Mitigating High-Rate Application Layer DDoS Attacks in Software Defined Networks

João Henrique[1], **Iguatemi E. Fonseca**[1] **and Vivek Nigam**[1]

[1]Federal University of Paraíba, Brazil

`jhenrique@gmail.com, {iguatemi,vivek}@ci.ufpb.br`

Abstract. *Differently from previous attacks, many recent DDoS attacks have not been carried out over the network layer, but over the application layer. The main difference is that in the latter, an attacker can target a particular application of the server, while leaving the remaining applications still available, thus generating less traffic and being harder to detect. Recently, we have proposed the use of selective strategies for mitigating Low-Rate Application Layer DDoS attacks (ADDoS). Unfortunately, due to their higher traffic load, High-Rate ADDoS attacks (*e.g.*, GET flooding) remain a serious problem as they still generate traffic in much lower proportions than usual network-layer DDoS attacks, but they generate traffic load high enough to render our and other defenses ineffective. This paper proposes a new defense mechanism, called SHADE, that uses selective strategies in Software Defined Networks (SDN) for mitigating High-Rate ADDoS. As SDN controllers have a global view of the network, they allow re-directing traffic when applications, such as web-servers, are (on the verge of being) overloaded. Traffic is re-directed to mitigation applications implementing our selective strategies, reducing the traffic hitting the target application and ultimately allowing it to serve more clients. We carried out a number of simulations on realistic attack scenarios. Our simulations show that without our defense, the target application can only serve 19% clients with an average time to service of 5.4s, while with our defense it can serve 61% of clients with an average time to service of 1.6s.*

1. Introduction

Application-Layer Distributed Denial of Service (ADDoS) attacks is a new generation of Distributed Denial of Service (DDoS) attacks where attackers target a particular application by exploiting application layer protocols, such as HTTP and SIP. Differently from Network-Layer DDoS attacks (such as SYN-flooding), which attack the whole machine affecting all its applications, by targeting a single application, ADDoS generates much less traffic and, therefore, are harder to detect using traditional network traffic analysis tools.

We can further classify ADDoS attacks into two types: Low-Rate and High-Rate attacks. Low-Rate attacks are carried out by attackers simulating the traffic of legitimate clients. Examples of such attacks include Slowloris [slo13a], POST [rudy13] and Slowread [slo13b] attacks. High-Rate DDoS attacks, on the other hand, are similar to Network-Layer DDoS in that they send a large amount of packages to the target application exhausting its resources. However, as they target a single application and not the whole server, the number of packages is still very small for defenses based on network analysis to be effective. Indeed, GET-Flooding, which is an High-Rate ADDoS attack which sends a large number of GET requests to the target web-server, is one of the most popular DDoS attacks (in 2014, it was placed fourth in popularity after DNS, TCP and UDP flooding [Foc14]).

In our previous work [DNF14], we proposed a defense mechanism, called SeVen, for mitigating Low-Rate ADDoS attacks. While SeVen can guarantee very high availability rates (around 95%) against Low-Rate ADDoS attacks, as our simulations demonstrate, it is much less effective against High-Rate ADDoS Attacks with an availability of only 24% of legitimate clients. The main problem, as in all flooding attacks, is the very high rate of packages generated by attackers (more than 180 times the normal client traffic rate). Indeed, without any defense, an Apache web-server suffering a GET-Flooding attack can only serve 19% of the legitimate clients.

This paper proposes a new approach for mitigating High-Rate ADDoS attacks by using Software Defined Networks (SDN). By decoupling the control and the data planes. SDN transfers the network controlling, such as routing and traffic management, to a powerful centralized controller with a global overview of the flows in the networks. The lower level hardware, such as switches, simply follow the decisions specified by the controller. SDN, thus, facilitates the management of networks allowing to easily control traffic according to the specified network decision logic.

