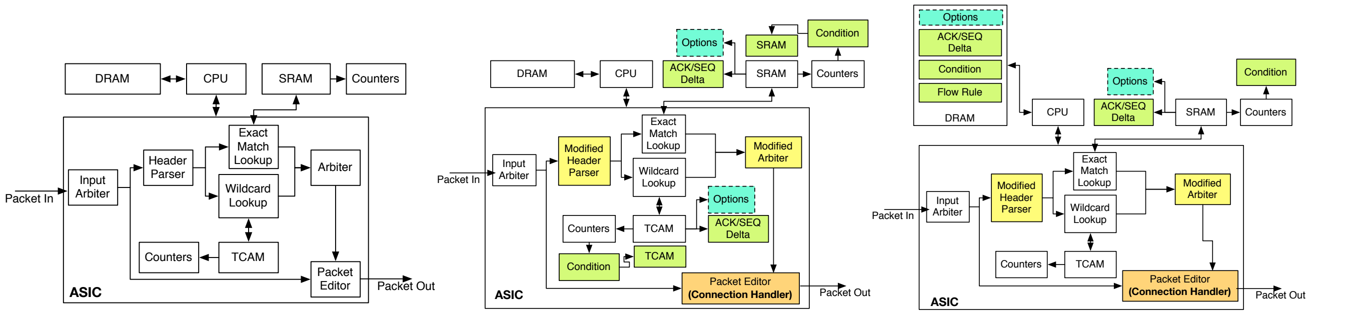
5.1 AVANT-GUARD Use Cases

We first describe an example use case for a security application and then compare two scenarios: (i) implementing the security application with existing OpenFlow technology and (ii) implementing the same function with AVANT-GUARD. We select three common network threats for comparison: (i) network saturation attack, (ii) network scanning attack, and (iii) network intrusion attack. For each case, we employ the software OF reference switch implementation [21] for our data plane, and AVANT-GUARD has been implemented into this reference implementation. We turn on or off the functions of AVANT-GUARD to compare the functions of each case. For the control plane, we use a modified POX controller [23] for both switches. The host running this S/W OF switch (w AVANT-GUARD or w/o AVANT-GUARD) was configured with an Intel Core-i5 CPU and 8 GB of memory.

5.1.1 Network Saturation Attack

Example Scenario: The test environment for this experiment is shown in Figure 12. It includes an OpenFlow switch (i.e., the data plane) in which AVANT-GUARD has been implemented; a POX network controller; a server that hosts a web service; a normal client that contacts the server with HTTP requests; and an attacker who performs a TCP SYN flood attack.

In this scenario, we measure the response time (i.e., the time it takes a normal client to fetch a page of data from the remote



(A) **Traditional SDN switch**

(B) **AVANT-GUARD SDN switch**

(C) **Off-ASIC implementation**

Figure 11: Hardware architectural designs of (A) Traditional SDN switch (B) SDN switch with connection migration and actuating triggers and (C) SDN switch with connection migration and actuating triggers off-ASIC

Item	*Case*	*Response Time*	*Overhead*
Original	w/o DDoS	0.3917 s	-
	w/ DDoS	∞	∞
AVANT-GUARD	w/o DDoS	0.3990 s	1.86 %
	w/ DDoS	0.4001 s	2.1 %

Table 2: Average response time of each test case (Overhead means the percentage of additional response time compared with the Original - w/o DDoS case)

server under two situations: with and without background TCP SYN floods in this OpenFlow network). The attacker generates 1,000 connection attempts per second to the server, and we repeat this over 500 seconds to measure the average response time.

The test result showing the average response time is summarized in Table 2. The normal client can retrieve the web page in 0.4 seconds, but it does not get any response during a background TCP SYN flood attack due to the effect of control/data plane saturation mentioned earlier. However, AVANT-GUARD can effectively defend the network from this attack, enabling the normal client to retrieve the webpage without any problem, because our data plane automatically and transparently classifies and removes the malicious TCP connection attempts. Our system introduces only a negligible delay overhead (around 2.1%) for the normal client, even during severe saturation attacks.

