Flowfence: a denial of service defense system for software defined networking

Conference Paper · October 2015

DOI: 10.1109/GIIS.2015.7347185

CITATIONS
10

READS
93

4 authors:

Andrés Felipe Murillo
Los Andes University (Colombia)
11 PUBLICATIONS **67** CITATIONS
SEE PROFILE

Sandra Rueda
Los Andes University (Colombia)
35 PUBLICATIONS **221** CITATIONS
SEE PROFILE

Diogo Menezes
Universidade Federal Fluminense
35 PUBLICATIONS **238** CITATIONS
SEE PROFILE

Otto Carlos M. B. Duarte
Federal University of Rio de Janeiro
321 PUBLICATIONS **2,343** CITATIONS
SEE PROFILE

FlowFence: A Denial of Service Defense System for Software Defined Networking

Andrés Felipe Murillo Piedrahita and Sandra Rueda
Systems and Computing Engineering Department
School of Engineering
Universidad de los Andes, Colombia
Email: {af.murillo225, sarueda}@uniandes.edu.co

Diogo M. F. Mattos and Otto Carlos M. B. Duarte
Grupo de Teleinformatica e Automação
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
Email: {menezes, otto}@gta.ufrj.br

***Abstract*—Most Denial of Service (DoS) attacks intend to generate a traffic pattern that is indistinguishable from legitimate traffic, making it hard to detect an attack. Conventional defenses for these attacks are not scalable, are slow to react or introduce an overhead to each routed packet. In this paper, we present FlowFence, a lightweight and fast denial of service detection and mitigation system for Software Defined Networking (SDN). The FlowFence architecture includes routers running daemons to monitor the average occupation of their interfaces to detect congestion conditions, and an SDN controller that coordinates bandwidth assignment of controlled links. The controller limits the flow transmission rate along a path to prevent users' starvation. The mitigation procedure of starvation state allocates an average bandwidth, while flows exceeding the mean are penalized. The penalization is proportional to the difference between the fair limit and the current bandwidth usage. A system prototype was implemented and evaluated in the Future Internet Testbed with Security (FITS). The results show that the proposal avoids users' starvation of network resources without adding much overhead in the network.**

I. Introduction

Denial of Service (DoS) attacks are the most important Internet threat. During the last years, large scale DoS attacks have been presenting a growing pattern in their volumes, reaching 100 Gb/s in 2010 and 400 Gb/s in 2014 [1]. These volumes can compromise the main Internet links, routers, and services. They may also cause interruptions of multiple services, including critical infrastructures, like Smart Grids [2], with huge financial damages. A DoS attack becomes successful when malicious users intentionally consume enough resources that deprives the resources of a target victim, which are aimed at providing services to the legitimate users. Sophisticated attacks mimic legitimate traffic, making them difficult to detect and to prevent. Distributed Denial of Service (DDoS) attacks use geographically distributed machines to strength the attack, achieving a very high concentration of requests at the destination victim, as well as in the last communication links close to such destination. The geographical distribution of attackers hides their location.

Source-based DoS detection is close to the attacker, but it is not a trivial task in a DDoS, because the number of requests generated by every attacking machine may be very low. Destination-based DoS detection uses mechanisms to detect and to block traffic at the destination. Nevertheless, the use of defenses at the destination does not avoid network resource consumption. The hybrid DoS detection combines close to the destination detection and mechanisms to block traffic at network routers. In this way, it is possible to reduce the concentration of false requests at the victim and to control network resource consumption [3]. The hybrid mechanisms, however, work in a distributed way, which could be slow for critical applications, or could require additional headers in the network packets, degrading network performance.

