Attacking Software-Defined Networks: A First Feasibility Study

Seungwon Shin and Guofei Gu
{swshin, guofei}@cse.tamu.edu

ABSTRACT

Software Defined Networking (SDN) is an emerging technology that attracts significant attention from both industry and academia recently. By decoupling the control logic from the closed and proprietary implementations of traditional network devices, it enables researchers and practitioners to design new innovative network functions/protocols in a much easy, flexible, and powerful way. However, till today, SDN is still not well investigated by the security community. In this paper, for the first time we show a new attack to fingerprint SDN networks and further launch efficient resource consumption attacks. This attack demonstrates that SDN brings new security issues that may not be ignored. We provide the first feasibility study of such attack and further discuss possible defense techniques and insights.

1. INTRODUCTION

Very recently, Software Defined Networking (SDN) has quickly emerged as a new promising technology for future networks. With the separation of control plane from data plane thus enabling the easy addition of new, creative, powerful network functions/protocols, SDN has attracted significant attention from both academia and industry. In academia, since the publication of OpenFlow [4], which is a key component to realize the SDN concept, many research ideas based on SDN/OpenFlow have been proposed (and still go on) (e.g., [10] [3]). In industry, SDN is widely considered as the new paradigm for future networks, and many companies are deploying or plan to deploy such technology in order to strengthen their network architectures, reduce operational cost, and enable new network applications/functions.

While SDN is promising, one important aspect is seriously missing from the community: security. As a very new technology, there is very little research in investigating the security of SDN, e.g., whether SDN brings new security issues. As one of the first step towards a better understanding of the security of SDN, it is very important to investigate its attack/threat surface.

In this paper, we demonstrate an effective and efficient attack against software-defined networks with the knowledge of some basic characteristics of the SDN technology. Essentially, since the control plane is separated from the data plane in a SDN network, the data plane will typically ask the control plane to obtain flow rules when the data plane sees new network packets that it does not know how to handle. By exploiting this key property, our proposed attack can first fingerprint whether a given network uses SDN/OpenFlow switches and then generate specifically crafted flow requests from the data plane to the control plane. This has two effects: (i) it can make the (logically centralized single-point) control plane hard to handle all requests (i.e., **control plane resource consumption or DoS attack**); (ii) the generated fake flow requests can produce many useless flow rules that need to be held by the data plane, thus making the data plane hard to store flow rules for normal network flows (**data plane resource consumption or DoS attack**). To demonstrate the feasibility of such attack, we create a new SDN network scanning prototype tool (named as HFC SCANNER) to remotely fingerprint networks that deploy SDN, and this method can be easily operated by modifying existing network scanning tools (e.g., ICMP scanning and TCP SYN scanning).

In short, our contributions can be summarized as follows:

- We propose a new fingerprinting attack scenario against SDN networks - to the best of our knowledge, this is the first work talking about the network attack for SDN networks.
- We verify the feasibility of the proposed attack with real-time data - we show that the proposed attack can easily fingerprint a SDN network and further consume its resources (DoS attack).
- We further discuss possible defense techniques and insights to mitigate the proposed attack.

2. MOTIVATING EXAMPLE

In a SDN environment, the control plane can dynamically enforce flow rules when the data plane requires (i.e., reactive mode), and it enables us to control the network efficiently. However, this kind of reactive mode control can cause serious problem when there are too many requests from the data plane to the control plane - **we call this resource consumption attack**. To describe this problem more clearly, we show an example case when a network administrator uses an OpenFlow controller (for the control plane) and an OpenFlow switch (for the data plane) to control his network.

When an OpenFlow application enforces a flow rule into a switch, it needs to specify a condition field (also called the match field) to describe a network flow that the application wants to control. This condition field consists of 15 elements [8], and an OpenFlow application can determine the level for controlling network flows by setting each condition field. For example, if an application specifies all elements in a condition field, it will control network flows with fine-grained level. Likewise, an application can set wildcards (i.e., don't care bits) to control network flows with coarse-grained level.