We also measure the overhead of connection migration on normal TCP connections during normal network operations (i.e., without attacks) using the same experimental setup shown in Figure 12. From Table 2, we can see that the overhead caused by connection migration on normal TCP connections is minimal (1.86 %).

To further show the effect of saturation attacks on normal traffic in detail, we vary the packet-sending rate of the network saturation attack from 0 to 800 per second, and we send the requests from 10 benign clients to a target web server at the same time. The test results are shown in Figure 13, and we can easily observe that requests from benign clients are hardly delivered to the web server when the network saturation attack happens using the unmodified OpenFlow switch (nearly 0% when the flooding attack sends more than 100 packets per second). However, with AVANT-GUARD, all requests from benign clients are delivered to the web server, even while the network is under a severe network saturation attack.

Implementation Comparison: To detect TCP SYN flood attacks with an OpenFlow application, the application typically must be aware of the TCP session information (e.g., whether or not a TCP connection is successful). However, this session management will cause control flow saturation issues that we discussed earlier.

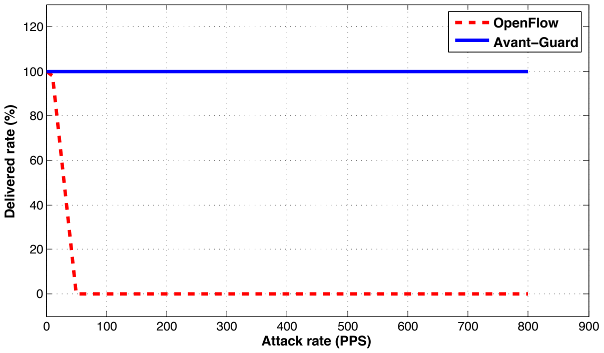


Figure 13: Percentage of successfully delivered packets to the web server from benign clients

That is the data plane will receive many TCP SYN packets that it will report to the control plane, which also receives the same number of requests from the attack. Currently, OpenFlow does not provide any way to reduce this effect.

With an AVANT-GUARD application, because the data plane automatically responds to all TCP SYN packets without high overhead, the data plane need not handle (i.e., forward or drop packets) all attack packets. In addition, the control plane only receives the requests for a successfully established TCP connection. Thus, the control plane does not suffer from network saturation attacks, and the application can easily detect such attacks. In this case, AVANT-GUARD makes the control plane and the SDN network more resilient and scalable.

5.1.2 Network Scanning Attack

Example Scenario: We test how our system defends a network from a network scan attack, and the test environment is shown in Figure 14. In this test, we use Nmap [24] to vertically scans all network ports of a file server (10.0.0.2) that only opens network port 10000.

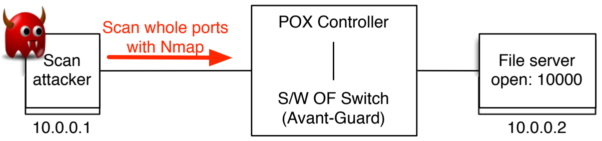
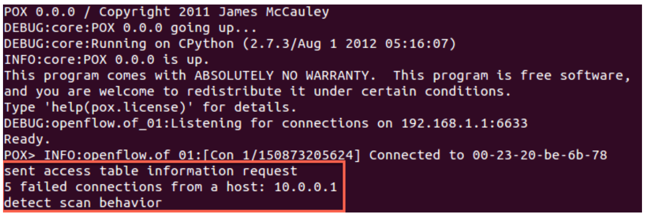


Figure 14: Environment for network scan attack scenario

If we employ Avant-Guard, the data plane automatically maintains the information on the TCP connection attempts in the access table and reports session information to the control plane, which can easily detect scan attempts by applying a simple threshold-based scan-detection algorithm. Here, we write a simple security application for detection of a network scan attack that regards a remote host as a scanner if it initiates five failed TCP connection attempts. This application only needs to ask the data plane to report the information on the TCP connection attempts; it does not itself need to maintain TCP sessions. The detection result is marked with a red rectangle in Figure 15.