Our contributions are two-fold:

- **SHADE: A SDN framework for mitigating High-Rate ADDoS:** As SDN controller's have a global view of the flows in the network, setting the rules for routing packages, it can be used to re-direct the flow towards an application that is overloaded. In particular, we use SeVen additionally for monitoring the capacity use of web-servers (Apache). Whenever SeVen detects that the web-server is in the verge of being overloaded, *i.e.*, its capacity to process request is reaching its limit, then it sends a message to the controller to activate a selective defense. The controller re-directs the flow to this web-server making a de-tour to a mitigating application implementing a selective strategy. The mitigation application will drop a number of packages reducing, thus, the traffic rate that is hitting the overloaded web-server allowing legitimate clients to be served. We call our defense mechanism SHADE–**S**elective **H**igh r**A**te **D**DoS d**E**fense;
- **Simulation Results:** We implemented this architecture using standard SDN simulators (Mininet [Min15]), and controllers(Ryu [ryu15]).[1] We also adapted the code implementing SeVen so that it monitors the web-server's capacity and communicates the controller whenever the web-server is overloaded. Finally, we also implemented SHADE, including the machinery to re-direct traffic using SDN rules. Our simulation results have shown that under a GET-flooding attack (generated using the tool LOIC [LOI13]), without using any defense, only 19% (in average) of legitimate clients with a time to service (TTS) of 5.4 seconds, and using our SHADE, 61% of clients (in average) are served by the web-server with a TTS of approximately 1.6 second. We also observed that SHADE discards in average 70% of the traffic generated by the attackers. This means that a great part of the

[1]Due to their precision in simulating SDN networks, carrying out simulations using Mininet is the standard in the SDN literature for validating solutions.

useless traffic generated by the attacker is discarded already at the entrance of the network where the web-server is and therefore, not passing through other switches and machines inside the network.

This paper is organized as follows: Section 2 briefly reviews the main technologies that we use to implement SHADE, namely SDN and SeVen. We assume some familiarity with SDN. We also explain the main High-Rate ADDoS attack used by attackers and also used here, namely the GET-Flooding attack. Our solution for mitigating High-Rate DDoS attack, namely SHADE, is explained in Section 3 and we detail our main results in Section 4. Finally, we review related work in Section 5 and conclude by pointing out future work in Section 6.

2. Preliminaries

We discuss the basic machinery used to develop and validate our ADDoS mitigation solution, namely, the attack GET-Flooding, Software Defined Networks and the defense SeVen.

SDN In traditional networks controlling tasks, such as routing, traffic engineering and access control, is implemented in low level machinery, such as routers, implementing for example distributed routing protocols. This makes the maintenance of the network hard and error-prone as each router participates in the implementation of the network decisions [GHM+05]. SDN solves this problem by decoupling the control plane, where the networking controlling tasks are defined, such as routing, and the data plane, where packages are managed following the decisions specified in the control plane. Programmers can specify the logic of their network decisions, such as traffic management, using standard APIs such as OpenFlow [Ope15].

As they can be easily configured, SDN has been successfully used for a number of applications including DDoS mitigation. However, until now most of this work has concentrated in very large traffic attacks, such as traditional Network-Layer DDoS attacks, such as SYN-Flooding, and Amplification attacks. Few have considered using SDNs for mitigating Low-Rate ADDoS. (More details in Section 5.)

GET-Flooding The main High-Rate ADDoS attack used by attackers, called GET-Flooding, exploits the GET method of the HTTP protocol. The idea is simple. The attacker (or his infected bots) simply sends a large amount of GET request to the target web-server exhausting its memory and CPU time in the server making it not serve legitimate clients. Notice that since a web-server has only a fraction of the memory and CPU usage of the machine, one does not require a huge number of packages. Moreover, even amateurs can carry out such an attack by using tools such as LOIC [LOI13]. The GET-flooding attack is very effective. It is possible to make a small to mid size web-server unavailable using just some instances (3-4) of LOIC. In 2014, it was one of the four most popular attacks [Foc14].