Usually, legacy network routers (or switches) use some flow rules with wildcards to control network flows (i.e., coarse-grained level), and it is very natural because their main goal is to simply deliver network packets to destinations. It is also applicable when an OpenFlow application conducts routing functions. Beyond that, we also want to design interesting and complicated network functions such as load balancing [14]. In this case, an OpenFlow application will control network flows in more fine-grained level (e.g., 4-tuples: source and destination IP address, and source and destination port), and this could cause flooding attacks targeting OpenFlow switches and the controller.

To clearly illustrate these cases, we assume that an OpenFlow application controls network flows by enforcing flow rules defining 4-tuples to perform load balancing. We also assume that this enterprise network usually receives 1,000 requests per second, and it is capable to handle them with the available resource. However, if an attacker uses 10 bots to send fake TCP SYN requests to this network and each bot creates 100 processes to do this job, then, this attacker can easily send much more connection requests that can be handled by the OpenFlow application (whenever an OpenFlow switch receives a packet that does not match with any flow rules in a flow table, it will ask the controller). It also happens when even one host sends connection trials to different ports of a host (i.e., a kind of vertical scanning), and this scenario is illustrated in Figure 1. If the 10.0.0.1 host sends connection requests to network ports from 1 to 10,000 of 20.0.0.1 host, it will cause 10,000 flow rule requests to the controller, and it will require 10,000 flow table entries in the switch.

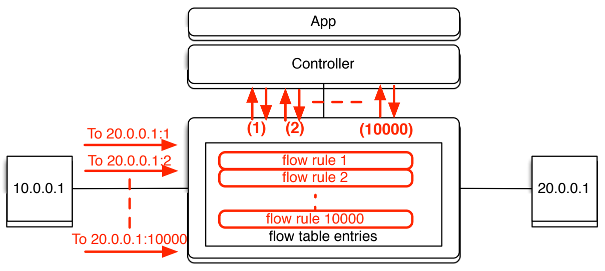

Figure 1: Possible resource consumption attacks to an OpenFlow switch and a controller

It is a serious problem to both of controllers and switches. In the case of the controller, it should handle much more requests than usual, thus, it can face serious performance problems (remember that a network packet is pending for process by the switch until the controller gives a flow rule for the packet). The OpenFlow switch also suffers. It should maintain flow rules for each network packet, and it can consume the capacity of fixed flow table entries in the switch. Thus, it is possible that further network packets can not be handled by the switch or they should wait until the switch clears off old flow rules. Finally, this kind of attack can consume resources of the control plane (flow rule handling capacity) and the data plane (flow rule entries).

3. ATTACK METHOD

In this section, we describe how attackers can fingerprint SDN networks, and how they can consume resources for SDN easily.

3.1 Assumption and Attack Model

Assumption: Before we describe our attack method, we clarify two basic assumptions. First, we assume that a target SDN network is managed by more fine-grained flow conditions (e.g., 4-tuples) than legacy networks. We believe that this assumption is valid because most cases people deploy SDN technology to control their network with micro-level flow rules for efficient management [14]. Second, we assume that an attacker can remotely scan the target network (e.g., TCP scanning, or even just ICMP scanning). This is a commonly acceptable assumption because most networks allow some TCP connection requests or ICMP (ping) requests from remote hosts.

Attack Model: Usually, a computer network is modeled with a graph structure whose nodes are hosts and links are connections. In addition, a set of linked nodes (hosts) can be considered as a special purpose network (e.g., an enterprise network), and we call this a group. In this case, we have two different groups: (i) a group with SDN technology (we call this SDN-group) and (ii) a group without SDN technology (we call this non-SDN-group).