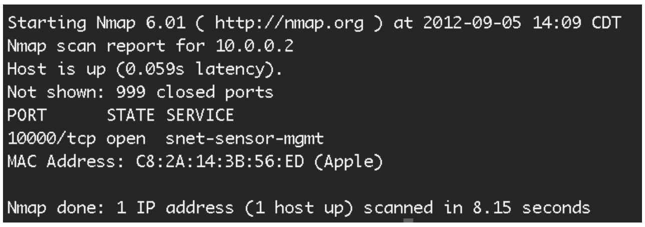


```
POX 0.0.0 / Copyright 2011 James McCauley
DEBUG:core:POX 0.0.0 going up...
DEBUG:core:Running on CPython (2.7.3/Aug 1 2012 05:16:07)
INFO:core:POX 0.0.0 is up.
This program comes with ABSOLUTELY NO WARRANTY.  This program is free software,
and you are welcome to redistribute it under certain conditions.
Type 'help(pox.license)' for details.
DEBUG:openflow.of_01:Listening for connections on 192.168.1.1:6633
Ready.
POX> INFO:openflow.of_01:[Con 1/150873205624] Connected to 00-23-20-be-6b-78
sent access table information request
5 failed connections from a host: 10.0.0.1
detect scan behavior
```

Figure 15: Network scan-detection result

Implementation Comparison: To detect a TCP scanning attack with an OpenFlow application, we need to check whether each TCP session is successful at the application layer. However, this check requires that the application manages each TCP flow making it vulnerable to control flow saturation attacks. If we implement the same application with Avant-Guard, it only needs to periodically read the access table to collect TCP session information.

In addition to this, we can implement the whitehole function with Avant-Guard easily. The whitehole function provided by our system can be easily observed by looking at Nmap scan results. In the absence of our approach, Nmap can successfully scan the file server, and finds that network port 10000 is open, as shown in Figure 16. Figure 17 shows the scan results of Nmap when applying our system (the network environment is the same as in Figure 14). Although the file server only opens port 10000, Nmap thinks that *all* network ports are open.



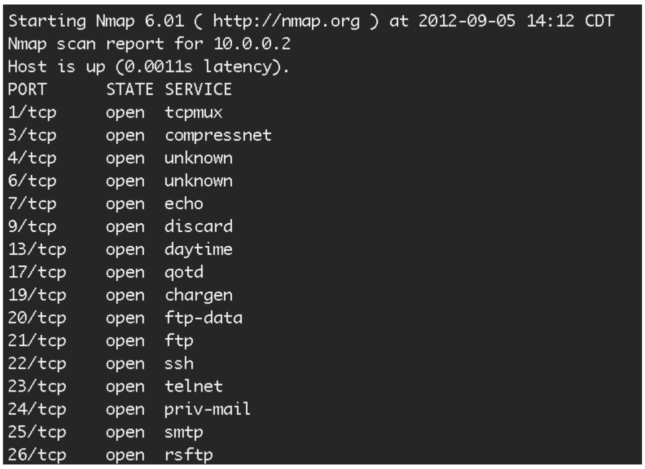
```
Starting Nmap 6.01 ( http://nmap.org ) at 2012-09-05 14:09 CDT
Nmap scan report for 10.0.0.2
Host is up (0.059s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
10000/tcp open  snet-sensor-mgmt
MAC Address: C8:2A:14:3B:56:ED (Apple)

Nmap done: 1 IP address (1 host up) scanned in 8.15 seconds
```