978–1–4673–7707–2/15/$31.00

In this paper, we propose FlowFence, a congestion avoidance mechanism system for mitigating denial of service on Software Defined Networking (SDN). Software Defined Networking employs a logically centralized controller that knows the global network view, monitors the current status of a network, and configures the switches to process, to forward, and to discard packets [4]. FlowFence applies a simple bandwidth control to mitigate DoS impact without requiring the complexity of additional headers in network packets. The FlowFence architecture is composed of network routers and an SDN controller that monitors the usage level of their interfaces. When a congestion state is detected, the router notifies the controller and the controller sends commands back to routers to limit bandwidth usage on the congested interfaces. Flows with bandwidth consumption higher than a fair usage are penalized through the application of a reduction that is proportional to the difference between current and fair usages. A prototype of FlowFence was implemented in the Future Internet Testbed with Security (FITS) [5]. The prototype was evaluated and the results show that FlowFence avoids starvation of legitimate users in presence of denial of service attacks with high volume of flooding packets.

The rest of paper is organized as follows. Section II presents the related work. Section III present the FlowFence design, while Section IV describes FlowFence architecture and implementation details. The experiments and results of the FlowFence evaluation are presented in Section V. Section VI concludes the paper.

II. Related Work

Yan and Yu argue that, although Software Defined Networking is a target of DDoS by itself, the logically centralized control of the SDN brings new possibilities to defeat DDoS, especially in cloud computing environments [6]. Software Defined Networking technology can be helpful to develop

improved security mechanisms, because it can dynamically and automatically reconfigure the network when particular security anomalies are detected. Moreover, using SDN a reconfiguration may be triggered based not only on single events but also on a global state vision of the network, which increases effectiveness.

Avoiding address spoofing is an important step against DoS attacks. Kwon *et al.* propose the BGP-based Anti-Spoofing Extension (BASE) that is a mechanism for avoiding address spoofing on Software Defined Networking [7]. The main idea of BASE is to mark packets with an identification of its path. If the packet has a mark that is not in accordance with its address, the packet is recognized as spoofed address. The marking values are distributed through BGP. BASE takes advantage of the logically centralized control of SDN to process marking values that identifies paths in a SDN Autonomous System. FlowFence, on the other hand, does not focus on identifying who is the DoS attacker on the network and intends to be much more simple and not mark packets.

Silva *et al.* [8] propose a Software Defined Networking (SDN) architecture, using Pyretic[1] and Resonance[2], that performs network reconfigurations in the presence of DoS attacks and intrusion attempts. The system uses Pyretic to detect events at SDN controllers and to activate policies to respond against these events. In the case of DoS attacks, a simple threshold was established to trigger the policy. FlowFence uses a similar approach as it trigger the congestion event when it detects congested output interfaces, but FlowFence applies a progressively stronger bandwidth control over the flows instead of dropping all the packets in a single step.

Lim *et al.* [9] propose an SDN scheme for blocking DoS attacks that rely on botnets and do not spoof IP addresses. The scheme includes a Blocking Application running on top of an SDN controller and a set of protected servers that communicate with the controller through secure communication channels. Each server monitors traffic and notifies the Blocking Application when it detects a DoS attack. The application handles a pool of IP public addresses and assigns a new IP address to the server that notified the condition. The controller provides arriving clients with the new address of the server as the address for the service, and the controller also tears down old connections. The new IP should be provided to clients imposing a high computational barrier for bots, like CAPTCHAs. The switches in the network use SDN redirection functions to migrate connections to the new address. Clients that continuously try to connect to the old address are marked as bots and their packets are dropped.

Bedi *et al.* [10] propose a DoS mitigation mechanism based on active queue management that provides fair usage to legitimate flows and drops all packets from malicious flows. The mechanism uses Deterministic Fair Sharing (DFS) to assign each flow a share of the router buffer; a share is calculated based on current buffer capacity, number of active flows and queue length. Also, the mechanism uses a marking probability (Pm) to identify malicious flows; this probability increases as a flow behaves more unfairly. Flows marked as malicious have all their packets dropped. Bit-rate statistics are maintained at the entrance interfaces of the router and marked flows are unmarked as malicious as they return to use a fair share of the buffer capacity. The main idea of FlowFence is simplicity and it does not force the routers to perform processing operations on each incoming packet to avoids processing overhead.