Here, our attack can be modeled as follows. First, an attacker node contacts multiple groups, whose node(s) can be reached by the attacker node, by sending network probing packets. Second, the attacker node collects the data samples of response times for the probing packets. *Note that the response time means the time difference between the time when a node sends a probing packet and the time when a node receives a response for the probing packet, and we use this definition for the response time in our paper*. Third, the attacker node conducts a statistical test with collected data samples, and it will differentiate SDN-groups from non-SDN-groups. Finally, if the attacker node detects a SDN-group, it conducts a DoS attack to the SDN-group by sending attack packets. Our attack differs from the existing DoS attacks in that it mainly targets the data plane and the control plane of SDN instead of end hosts. In the following sections, we describe each operation of this model in detail.

3.2 Fingerprinting a SDN network

Since SDN separates the control plane from the data plane, the data plane asks the control plane to get a flow rule when there is no flow rule to handle this packet (*we call this New-Flow*). In this case, the data plane needs to wait until it receives a flow rule from the control plane (usually, we call this flow setup time), and it can be considered as a kind of additional delay compared with the case of legacy network devices, in which the control plane and the data plane are integrated into a single box. If the data plane receives a flow rule, it does not need to ask the control plane when it handles network flows that are matched to the condition fields of the flow rule (*we call this Existing-Flow*).

If a client contacts a SDN network, this client will face these two cases. Here, the important part is that this client will observe different response times when the client sends packets to a SDN network, because the flow setup time can be added in the case of New-Flow compared with the case of Existing Flow. To describe this more clearly, we simply formalize the response time that is observed at a client side. First, we define the response time for the Existing-Flow case as α, and flow setup time for β. In addition, for brevity, we define the response time for the case of New-Flow and Existing-Flow as $T1$ and $T2$ respectively, and they can be represented as follows.

$T1$ (w/o flow rule in the data plane) = $\alpha + \beta$
$T2$ (w flow rule in the data plane) = α

In this case, if an attacker can clearly differentiate $T1$ from $T2$, he can fingerprint a SDN network. However, an attacker will still face two following challenges: (i) how to collect $T1$ and $T2$ values and (ii) how to know whether $T1$ values are different from $T2$ or not.

Header Field Change Scanning: The first challenge can be addressed by our new network scanning method, *header field change scanning (HFC scanning)* that scans networks as changing network header fields. When this scan tool collects $T1$ and $T2$ values, it uses the characteristics of SDN networks, and that is *the data plane in a SDN network asks the control plane if it does not have any flow rules to handle a new incoming network flow*. It implies that if an attacker knows the condition for a flow rule managing a network, he can send a packet that lets the data plane ask to the control plane, and it finally helps the attacker collect $T1$ and $T2$ values easily.

To do this, our scanning method should attempt to understand (reverse engineer) the condition of the flow rules managing the network. The condition can be considered as the granularity of network flow rules enforced by a SDN application. For example, if a SDN application manages a SDN network by enforcing 4-tuples flow rules (i.e., source and destination IP address, and source and destination port), then, the condition of a flow rule for this network is 4-tuples. Here, the problem is that the attacker does not know the condition of such flow rule in use initially.

HFC SCANNER solves this problem by simply modifying the fields of a packet header when it scans a network. It first sends two (or more) same network packets to a target network, and if it receives response packets for these packets, it sends another two (or more) packets whose header field (e.g., destination IP address) has been modified to the target network. Whenever it receives a response packet, it measures the response time (i.e., the time difference between the time when HFC SCANNER sends a packet and the time when HFC SCANNER receives a response packet). Here, the key point is HFC SCANNERr sends the same network packet to a target network twice (or more). HFC SCANNER assumes that the first packet may make the data plane ask the control plane, and it also assumes that the second packet will be handled by the data plane itself, because the data plane should have a flow rule for the first packet. Based on this intuition, HFC SCANNER may consider that the response time for the first packet as $T1$ and the response time for the second packet as $T2$.

HFC SCANNER continues this operation until it collects enough samples by modifying a value for a (dfferent) field of a network packet header. For example, if HFC SCANNER conducts this operation when it changes a destination IP address of a packet and it wants to collect 10 samples, then it modifies the destination IP address of a packet ten times, and it sends each modified packet to a target network twice (or more).