Figure 16: Nmap scan result without Avant-Guard

5.1.3 *Network Intrusion Attack*

Example Scenario: We set up an attack scenario as shown in Figure 18, and in this case, an attacker (10.0.0.1) sends an RPC buffer overflow attack to another host in the network (10.0.0.2). Here we assume two things: (*i*) the control plane already requested the data plane to deliver packet payloads delivered to 10.0.0.2 and (*ii*) a security application has a signature for the attack. In this test scenario, the application uses snort rules to detect malicious payloads. The result is shown in Figure 19, where we find that the security application accurately detects the attack (red rectangle in Figure).



```
Starting Nmap 6.01 ( http://nmap.org ) at 2012-09-05 14:12 CDT
Nmap scan report for 10.0.0.2
Host is up (0.0011s latency).
PORT      STATE SERVICE
1/tcp     open  tcpmux
3/tcp     open  compressnet
4/tcp     open  unknown
6/tcp     open  unknown
7/tcp     open  echo
9/tcp     open  discard
13/tcp    open  daytime
17/tcp    open  qotd
19/tcp    open  chargen
20/tcp    open  ftp-data
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
24/tcp    open  priv-mail
25/tcp    open  smtp
26/tcp    open  rsftp
```

Figure 17: Nmap scan result with Avant-Guard - Whitehole

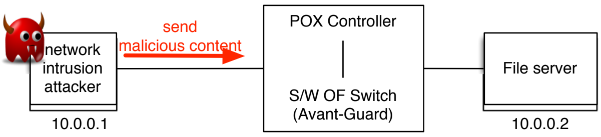


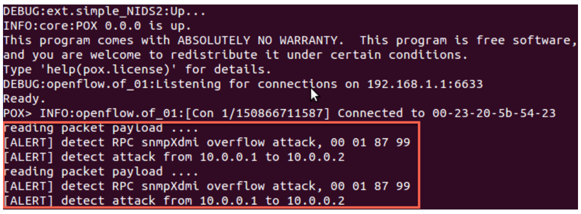
Figure 18: Network intrusion attack scenario

Implementation Comparison: An attacker can send malicious contents (e.g., exploits) to infect a target victim. To date, many approaches for detecting network intrusions have been proposed, and most of them rely on deep packet inspection to identify known attack patterns. However, an OpenFlow control plane (and applications) cannot see all network packet payloads. This is because OpenFlow was originally designed to handle mostly layer 2/3 network traffic. The data plane only reports network header information to the control plane, and if there is a matching flow rule in the flow table, the data plane does not even report header information to the control plane.

The actuating trigger module of Avant-Guard provides a new capability that can deliver a packet payload to the control plane (i.e., condition for packet payload). In this case, the Avant-Guard application simply defines a condition involving header fields (e.g., source or destination IP) that it wants to investigate and then forwards these criteria to the switch. The switch will report every packet payload that matches the condition to the application.

5.2 Overhead Measurement

When we measure the performance overhead of Avant-Guard, we use the same test environment as in Figure 12.



```
DEBUG:ext.simple_NIDS2:Up...
INFO:core:POX 0.0.0 is up.
This program comes with ABSOLUTELY NO WARRANTY.  This program is free software,
and you are welcome to redistribute it under certain conditions.
Type 'help(pox.license)' for details.
DEBUG:openflow.of_01:Listening for connections on 192.168.1.1:6633
Ready.
POX> INFO:openflow.of_01:[Con 1/150866711587] Connected to 00-23-20-5b-54-23
reading packet payload ....
[ALERT] detect RPC snmpXdmi overflow attack, 00 01 87 99
[ALERT] detect attack from 10.0.0.1 to 10.0.0.2
reading packet payload ....
[ALERT] detect RPC snmpXdmi overflow attack, 00 01 87 99
[ALERT] detect attack from 10.0.0.1 to 10.0.0.2
```