SeVen Our previous work [DNF14] introduced SeVen, a selective defense for mitigating ADDoS. SeVen uses selective strategy to mitigate DDoS: It works as a proxy and monitors the usage of the web-server under its protection. When the web-server is not overloaded[2], it allows any request (even of attackers) to be processed. However, *when the web-server is overloaded* (that is, it can no longer process a new request simultaneously), and a new request arrives, SeVen decides (using some probability function) whether the web-application should process this new request. There are two outcomes:

1. SeVen decides that it should not process this new request, in which case it simply returns a message that the web-server is unavailable;
2. SeVen decides that it should process this new request. Then it should also decide which request being currently processed should be dropped. This decision is taken based on another probability distribution.

Intuitively, selective strategies work because whenever an application is overloaded, it is very likely that it is suffering an attack. Instead of blocking all new requests (from both attackers and legitimate clients) as done without our defense, SeVen opens the possibility of new request from legitimate clients to be processed improving considerably the availability of the application. We applied such selective strategies to two different applications: Low-Rate ADDoS exploiting the HTTP protocol [DNF14], namely for mitigating Slowloris and POST attacks, and Telephony Denial of Service attacks exploiting the SIP protocol used in Voice over IP applications [LDF$^+$ed].

3. SHADE– Selective Strategies in SDN

Problem Statement Although SeVen works well for mitigating Low-Rate ADDoS attacks, it also suffers when trying to mitigate High-Rate ADDoS attack. This is because SeVen also has limited memory and CPU power and it cannot handle the overwhelming number of packages that arrive when an application is suffering a High-Rate ADDoS, such as in the GET-Flooding attack:

When suffering a Low-Rate ADDoS attack, such as a Slowloris attack, SeVen can still guarantee service to 95% of legitimate clients. However, when suffering a High-Rate ADDoS attack, such as GET-Flooding, SeVen can only guarantee service to an average of 24% of legitimate clients.

For example, when carrying out the GET-flooding attack, the number of packages arriving the application increases by a factor of 180. Despite this increase on the number of packages, however, the overall traffic increase is very low (of some megabytes) when compared to usual flooding attacks carried out over the network layer. Therefore, defenses based on network analysis tools are not able to detect these attacks [ZJT13].

The problem is, therefore, to develop a mechanism to defend an application (web-server) from both Low-Rate ADDoS and High-Rate ADDoS attacks finding mechanisms to mitigate them. While Low-Rate ADDoS attacks can be handled by SeVen, this paper proposes a mechanism for mitigating High-Rate ADDoS attacks using SDN and SeVen.

Proposed Solution Our solution, called SHADE (**S**elective **H**igh r**A**te **D**DoS d**E**fense) combines the following two technologies which have complementary features:

[2]That is, the number of request being processed is less than its maximum capacity.