If HFC SCANNER collects enough samples for a condition, it tries to collect samples for other conditions. Since HFC SCANNER (and the attacker) does not know the condition of a flow rule that is used for a SDN network, it tries most possible network conditions. To do this, HFC SCANNER continues the operation explained in the above with modifying another field of a network packet header. For example, if HFC SCANNER collects some samples with modifying the value of destination IP address, after that it will collect some samples with modifying the value of destination port. Finally, HFC SCANNERr collects sample values for each possible flow condition, and these values will be used to detect whether a target network is using SDN functions or not. The overall operation of HFC SCANNER is presented in Figure 2.

Advanced Scanning Technique: When HFC SCANNER probes networks by modifying a field of the packet header, it is not easy to change some parts. For example, if HFC SCANNER changes the source IP address of a packet header, it can not receive the response packet. In this case, HFC SCANNER can choose another strategy to conduct such scanning more robustly. First, HFC SCANNER uses the timeout setting of a flow rule. Usually, a flow rule enforced by the control plane has a timeout (e.g., in the case of OpenFlow specification, it is recommended at 30 seconds), and if this timeout has been expired, the rule will be deleted. Thus, HFC SCANNER can scan a network without changing a source IP address, but it just sends the packet in more than 30 seconds interval. Moreover, using timeout can also provide the effect of stealthy scanning, thus, an attacker can scan a network with minimal detection.

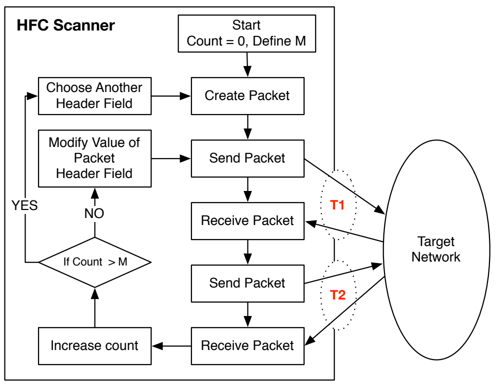

Figure 2: Simplified function diagram of HFC SCANNER

Second, an attacker can rent some bot infected hosts to run HFC SCANNER in multiple hosts. Since it is cheap to rent bot infected hosts (e.g., renting 1,000 hosts just costs 25 US $ [15]), the attacker can easily hire multiple hosts and he can run HFC SCANNER with its own source IP address, which is the same effect of modifying source IP address.

Statistical Testing for Two Sample Sets: If an attacker collects samples of $T1$ and $T2$ using HFC SCANNER, he will face the second challenge. This challenge can be solved by employing statistical testing methods, such as *t-test* [2]. This test simply tests whether two samples (i.e., $T1$ and $T2$) are significantly (statistically) different from each other or not with a high confidence. This test just requires a mean and standard deviation values of each sample that can be easily obtained, and the test method is pretty simple. Of course, an attacker can easily use more advanced statistics or machine learning techniques to improve the accuracy.

3.3 Launching DoS attacks to a SDN network

If an attacker runs HFC SCANNER and collects network information, he can investigate whether a target network is using SDN or not through a simple statistical testing method. If the test results show that a target network is likely to use SDN, the attacker will further conduct the resource consumption attack. Since the attacker already knows the condition of the flow rule for the target network (with the help of HFC SCANNER), now he just needs to send network packets to consume SDN resources of the target network.

The basic requirement for this attack is pretty simple, i.e., to send network packets that can produce new flow rules. For example, if the attacker knows that the condition of the flow rule for the target SDN network depends on destination IP address, he can send many network packets with different destination IP addresses to the target SDN network. Then, these packets will be reported to the control plane (i.e., resource consumption attack of the control plane), and the corresponding new flow rules will be enforced to the data plane (i.e., resource consumption attack of the data plane).