Figure 19: Network intrusion detection based on simple payload inspection

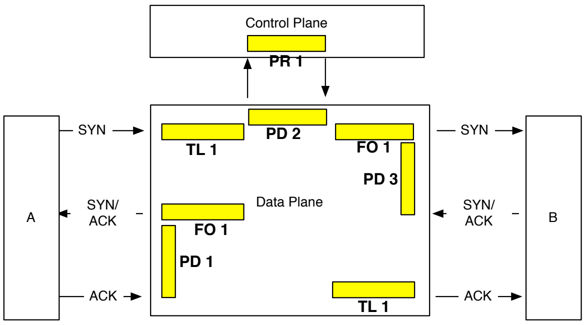


Figure 20: Breakdown of connection establishment delays in OpenFlow

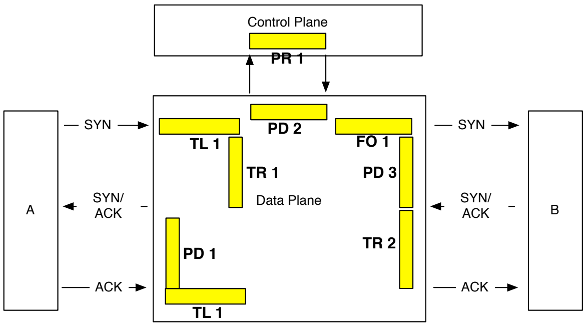


Figure 21: Breakdown of connection establishment delays in AVANT-GUARD

5.2.1 Connection Migration

To understand the overhead of connection migration at the micro level, we analyze the internal operations for establishing a TCP session in two cases: (*i*) using the software OF switch reference implementation and (*ii*) using the AVANT-GUARD extension. Figures 20 and 21 illustrate the breakdown of connection-establishment delays in the two systems, respectively.

In the case of a typical OF switch, the delay in establishing a new TCP session for which there is no flow rule can be broken down into seven components that start with the switch receiving a TCP SYN packets and end with the transmission of the ACK packet to the target server: (*i*) lookup a flow table and forward (TL1); (*ii*) ask the control plane for a flow rule and receive the rule (PD2) - (processing time in the control plane (PR1) is not included); (*iii*) insert a flow rule and forward (FO1); (*iv*) receive a SYN/ACK packet (PD3); (*v*) forward a packet based on the flow rule (FO1); (*vi*) receive an ACK packet (PD1); and (*vii*) lookup a table and forward (TL1).

In the case of AVANT-GUARD, the breakdown is a little different because the data plane automatically responds with SYN/ACK packet. With AVANT-GUARD the eight operations include: (*i*) lookup a flow table and forward (TL1); (*ii*) generate a SYN/ACK packet (TR1); (*iii*) receive an ACK packet (PD1); (*iv*) lookup the flow table (TL1); (*v*) ask the control plane to get a permission for migration and receive the rule for migration (PD2) - (processing time in the control plane (PR1) is not included); (*vi*) forward a SYN packet to a target host (FO1); (*vii*) receive a SYN/ACK packet (PD3); and (*viii*) generate an ACK packet and send it (TR2).

We summarize the delay breakdown in the two cases as follows:

- OpenFlow case = TL1 + PD2 + FO1 + PD3 + FO1 + PD1 + TL1
- AVANT-GUARD case = TL1 + TR1 + PD1 + TL1 + PD2 + FO1 + PD3 + TR2

We instrumented the software switch to measure the respective delay of each component and illustrate them in Figures 22 and 23. To get these results, we initiated many TCP connection attempts to a switch and then used the average values from our measurements. The average connection establishment delay in the OF software switch case is 1608.6 us; for AVANT-GUARD the average is 1618.74 us. Thus, the overhead of connection migration is 0.626%, which is very small.

However, the delays here are dominated by the propagation delay. If we remove PD1, PD2, and PD3 from the above measurements, the original S/W OF switch incurs a composite delay of 32.4 us, and AVANT-GUARD incurs a delay of 42.54 us. In this case, the overhead is 23.84%, which is somewhat more substantial, but still not prohibitive.

5.2.2 Actuating Triggers

To estimate the overhead of actuating triggers, we measure the time to check each condition (e.g., traffic-rate based condition) and to activate a flow rule. Our results are summarized in Table 3. For the traffic-rate based condition, we simply define a condition that checks if the PPS of a flow is greater than 100. We see that the overhead for each condition is relatively small (even nearly 0 in the case of payload-based condition). In comparison with the elapsed time for connection migration, condition checks only involve around 1.6% of overhead.