AVANT Guard [11] presents two extensions to the SDN architecture to improve resiliency: Connection Migration and Actuating Triggers. Connection Migration is a protection mechanism against SYN Flood attacks; the mechanism proxies TCP SYN requests and uses SYN cookies to classify legitimate SYN requests, only legitimate requests are authorized and migrated to the real target. Additionally, AVANT Guard introduces Actuating Triggers; a trigger is an event that may happen and may be detected at the data plane, such as an exceeding rate of SYN requests to a server. When an event is detected within the data plane statistics it may trigger an event to the controller or insert a flow rule into a specified flow table to mitigate the attack. AVANT GUARD is designed to protect the SDN itself.

Braga *et al.* propose an Intrusion Detection System as an SDN application [12]. The IDS is based on Self Organizing Maps (SOM); it receives information from the controller and uses the SOM to identify flows belonging to a DDoS. The IDS presents the best results using the following four flow metrics: mean of packets per flow, mean of bytes per flow, mean of flow duration, and percentage of paired flows. IDS based on classifiers are sensitive to training data sets and, thus, they may not identify unknown attack types.

NetFence [13] is a closed loop congestion control architecture that mitigates DoS impact. In this architecture, routers detect congestion in their output interfaces and send authenticated feedback information to the routers that are close to the congestion source to reduce the DoS impact. To safely exchange this information, the authors propose an extra layer between the IP and TCP layers [14]. FlowFence bases on two ideas of their work: setting a threshold of fair bandwidth usage as a classifying metric and applying bandwidth control to mitigate DoS impact. Nevertheless, FlowFence uses the SDN paradigm, hence, it does not need to modify the TCP/IP stack as NetFence proposes. Additionally, FlowFence does not require feedback information for every packet, neither the use of a cryptographic hash for every packet. A header with a cryptographic hash increases the processing time.

Mattos and Duarte propose XenFlow [15], [16], a mechanism to guarantee QoS in virtual networks. The mechanism runs on top of the Xen hypervisor and OpenFlow, and it implements network resource control using queue management. First, each virtual machine must define its QoS parameters. Afterwards, an application on the hypervisor reads these parameters and associates them to a set of queues, as a way to guarantee the minimum amount of resources requested by each virtual machine. If the physical machine has available resources after the initial resource assignment, the remaining resources are distributed proportionally to the defined QoS parameters. Although XenFlow and FlowFence have different goals, FlowFence uses queue management, as proposed by XenFlow to control bandwidth assignment.

[1]Pyretic is a framework for defining high level policies in SDN - http://www.frenetic-lang.org/pyretic/

[2]Resonance is a security-driven controller that implements Pyretic http://resonance.noise.gatech.edu/

III. The FlowFence Design and Architecture

A. Attacker Model

A network flow can affect performance of other flow intentionally or not, due to malicious usage or unexpected network traffic. We classify a malicious behavior when users perform activities that interrupt or degrade the performance of other flows on the network. For unexpected network traffic, we denote every flashcrowd, or similar behavior, that may congest the network suddenly. Both behaviors are harmful and cause a denial of service. Therefore, for sake of simplicity, we define all the flows that perform such behaviors as attacking flows. Attacking flows are launched by attackers that may deploy one of the following behaviors.

1) Flood-Based Attacks: The attacker can generate flood-based DoS attacks that try to exhaust bandwidth resources nearby the victim and forwarding capacities of the routers. The model does not consider DoS attacks that exploit vulnerabilities, like TCP-SYN DoS attacks, and other vulnerability-based attacks. It is also assumed that the attacker does not compromise the SDN controller or routers.

2) Malicious Traffic Mimicking Legitimate Traffic: It is assumed that the attacker can launch floods with traffic that is indistinguishable from legitimate traffic; for this reason it is not possible to differentiate legitimate from malicious traffic.

B. FlowFence Objectives

FlowFlace aims to avoid starvation of legitimate users, even under a high volume non-differentiable traffic. As a second goal of FlowFence, we design a simple and responsive system that uses unmodified protocol stack. Hence, FlowFence must not require a modification in the traditional TCP/IP protocol stack used in TCP/IP networks. This type of modification would greatly impact applicability of a proposal. At last, another main goal is to fast respond against any congestion condition that is verified on the network.