Let's take a closer look at a more practical example case. HP 5406zl switch that supports OpenFlow functions has been know that it supports 1,500 OpenFlow flow rules [1]. And, we assume that a network administrator uses this switch to provide SDN functions for his network, and he creates an application manages his network by enforcing flow rules defining 4-tuples. In this case, if the attacker fingerprints this network and finds its condition (i.e., 4-tuples) through HFC SCANNER and a statistical testing method, he can simply produce 1,500 packets with different source ports to consume all flow tables of this switch. The default timeout for a flow rule is 30 seconds [8], and it means that if the attacker generates 1,500 packets within 30 seconds, he can achieve his goal. The attacker only needs to send 50 packets per second, which is a very easy job with any kinds of computer.

Advanced Resource Consumption Technique: If an attacker sends too many packets to a target network to cause resource consumption, it could be detected by a security monitor. Thus, he needs to make a plan for attacking the target network with minimal detection. To do this, he can choose a condition that can hide himself. For example, if the source IP address can be used to let the data plane ask the control plane, an attacker can send packets with spoofed source IP addresses to a target network. Then, the SDN resources of the target network (i.e., flow table entries and flow rule handling capacity of the control plane) will be consumed without revealing the attacker's origin.

4. EVALUATION

To understand the feasibility of the proposed attack, we have evaluated it with a real world test. HFC SCANNER requires $T1$ and $T2$ values to detect a SDN network. However, in current network situation, it is very hard to collect this information from the Internet, because SDN is not widely deployed to many networks (but we believe that SDN will be employed to many networks soon). Therefore, we have decided to use other measurement results to estimate $T1$ and $T2$ values.

4.1 Data Collection

We send 20 `ping` packets to 28 different networks (we call them target networks) to collect $T2$ values, and we collect the response times from the second packets (i.e., ignore the response time for the first packet) to avoid any possibility of including flow setup time of a SDN network. We send `ping` packets from a state in U.S.A., and the locations of the target networks are distributed in the same state, in different states (the same continent), and in different continents. Some information of target networks is shown in Table 1. The mean and standard deviation values of the response time for each target network is presented in Table 2.

Location	*Count*	*Domain Type*	*Percentage*
In state	3	3 .edu	10.7%
out of state	16	6 .com, 10 .edu	57.1%
out of continent	9	5 .com, 4 .edu	32.2%

Table 1: Information of target networks

4.2 Estimating $T1$ Values

It is very hard to get the information of $T1$, thus we decide to estimate $T1$ by adding flow setup time to $T2$ values. With the help of the previous work [12], we can get the information of flow setup time for three different control plane cases: (i) new version of NOX [7], (ii) Beacon [6], and (iii) Maestro [5]. From this work, we can obtain the mean and standard deviation value of flow setup time of each control plane, and we regenerate 20 flow setup times (i.e., β) based on this information (at this time, we assume that flow setup time follows the normal distribution). Finally, we add these regenerated values to $T2$ (i.e., one of regenerated value + one of $T2$ samples) to estimate $T1$ values.

There are several experimental results for flow setup time in the previous work [12], and we choose three cases for our evaluation: (i) when 1 switch is connected to the control plane (1-switch), (ii) when 16 switches are connected to the control plane (16-switch), and (ii) when 256 switches are connected to the control plane (256-switch). Therefore, we finally have 9 different SDN network environments for estimating $T1$ (i.e., three different control planes with three network configurations).

4.3 Fingerprinting Result

We apply *t-test* [2] to collected $T2$ and estimated $T1$ samples to figure out if $T1$ is significantly different from $T2$, and the test results (i.e., fingerprinting results) are shown in Table 3. The test results are same as the detection results by HFC SCANNER, and we report in the table whether $T1$ is distinguishable, and scan success (S in the Table) or failure (F in the Table).