Item	*time*
Traffic-rate based condition	0.322 us
Payload-based condition	$\approx$ 0 us
Rule activation	1.697 us

Table 3: Time for checking each condition

The time for activating a flow rule based on a condition includes the time for checking the traffic-rate based condition and the time for looking up a flow table. In our measurement, it is 1.697 us. This time is significantly less than typical propagation delay between the data plane and the control plane (i.e., PD2). In our setup, the time for PD2 is 459.81 us, and the time for activating a flow rule is just 0.36% of PD2. Thus, we can say that our approach significantly reduces the time for threat reaction/mitigation. For example, if a network is subject to SYN flood attacks at a rate of 1,000,000 per second (1 packet per us), then this network will receive at least 460 packets before stopping the attack. However, with AVANT-GUARD, only two additional packets would be received after the data plane reacts to the attack.

6. RELATED WORK

Our work is inspired by several parallel studies that explore security threats in SDNs [1, 14, 28] and attempt to deliver innovative security applications with SDN technology. Mehdi et al. developed SDN applications for detecting network scan attacks [17]. Jafarian et al. suggested a random host mutation approach to protecting a network from scanning attacks[10]. Popa et al. designed a new security application for a cloud network [22]. Braga et al. created a DDoS detection application using SDN [3]. Shin et al. proposed a new security-monitoring-as-a-service framework for a cloud network [25]. Unlike our approach which fundamentally alters flow management at SDN switch level, these studies focus on creating specific high-level SDN applications to provide improved security services. We believe that all these applications can benefit

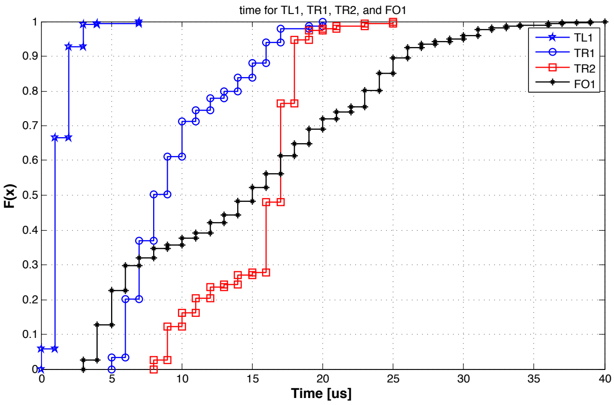


Figure 22: CDF of measured component delays (part 1)

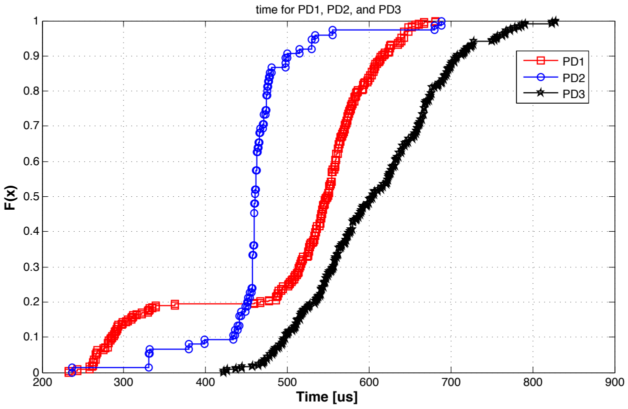


Figure 23: CDF of measured component delays (part 2)

from AVANT-GUARD. For example, the applications for detecting scan attacks in [17, 10] no longer need to manage TCP session by themselves because they can simply leverage AVANT-GUARD. FRESCO [26] is a a new framework designed to help accelerate the development of SDN security applications. We believe such frameworks can benefit from the resilience and trigger capabilities provided by AVANT-GUARD.