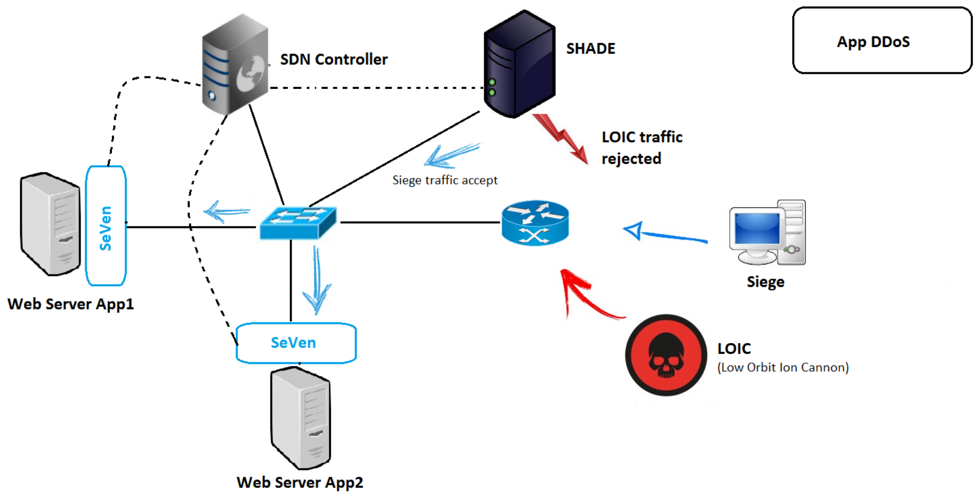

Figure 1. Defense Mechanism Topology in an SDN network

- **SeVen's Load Monitoring:** In order to apply its selective strategy, SeVen keeps track of resources being used by the web-server, namely, the number of request that are being processed. This allows us *to use SeVen as a sensor* indicating when the application under its protection is overloaded receiving too many packages. We classify levels of usage, *e.g.*, Green, Yellow and Red, depending on the web-server's usage:
  - **Green** indicates when the web-server usage is as expected. That is, there are enough free resources, *e.g.*, workers, to process new requests;
  - **Yellow** indicates that the web-server is being highly used and if it receives more requests, then it will reach its maximum capacity to process request;
  - **Red** indicates that the web-server's capacity is almost depleted and it cannot or will soon no longer be able to process new requests.
- **SDN's Centralized Control of Flows:** SDN controllers have a global view of the network and can control flows by setting rules in switches accordingly. This is a great feature to manage traffic in a network by redirecting a flow so that it passes some particular application, such as load balance applications or intrusion detection applications or in our case, a DDoS mitigation application. Whenever a web-server is overloaded, *e.g.*, in status Yellow or Red, it can communicate this to the controller. The controller can then set the flow to that particular application so that it is redirected to SHADE.

Figure 1 illustrates the topology of our defense mechanism against High-Rate DDoS attacks. SeVen is installed together with the applications. We assume a typical SDN setting, with a SDN controller (Ryu), applications (Apache web-servers), legitimate clients (generated using Siege) and attackers (generated by LOIC instances). Whenever SeVen detects a change in the state of the web-server (Green, Yellow or Red), it communicates it to the SDN controller. If the status is Green, then the controller does not redirect the flow to SHADE. Otherwise, it does redirect the flow by installing a higher priority rule in the switch.

Figure 2 illustrates how the SDN rule is used to re-direct traffic. The computer to the right sends a package to the server to the left of the figure running SHADE. The

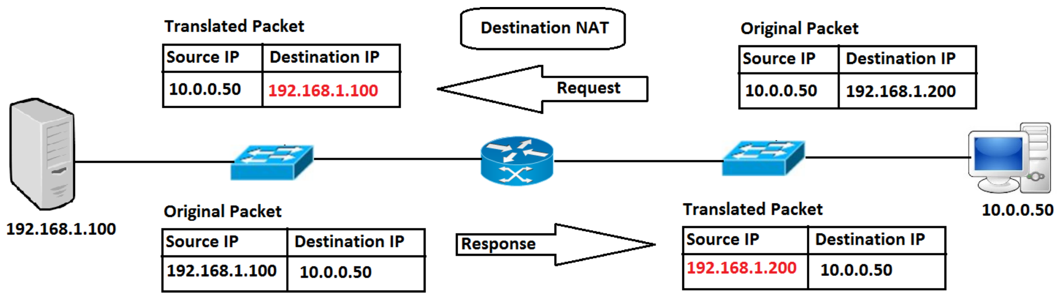

Figure 2. Defense Mechanism Topology in an SDN network

switch follows the rule installed by the controller. In the diagram in Figure 2, the installed rule translates the destination header of the incoming package to the IP address of the machine running SHADE. Similarly, translates the source IP header of the machine running SHADE to the IP of the machine running the web-server so that the client or attacker does not detect that the flow is passing through SHADE.