When there is a single switch connected to the control plane, HFC SCANNER can find most of target networks. In the case of the NOX control plane, it can fingerprint 24 networks out of 28 (i.e., fingerprinting rate is 85.7%), and the results are the same when there is Beacon or Maestro control plane. Interestingly, HFC SCANNER does not detect 4 networks for all control planes, and these networks show relatively longer response time than other networks and have large standard deviations. In this case, flow setup time may not be distinguishable. However, since most networks (even located in the difference continents) can be reached in short time (e.g., less than 90 ms), HFC SCANNER can still effectively figure out whether they employ SDN or not.

When 16 switches are connected to the control plane, fingerprinting results are similar to the case of 1 switch. However, in the case of the Beacon control plane, HFC SCANNER does not fingerprint any networks. The reason is the flow setup time of the Beacon control plane (16 switch case) shows large standard deviation values compared with its mean value (i.e., mean value is 11.89 and standard deviation is 26.22). Therefore, the distribution of flow setup time is not distinguishable. This kind of characteristics can be also observed when 256 switches are connected to the control plane. In this case, the standard deviation is also quite high compared with mean value of the flow setup time. However, in the case of Maestro control plane, HFC SCANNER can detect most of networks because the standard deviation of the flow setup time for the Maestro control plane is not so high.

Based on these results, we find two cases that a SDN network is likely to be fingerprinted by a remote attacker: (i) if the flow setup time is not so smaller than the response time (in our experiment, more than 10% of the response time), and (ii) if the standard deviation of the flow setup time is not much bigger than the mean value of the flow setup time (in our experiment, less than 2 * mean). In addition, we observe that the distance of a target network from the attacker is not so important, instead the response time is much more important. If the response time is comparable with the flow setup time, we believe that an attacker can fingerprint most network using SDN no matter where they exist.

4.4 DoS Attack Result

We have set up a test environment to understand whether the proposed DoS attack is successful or not, and the environment con-

	in state			out of state										
ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Average (ms)	13.49	67.42	68.28	89.34	69.42	83.47	62.89	56.9	50.38	87.48	78.03	52.61	62.82	70.29
STD	3.55	3.01	7.23	2.58	4.49	2.42	2.31	3.18	4.17	62.83	2.86	2.36	4.28	2.09
	out of state					out of continent								
ID	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Average (ms)	61.74	109.58	95.93	99.7	61.52	354.69	143.27	48.93	49.16	338.7	333.26	130.19	62.21	50.42
STD	1.52	4.71	3.09	11.87	2.6	94.45	1.67	1.08	2.62	103.52	92.97	3.53	5.39	5.71

Table 2: Average response time for each target network

	in state			out of state										
ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14
NOX (1-switch)	S	S	S	S	S	S	S	S	S	F	S	S	S	S
NOX (16-switch)	S	S	F	S	S	S	S	S	S	F	S	S	S	S
NOX (256-switch)	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Beacon (1-switch)	S	S	S	S	S	S	S	S	S	F	S	S	S	S
Beacon (16-switch)	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Beacon (256-switch)	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Maestro (1-switch)	S	S	S	S	S	S	S	S	S	F	S	S	S	S
Maestro (16-switch)	S	S	S	S	S	S	S	S	S	F	S	S	S	S
Maestro (256-switch)	S	S	S	S	S	S	S	S	S	S	S	S	S	S
	out of state					out of continent								
ID	15	16	17	18	19	20	21	22	23	24	25	26	27	28
NOX (1 switch)	S	S	S	S	S	F	S	S	S	F	F	S	S	S
NOX (16-switch)	S	S	S	F	S	F	S	S	S	F	F	S	S	S
NOX (256-switch)	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Beacon (1-switch)	S	S	S	S	S	F	S	S	S	F	F	S	S	S
Beacon (16-switch)	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Beacon (256-switch)	F	F	F	F	F	F	F	F	F	F	F	F	F	F
Maestro (1-switch)	S	S	S	S	S	F	S	S	S	F	F	S	S	S
Maestro (16-switch)	S	S	S	S	S	F	S	S	S	F	F	S	S	S
Maestro (256-switch)	S	S	S	S	S	F	S	S	S	F	F	S	S	S