AVANT-GUARD's connection migration function is inspired by the SYN cookie, and there are many commercial products employing the SYN cookie idea to defeat TCP SYN flooding attacks (e.g., Cisco Guard [6] and Juniper Junos [12]). In addition, Mahimka et al. proposed a middle-box solution that can defeat network flooding attacks [16]. To our knowledge, we are the first to apply SYN cookies and connection migration in an SDN network.

Researchers have also proposed new architectures to improve SDN performance by reducing the communication between the switch and controller. Onix [13], Maestro [30], and Hyperflow [27] attempt to build more scalable SDN networks using distributed control planes. While we share a common goal in improving the scalability of SDN networks, we differ in the specific techniques that we propose to achieve this goal and in our emphasis on security. Mogul et al. proposed a new data plane architecture to reduce memory space by moving counters to the control plane [11]. Lu et al. [15] devised a new SDN architecture that handles a part of packet processing in the CPU. DIFANE [29] proposed a new solution that seeks to reduce switch-to-controller traffic by keeping all traffic in the data plane by selectively directing packets through intermediate switches that store the necessary rules. DevoFlow [7] seeks to improve the visibility and statistics gathering capabilities of OpenFlow and suggests a new data plane architecture to manage flow rules efficiently. However, the goals and main approach of DevoFlow and DIFANE are very different from our work in that these studies are focused on improving performance under normal network conditions. In contrast to AVANT-GUARD introduces new techniques to improve network resilience under attacks. The concept of *actuating triggers* is similar to event-triggers described in DevoFlow, with the key difference that our triggers result in the dynamic insertion of a flow rule. In contrast, DevoFlow event-triggers simply result in a call back to the controller.

7. LIMITATIONS AND DISCUSSION

We now discuss some limitations in our work. First, the connection migration component of AVANT-GUARD primarily improves resilience against TCP SYN flood and network scanning attacks. Thus, it might not help application developers who want to defend against application layer DoS attacks or attacks based on UDP or ICMP protocols. However, delayed connection migration and actuating triggers could be selectively used to mitigate the impact of such attacks. Further, most well-known network services are based on TCP and most common network attacks are against TCP-based services. Thus, AVANT-GUARD would benefit the majority of network systems and applications. Extending AVANT-GUARD to better support more sophisticated attacks and non-TCP flows is future work.

Second, the use of connection migration imposes a small but measurable delay to normal network connections. This represents a very reasonable trade-off between security and performance. To reduce unnecessary overhead, we envision that certain networks may turn off connection migration by default and initiate it only under duress. This policy could be expressed as an actuating trigger. In addition, it could be combined with whitelists or dynamic reputation schemes to improve the quality of service to legitimate users during attacks. In our future work, we hope to more thoroughly investigate such trade-offs that occur when using connection migration and design better guidelines for different scenarios.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we propose AVANT-GUARD, a new framework to advance the security and resilience of OpenFlow networks with greater involvement from the data-plane layer. The goal of AVANT-GUARD is to make SDN security applications more scalable and responsive to dynamic network threats. A key challenge, which we address, is the inherent bottleneck introduced by the interface between the control plane and the data plane that knowledgeable adversaries can exploit. Connection migration enables the data plane to shield the control plane from such saturation attacks. The second challenge is the issue of responsiveness. SDN security applications need expeditious access to network statistics from the data plane as a method for quickly responding to network threats. To address this, we introduce actuating triggers that automatically insert flow rules when the network is under duress. We present a software implementation of AVANT-GUARD, and demonstrate that the overhead imposed by the AVANT-GUARD security extensions is minimal, with connection delay increases of much less than a 1% overhead, while providing resilience to an important adversarial model that may hinder SDN adoption.

Acknowledgements. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and the United States Air Force under Contract No. FA8750-11-C-0249 and by the National Science Foundation (NSF) under Grant no. CNS-0954096. All opinions, findings and conclusions or recommendations expressed herein are those of the author(s) and do not necessarily reflect the views of DARPA, the United States Air Force or NSF. It is approved for Public Release, Distribution Unlimited.

9. REFERENCES

[1] K. Benton, L. J. Camp, and C. Small. Openflow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN)*, 2013.