Finally, the controller also informs SHADE the status of the web-server (Green, Yellow, and Red). SHADE when receiving the flow to the target web-server, uses the SeVen status informed by the controller to apply a selective strategy as follows:

- **Yellow:** If the status is Yellow, then SHADE applies a less strict selection strategy dropping less packages. In our simulations, we adopted a Round Robin strategy selecting 1 package of every 3 packages received. The packages not selected are dropped and corresponding client is informed that the service is not available, while the selected packages are sent to the web-server. That is, SHADE reduces by two thirds the traffic hitting the web-server.
- **Red:** If the status is Red, then SHADE applies a much more aggressive selection strategy dropping many more packages. In our experiments, SHADE applies a Round Robin strategy selecting just 1 package out of 5 packages received reducing considerably the overall traffic received by the application.

Remark: We used just three levels of usage (Green, Yellow, and Red) as they were enough to illustrate that our defense can mitigate High-Rate ADDoS attacks. There are other possibilities such as use instead of the capacity, the rate of incoming packages, or a more fuzzy classification. The best choice will depend on the particular scenario and is left out of the scope of this paper.

Example Assume, for illustration only, that the web-server can only process 10 requests and that the status of the web-server is Yellow if the number of requests being processed, n_P, is $7 \leq n_P < 9$, it is Red if $n_P \geq 9$ and Green otherwise. Moreover, assume that it is currently processing the following five requests identified as $id_1, \ldots, id_5$:

$$\langle id_1, id_2, \ldots, id_5 \rangle$$

Since $n_P < 7$, the status of the web-server is Green. This means that any new request incoming into the network does not pass through SHADE.

Assume now that a large number of requests, $id_6, \ldots, id_{17}$, possibly due to a DDoS attack, arrive the head of the network where the web-server is. The requests id_6 and id_7

are going to arrive SeVen without passing through SHADE because the web-server status is Green. Thus, the web-server starts processing id_6, but when it starts processing id_7, it sends a signal to the controller that its status is now Yellow because $n_P = 7$. This causes the controller to re-direct the flow to the web-server using the SDN rule illustrated above through SHADE implementing our selective strategy.

This means that from the next 3 request only one is sent by SHADE to the web-server, *i.e.*, two requests from $id_8, \ldots, id_{13}$ are selected. Say id_9 and id_{12} are selected by SHADE. These are sent to the web-server, which is now processing the requests:

$$\langle id_1, id_2, \ldots, id_5, id_6, id_7, id_9, id_{12} \rangle$$

At this point the status of the web-server is Red, which means that SHADE is going to select one request out of the next 5 requests, $id_{13}, \ldots, id_{17}$. Say that it selects the request id_{14}. The the web-server is processing the requests:

$$\langle id_1, id_2, \ldots, id_5, id_6, id_7, id_9, id_{12}, id_{14} \rangle$$

Now the web-server is in its full capacity. SeVen will now adopt its own selective strategy as described in Section 2. However, SHADE is still selecting one out 5 new incoming request reducing the load that is hitting SeVen and the web-server.

To illustrate how SeVen works (for more details see [DNF14]) say that a new request id_ν arrives SeVen, *i.e.*, it is selected by SHADE. Now, since the maximum capacity of the web-server is reached, SeVen decides whether it will process request id_ν or not. It decides this using some criteria (which is not important here). Say that it decides to process request id_ν. SeVen has to now decide which one of the requests being currently processed it shall drop. It decides this using some probability, here uniform probability. Say that it decides to drop id_5, then the requests being processed by the web-server are updated to:

$$\langle id_1, id_2, \ldots, id_4, id_6, id_7, id_9, id_{12}, id_{14}, id_\nu \rangle$$

where id_ν replaces id_5.

Therefore, there are two ways a package can be discarded by the selective strategy used by SHADE or by the selective strategy used by SeVen. An advantage of discarding an (attacker) request by using SHADE is that this package does not enter the network. It is blocked already at the network's entrance. On the other hand, discarding a package by using SeVen means that the package traveled the complete route until reaching the web-server before being dropped.

Finally, we observe that the use of the selective strategy by SHADE and by SeVen have very different purposes. While the strategy adopted by SeVen is built to mitigate Low-Rate ADDoS attacks, the strategy adopted by SHADE is built to mitigate High-Rate ADDoS attacks. We validate this intuition next by carrying out a number of simulations.