Table 3: Fingerprinting results for target networks (S - Success, F - Failure)

sists of a single OpenFlow switch, a controller, and two hosts for network communications. We use the software based OpenFlow switch implementation for the OpenFlow switch [9], and it is installed on an independent linux host (i.e., software switch), and we set the maximum flow rules for this switch as 1,500, which is the same configuration for HP 5406zl switch [1]. We use POX as the controller, and it launches a simple layer 4 packet switching application (thus, this application enforces a flow rule with 4-tuples granularity). Other two hosts are simple linux boxes; one is used for the proposed attack, and the other is used to run a TCP server program. For DoS attack packets, we simple use the *tcpreplay* tool [13] to send multiple network packets, whose 4-tuples are different from each other. In addition, we control the packet sending rate from 50 pps (packets per second) to 600 pps to launch diverse attack scenarios.

Figure 3 (time) presents that how long it will take for an attacker to consume all flow rule entries in the data plane, and it clearly shows that an attacker can successfully conduct a DoS attack to the data plane. Although an attacker sends attack packets in low rates (e.g., 50 pps or 60 pps), he can make the data plane hard to handle normal network flows in 30 seconds. Moreover, if the attacker sends attack packets intensively, he can consume all flow table entries in 3 seconds. It is a critical problem, because even if there might be a defending approach to protect the data plane from the DoS attack, if it does not stop sending attack packets in 3 seconds, it cannot protect the data plane.

In addition, we measure the bandwidth required for the proposed attack, and it is presented in Figure 3. It shows that the DoS attack requires around 200 Kbps (bytes per second) in maximum and 20 Kbps in minimum. It implies that it is possible that an attacker can conduct a relatively stealthy DoS attack to a SDN network. If the attacker can hire multiple hosts by renting bot infected hosts, he can conduct a DoS attack with minimal detection. For example, if the attacker hires 100 hosts, each host only requires to send packets at 200 bps, which mimics a normal client quite well and thus hard to detect/defeat.

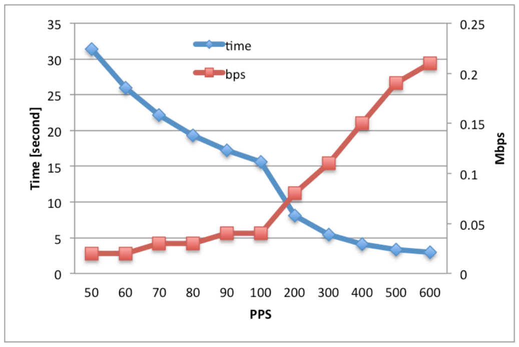

Figure 3: Required attack time and network bandwidth for DoS attack

5. POSSIBLE DEFENDING TECHNIQUES

To defend SDN networks from the proposed attack, we now discuss some possible defending methods.

First, we can compress the flow rules by changing the condition of a flow rule to make the flow rule cover wider ranges with wildcards. For example, if the destination port is a part of a condition for a SDN network providing load balancing function, we can change this part (i.e., the destination port) into the wildcard to let a single flow rule handle many network flows, when there are too many network flows to handle. Of course, it could ruin some load balancing policies. However, it can make the SDN network still be working, and in some situation it is more important (to keep the network resilient from proposed DoS attack) than preserving the load balancing policy.

Second, we can add some new functions that can detect this kind of scanning attack to the data plane. If a host produces some network flows that can generate new flow rules in a short time period, we can consider it as suspicious. Then, we can ignore some network packets from this source for some time interval. However, this may not work for distributed attacking hosts, e.g., a botnet.

Third, based on our detection results, we observe that if the standard deviation of the flow setup time is high, HFC SCANNER cannot effectively fingerprint a SDN network. We can use this characteristics to mislead the proposed attack. For example, we can make the control plane varies the flow setup time dynamically to confuse HFC SCANNER. However, we should consider that the flow setup time should still meet some requirement (e.g., flow should enforced within 20 ms) in some cases.