C. The FlowFence Defense Procedure

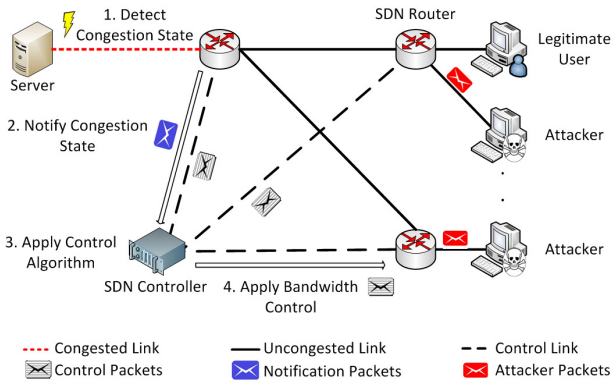

Figure 1. The FlowFence defense procedure: 1) Routers detect congestion at their output interfaces; 2) notify the SDN controller; 3) the controller makes decisions; and 4) sends commands back to the routers to control bandwidth usage.

The FlowFence architecture is based on OpenFlow routers using POX SDN controller. The routers monitor their output interfaces to detect congestion conditions and notify to the controller. The controller receives congestion notifications and flows statistics, and sends commands to the congested routers to control bandwidth usage of their flows. Thus, when a router notifies about a congested interface, the SDN controller sends commands to the notifying router, as well as to all routers forming the forwarding packet path to that router, to control bandwidth assignment for all flows using the congested interface. Figure 1 presents the FlowFence procedure. When no congestion condition exists, FlowFence only monitors the flows that are traversing the network. Bandwidth is granted based on flow behavior. If the flow transmitted bit-rate is less than an estimated fair share, then the flow is classified as a well-behaved flow and receives bandwidth equal to its estimated transmitted bit-rate. In this paper a fair share, for simplicity, is defined as the ratio between the link capacity and the number of flows using that link. If the flow transmitted bit-rate is higher than a fair share threshold, then the flow receives a fair share minus a penalization that is calculated based on the difference between fair usage and its current usage rate. Finally, remaining bandwidth is equally shared between well behaved flows.

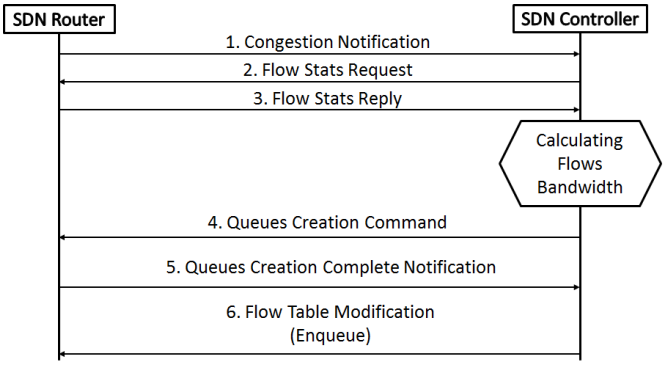

Figure 2. Time sequence of FlowFence messages to control bandwidth usage of congested interfaces.

Figure 2 shows the message exchange between an SDN router and the controller to perform bandwidth control. All the messages travel over a secure channel, as the OpenFlow protocol specifies. In FlowFence, a Denial of Service condition is detected if the interface average-usage rate is higher than 80% [13]. To estimate the current interface usage, routers periodically capture samples of transmitted bytes per second and apply a sliding exponential window to estimate the interface average occupation. The use of an exponential sliding window avoids false-positive alarms.

The SDN controller keeps secure connections with the routers using the control plane interfaces. If a router detects a congestion condition, it sends a message to the controller. The notification message includes the interface capacity, the name, i.e., a unique identifier, and the type of the message, in this case, type is set to `notification`. Upon receiving a notification message, the controller checks the network topology, and requests statistics from every router that forwards flows passing through the congested link. The logically centralized SDN paradigm enables this operation and also allows the controller to rapidly react and communicate with every router somehow participating in the congestion. The key

idea is to perform an end-to-end congestion control, avoiding unnecessary link usage. The routers reply sending their flow statistics to the controller. These statistics include source and destination IP addresses, flow length, and flow duration.