4. Simulation Results

Set-up We implemented the scenario depicted in Figure 1 in the simulator Mininet version 2.2.1 using the following tools:

- The applications being defended were Apache web-servers version 2.4.7. We used a small-web server with 30 workers;

- The SDN controller was Ryu [ryu15] version 3.6;
- For generating the client traffic, we used the tool Siege [Sie15] version 3.0.5. We used Siege to simulate 20 legitimate clients;
- For generating the attack, we used two instances of the tool LOIC [LOI13] version 1.0.8.0. LOIC is a tool developed by the Anonymous group and is often used to carried out DDoS even by amateurs. In our simulations, both instances of LOIC together generated 180 times more attacker traffic than client traffic;
- We implemented in Python the SDN rules for re-directing flows to SHADE whenever the status of the web-server is Yellow or Red;
- We implemented SHADE in C++ and modified the SeVen code so that it sends the usage status of the web-applications to the controller.

Our simulations were carried out using virtual machines using VMWare Player over a Windows 8.1 machine with a Core i5 processor 4210u and 6 GB of memory.

Remark: Due to the limitations of the simulator used, Mininet, in particular, its deficiency in carrying out large flooding attacks, our simulations considered a small web-server with only 30 workers. However, the amount of client and attacker traffic generated was set taking into account the number of workers.

We considered three scenarios for our simulations:

- **No Defense:** In this scenario, we do not use SeVen nor our SDN defense mechanism. This means that all the traffic generated by the attacker and the client hit the web-server directly;
- **Only SeVen:** We add one layer of defense, namely SeVen, to protect the web-server from attacks. As SeVen also applies a selective strategy, we would like to understand how effective it is for High-Rate ADDoS attacks;
- **Using SHADE:** Finally, we added our SDN defense mechanism to understand the impact of our defense on mitigating High-Rate ADDoS attacks.

We used the following quality parameters to measure how effective our defense mitigation solution is:

- **Availability:** We measured the number of packages generated by legitimate clients that were served by the target web-application. We considered as failure to be served, packages that are served after more than 5 seconds.[3] That is, if a client sends a request at time 10 and receives an answer from the web-server only at time 16, we consider this a failure to serve the client.
- **Time of Service (TTS):** We measured the average TTS of the packages sent by legitimate clients. That is, the average time that served packages took to be responded by the web-server;
- **Amount of Useless Traffic Discarded:** We measured the amount of traffic generated by the attacker that our defense was able to discard. As our defense re-directs traffic already at the entrance of the local network, discarding packages from the attackers reduces the overhead caused by the attack in all the switches where the flow passes by.

[3]While there is no consensus on how long a user is willing to wait for a page to load, it seems that 5 seconds is a good threshold. See, for example, `http://www.icegiant.co.uk/web-design/articles/the-five-second-rule.html`.

Scenario	Average Availability	TTS (s)
No defense	17-23% (average 19%)	3.0 - 9.1 (average 5.4)
Only SeVen	15-29% (average 24%)	1.9 - 6.0 (average 3.4)
Using SHADE	**58-64% (average 61%)**	**0.7 - 2.6 (average 1.6)**

Table 1. Results Summary on Availability and TTS for the Three Scenarios Considered

Results We carried out a number of runs for each scenario (No defense, Only SeVen, and Using SHADE) 8 runs each with duration of 10 minutes. Table 1 summarizes our main results. When using no defense at all, the availability of legitimate clients fluctuated between 17-23% with an average of 19%, while the TTS fluctuated from 3.0-9.1s with an average of TTS of 5.4. When using only SeVen to defend the Apache web-server, the availability of legitimate clients fluctuated between 15-29% with an average of 24% and the TTS fluctuated from 1.9-6.0s with an average of 3.4s. Finally, when using SHADE, the availability increased to values between 58–64% with an average of 61% and the TTS decreased to values between 0.7-2.6s with an average of 1.6s.