6. RELATED WORK

Till now, there is very little work on the security attacks against SDN. In our early work [11], an evasion attack called dynamic flow tunneling is proposed to evade existing SDN security policies. In this paper, we focus more on the fingerprinting and DoS attacks on SDN. In [14], Wang *et al.* have also noticed the resource consumption problem when an OpenFlow application performs load balancing functions. They have proposed a method to change the granularity of flow rules dynamically to reduce this flooding pressure. They merge flow rules that they share some network space (e.g., the same /24 subnet) and change them into wildcard rules. This approach can reduce the number of fine-grained flow rules. However, it still can not address all issues. First, although their proposal can reduce the number of fine-grained flow rules, an attack can still generate flooding attack packets whose source and destination IP addresses are randomly chosen. Then, it is hard to merge flow rules for these packets. Second, OpenFlow switches use TCAM (Ternary Content Addressable Memory) to handle wildcards flow rules with high performance. Since this TCAM is much more expensive than SRAM that is used for storing flow rules without wildcards, switch vendors typically only equip a very small size of TCAMs into OpenFlow switches. DevoFlow [1] has been proposed to relieve this issue. It employs the technique of *flow rule cloning* that changes flow rules with wildcards into a flow rules without wildcards. Combining the approaches proposed by Wang *et al.* [14] and DevoFlow [1], we can reduce the effects of flooding. However, it still does not address the problem when an attacker generate random flooding attack packets.

7. CONCLUSION AND FUTURE WORK

In this paper, we introduce a new fingerprinting attack against a SDN network, and we also show its feasibility with real world experimental data. To the best of our knowledge, the proposed attack scenario is the first attack case to a SDN network that can be conducted by a remote attacker, and this attack could significantly degrade the performance of a SDN network without requiring high performance or high capacity devices. We believe that discussing new threats against a SDN network is very important because it can guide us to design better (more secure) SDN solutions. In addition, we believe that SDN is still evolving and improving, and this kind of discussion is necessary to lead SDN to the right track.

In our future work, we will set up a more realistic SDN network environment for our evaluation, and we plan to further improve HFC SCANNER. In addition, we will consider more possible network attack cases against a SDN network, as well as more possible defending techniques.

8. REFERENCES

[1] 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.

[2] J. Fisher Box. Guinness, gosset, fisher, and small samples. In *Statistical Science*, 1987.

[3] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *Proceedings of NSDI*, 2010.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. In *Proceedings of ACM SIGCOMM CCR*, April 2008.

[5] T. S. E. Ng, A. L. Cox, Z. Cai, F. Dinu, and J. Zheng. Maestro openflow controller. `https://code.google.com/p/maestro-platform/`.

[6] On.Lab. Beacon openflow controller. `https://openflow.stanford.edu/display/Beacon/Home`.

[7] On.Lab. Nox openflow controller. `http://www.noxrepo.org/`.

[8] OpenFlow. OpenFlow Swtch Specification version 1.1.0. Technical report, 2011. `http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf`.

[9] OpenFlow.org. Openflow switching reference system. `http://www.openflow.org/wp/downloads/`.

[10] L. Popa, M. Yu, S. Y. Ko, I. Stoica, and S. Ratnasamy. Cloudpolice: Taking access control out of the network. In *Proceedings of HotNets*, 2010.

[11] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for openflow networks. In *Proceedings of HotSDN*, 2012.

[12] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *Proceedings of HotICE*, 2012.

[13] A. Turner. TCPReplay. `http://tcpreplay.synfin.net/`.

[14] R. Wang, D. Butnariu, and J. Rexford. Openflow-based server load balancing gone wild. In *Proceedings of HotICE*, 2011.

[15] WEBROOT. How much does it cost to buy 10,000 U.S.-based malware-infected hosts? `http://blog.webroot.com/2013/02/28/how-much-does-it-cost-to-buy-10000-u-s-based-malware-infected-hosts/`.