[2] D. J. Bernstein. SYN Cookies. `http://cr.yp.to/syncookies.html`.

[3] R. S. Braga, E. Mota, and A. Passito. Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. In *Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN)*, 2010.

[4] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. In *Proceedings of ACM SIGCOMM*, 2007.

[5] M. Casado, T. Garfinkel, M. Freedman, A. Akella, D. Boneh, N. McKeowon, and S. Shenker. SANE: A Protection Architecture for Enterprise Networks. In *Proceedings of the Usenix Security Symposium*, August 2006.

[6] Cisco. Cisco Guard Anomaly Detector. `http://www.cisco.com/en/US/prod/collateral/modules/ps2706/ps6235/product_data_sheet0900aecd80220a7c.html`.

[7] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: Scaling Flow Management for High-Performance Networks. In *Proceedings of ACM SIGCOMM*, 2011.

[8] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. In *Proceedings of ACM Computer Communications Review*, 2005.

[9] G. Gu, Z. Chen, P. Porras, and W. Lee. Misleading and Defeating Importance-Scanning Malware Propagation. In *Proceedings of International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2007.

[10] J. H. Jafarian, E. Al-Shaer, and Q. Duan. Openflow Random Host Mutation: Transparent Moving Target Defense using Software Defined Networking. In *Proceedings of the 1st ACM Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[11] P. C. Jeffrey C. Mogul. Hey, You Darned Counters! Get Off My ASIC! In *Proceedings of the 1st ACM Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[12] Juniper. Junos Security Configuration. `http://www.juniper.net/us/en/products-services/nos/junos/`.

[13] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A Distributed Control Platform for Large-scale Production Networks. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[14] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards Secure and Dependable Software-Defined Networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013.

[15] G. Lu, R. Miao, Y. Xiong, and C. Guo. Using CPU as a Traffic Co-processing Unit in Commodity Switches. In *Proceedings of the 1st ACM Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012.

[16] A. Mahimka, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang. dFence: Transparent Network-based Denial of Service Mitigation. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.

[17] S. Mehdi, J. Khalid, and S. Khayam. Revisiting Traffic Anomaly Detection Using Software Defined Networking. In *Recent Advances in Intrusion Detection (RAID)*, 2011.

[18] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown. Implementing an OpenFlow switch on the NetFPGA Platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2008.

[19] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. Technical report, 2012. `https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf`.

[20] OpenFlow. OpenFlow Switch Specification version 1.0.0. Technical report, 2010. `http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf`.

[21] OpenFlow.org. OpenFlow Switching Reference System. `http://www.openflow.org/wp/downloads/`.

[22] L. Popa, M. Yu, S. Y. Ko, I. Stoica, and S. Ratnasamy. CloudPolice: Taking Access Control out of the Network. In *Proceedings of the 9th ACM Workshop on Hot Topics in Networks (HotNets)*, 2010.

[23] POX. Python Network Controller. `http://www.noxrepo.org/pox/about-pox/`.

[24] N. Security. `http://nmap.org/`.

[25] S. Shin and G. Gu. CloudWatcher: Network Security Monitoring Using OpenFlow in Dynamic Cloud Networks (or: How to Provide Security Monitoring as a Service in Clouds?). In *Proceedings of the 7th Workshop on NPSec*, 2012.

[26] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson. FRESCO: Modular Composable Security Services for Software-Defined Networks. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS)*, 2013.

[27] A. Tootoonchian and Y. Ganjali. HyperFlow: A Distributed Control Plane for OpenFlow. In *Proceedings of the Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN)*, 2010.

[28] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang. Towards a secure controller platform for openflow applications. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN)*, 2013.

[29] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable Flow-based Networking with DIFANE. In *Proceedings of ACM SIGCOMM*, 2010.

[30] T. S. E. N. Zheng Cai, Alan L. Cox. Maestro: Balancing Fairness, Latency and Throughput in the OpenFlow Control Plane. *Rice University Technical Report TR11-07*.