Clearly, the quality indicators improved by using SeVen, although it is built to mitigate Low-Rate ADDoS, with an increase of 23% in availability and 37% decrease in TTS. However, the quality measures improved considerably when using SHADE with *an increase of 220% in availability and a decrease of 70% in TTS*.

We also investigated how much traffic was dropped by SHADE and by SeVen both generated by attackers, which is useless traffic, and generated by legitimate clients. Table 2 summarizes our observations. Recall that both SHADE and SeVen adopt selective strategies (see end of Section 3). SHADE dropped 412k packages generated by the attacker corresponding 70% of all the traffic generated by the attacker. This means that a large amount of useless traffic generated by the attacker was already dropped at the entrance of the network by SHADE. From the remaining 123k packages, *i.e.*, the remaining 30%, that reached SeVen, only 1.2k packages was actually processed by the web-server. This means that only 0.3% of the total traffic generated by the attacker was processed by the web-server.

From the same table, we observe that the traffic generated by the attackers (588k) was almost 180 times greater than the traffic generated by the clients (3.3k). Despite this great difference, more packages from clients (2k) were served than attackers packages (1.2k). A possible explanation for this is the fact that the probability of dropping an attacker package is greater than a client package because there are many more attacker packages than client packages.

5. Related Work

There have been other proposal for using SDN for mitigation DDoS attacks. However, most of the literature [SYPG13, HSE14, BMP10, WXG15, FTSB15] on the topic focuses on Network-Layer DDoS attacks, such as SYN-Flooding, which have a much higher volume rate than Application-Layer DDoS attacks. We review some of this work.

Traffic	Dropped by SHADE (Average)	Accepted by SeVen (Average)
Attacker Traffic	70%	0.3%
Client Traffic	36%	61%
Number of Packages – Attacker	412k of 588k	1.2k of 123k
Number of Packages – Client	1.2k of 3.3k	2k of 2.1k

Table 2. Traffic Dropped by SHADE and Accepted by SeVen

Shin *et al.* [SYPG13] mitigate DDoS attacks by monitoring the number of handshakes TCP (SYN, SYN-ACK and ACK) that are not completed for each origin IP. If this number exceeds some threshold then it simply blocks the traffic from the IP origin using SDN rules. Since ADDoS attacks completes TCP handshakes, this solution cannot be used to mitigate these attacks.

The work of Braga *et al.* [BMP10] enumerates all possible parameters that can be extracted during an attack over the Network Layer (SYN-Flooding, UDP-Flooding, and ICMP-Flooding). From these parameters, they use a neural network to evaluate whether a flow is an attack or not. As all mechanisms based on neural networks, the classification quality of the network depends on the quality of data used to train them. Therefore, it is unlikely that their networks will be able to mitigate ADDoS attack which have very different behavior.

Other works [HSE14] identify possible attacks that exploit weaknesses of the SDN architecture. In particular, they demonstrate the possibility of denying the service of all applications on the SDN network by overflowing switches with a large number of rules. This attack is very different from the type of attacks that we investigate here and it can still be carried out when running SHADE. However, we believe it is possible to use selective strategies similar to the ones used here to mitigate such an attack. We leave this to future work.

SDN networks have also been used to mitigate Low-Rate ADDoS [OLL14]. Whenever any suspicious flow is identified they re-direct this flow using SDN rules to an application similar to the target application so that the attacker does not know that it has been identified. Although this work seems to be similar to ours, unfortunately, they do not explain how this could be done, even more so against Low-Rate ADDoS attack that have a very similar traffic profile as legitimate clients. If such distinction was indeed possible, then it would also be easy to incorporate this defense with SHADE.

Finally, more recently [FTSB15], Fayaz *et al.* proposed the combination of SDN networks and Network Function Virtualization (NFV) to mitigate large amplification attacks. Their solution is to identify large volumes of traffic and use suitable applications to mitigate the attack. The difference to our work is that we deal with High-Rate ADDoS which although have a higher traffic rate than Low-Rate ADDoS, they volume is insignificant compared to Amplification attacks.

6. Conclusions and Future Work

This paper proposes a novel defense, called SHADE, for mitigating High-Rate Application-Layer DDoS (ADDoS) attacks, namely the GET-Flooding attack, using selective strategies in SDN networks. By exploiting our existing tool SeVen for mitigating Low-Rate DDoS attacks and the fact that SDN controllers have a global view of the network, SHADE can mitigate High-Rate ADDoS attacks already at the entrance of the network where the target application is located. We demonstrated the effectiveness of SHADE by simulation using Mininet. In average, we saw an increase of 220% on availability, from 19% to 61%, and a decrease of 70% in Time to Service, from 5.4s to 1.6s. We also observed that 70% of the traffic generated by the attacker is dropped by SHADE already at the entrance of the network where the application is.

There are many directions for future work. We are investigating the effect of SHADE when suffering a combination of Low-Rate and High-Rate ADDoS attacks. We are also considering other selective strategies based on different parameters which may be obtained from the web-server, such as trusted users, etc.

We are also intending to implement SHADE in actual SDN networks. Although Mininet simulations is the standard for validating solutions in the SDN literature, it would be better to have actual experiments carried out on a actual SDN network.

Finally, we are also investigating how selective strategies can be used to mitigate SDN vulnerabilities such as the Rule Inundation attack that overloads switches with a large number of rules.

References

[BMP10] R. Braga, E. Mota, and A. Passito. Lightweight DDoS flooding attack detection using nox/openflow. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 408–415, Oct 2010.

[DNF14] Yuri Gil Dantas, Vivek Nigam, and Iguatemi E. Fonseca. A selective defense for application layer DDoS attacks. In *IEEE JISIC 2014*, pages 75–82, 2014.

[Foc14] NS Focus. http://www.prnewswire.com/news-releases/2014-mid-year-ddos-threat-report-documents-high-volume-high-rate-attacks-o html. 2014.

[FTSB15] Seyed Kaveh Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: Flexible and elastic ddos defense. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, pages 817–832, 2015.

[GHM$^+$05] Albert G. Greenberg, Gísli Hjálmtýsson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey G. Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4d approach to network control and management. *Computer Communication Review*, 35(5):41–54, 2005.

[HSE14] S. Hommes, R. State, and T. Engel. Implications and detection of dos attacks in openflow-based networks. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 537–543, Dec 2014.

[LDF$^+$ed] Marcílio Lemos, Yuri Gil Dantas, Iguatemi E. Fonseca, Vivek Nigam, and Gustavo Sampaio. A selective defense for mitigating coordinated call attacks. Submitted.

[LOI13] LOIC. `http://sourceforge.net/projects/loic/`. 2013.

[Min15] Mininet. `http://mininet.org/`. 2015.

[OLL14] Y.E. Oktian, SangGon Lee, and HoonJae Lee. Mitigating denial of service (dos) attacks in openflow networks. In *Information and Communication Technology Convergence (ICTC), 2014 International Conference on*, pages 325–330, Oct 2014.

[Ope15] OpenFlow. `https://www.opennetworking.org/Openflow`. 2015.

[rudy13] r-u-dead yet. `https://code.google.com/p/r-u-dead-yet/`. 2013.

[ryu15] ryu. `http://osrg.github.io/ryu/`. 2015.

[Sie15] Siege. `https://www.joedog.org/siege-home/`. 2015.

[slo13a] slowloris. `http://ha.ckers.org/slowloris/`. 2013.

[slo13b] slowread. `https://code.google.com/p/slowhttptest/`. 2013.

[SYPG13] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, CCS '13, pages 413–424, New York, NY, USA, 2013. ACM.

[WXG15] Haopei Wang, Lei Xu, and Guofei Gu. Floodguard: A dos attack prevention extension in software-defined networks. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 239–250, June 2015.

[ZJT13] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys and Tutorials*, 15(4):2046–2069, 2013.