After receiving flow statistics from a router owning a congested interface, the controller classifies the flows. To fast respond to this situation, we simply protect flows that use less bandwidth than a fair share of the capacity of the interface, and we assume as badly behaved flows and penalize those flows that use more bandwidth than a fair share. Therefore, if $bw_i > C_t/n$, then $flow_i$ is badly behaved, where bw_i stands for the bandwidth of $flow_i$, C_t is the capacity of the link and n is number of flows sharing the link. Therefore, the fair share strategy benefits low bandwidth flows, which cause low impact on the network and have short duration. A more detailed classification requires an Intrusion Detection System (IDS), which would increase FlowFence complexity and is considered out of the scope of this proposal.

Once flows have been classified, the controller sends to the router a command to create one queue for each classified flow, and assigns independent bandwidths to each queue, as bandwidth control is applied through queues. Bandwidth assigned to each queue depends on whether the associated flow was classified as well behaved. Well-behaved flows receive bandwidth according to:

$$bw_{\mathrm{i}} = bw_{ri} + (bw_{\mathrm{extra}}/n_{\mathrm{good}}), \quad (1)$$

where bw_{ri} is the bandwidth used by the $flow_i$, bw_{extra} is the remaining bandwidth after assigning bandwidth to all the flows, and n_{good} is the total number of well-behaved flows. The FlowFence implementation assigns bandwidth as follows. First, it assigns bandwidth to well behaved flows. Second, it assigns bandwidth to flows classified as badly behaved, which are penalized proportionally to their excess. Finally, it distributes remaining bandwidth among all the well-behaved flows. Flows classified as badly behaved flows receive bandwidth according to:

$$bw_{\mathrm{i}} = bw_r/n_{bad} - (1 - e^{(C_t/n) - bw_{ri}}) * \alpha * bw_{ri}, \quad (2)$$

where bw_{ri} is the bandwidth used by the $flow_i$, bw_r is the remaining bandwidth after granting bandwidth to well behaved flows, n_{bad} is the number of badly behaved flows identified at the router, C_t is the total interface capacity, n is the total number of flows, and α is a constant that the administrator set to determine the penalty aggressiveness. If α is equal to 0, then no penalty is applied and badly behaved flows will receive a fair share of the remaining bandwidth. If α is equal to 1, then the maximum exponential penalization is applied to badly behaved flows. The penalization was designed to apply a bigger control in offending flows whose bandwidth greatly exceeds the fair share usage.

IV. The FlowFence Prototype

The FlowFence prototype was implemented as a POX application in the SDN controller and a Python application in the forwarding devices. POX is an SDN controller[3] that supports OpenFlow version 1.0. The routers run the Open vSwitch[4] as a software for switching and routing. Open vSwitch supports the OpenFlow protocol and provides mechanisms for queue management and bandwidth control. The FlowFence Python application monitors interface usage, handles communication with the SDN controller, and applies bandwidth control.

All the experiments were performed in the Future Internet Testbed with Security (FITS), and controlled with the MAGI framework. FITS is an inter-university testbed developed by Brazilian and European universities [5]. FITS is a testbed based on the Xen hypervisor and on the OpenFlow switching. OpenFlow and Open vSwitch handle communication between the virtual machines. Prototype implementation and experimentation in a virtual environment take into account message processing and transferring delays, providing realistic results. Besides, we use the MAGI framework, developed by DeterLab [17], to create a highly controllable and replicable experiments in testbed environments.

In MAGI, each experiment follows a script, in which nodes run specific software, called agents. The script is used to synchronize the commands and events exchanged between the nodes participating in the experiment.

In the FlowFence experiments, each node was implemented as a virtual machine, and two physical machines hosted the virtual machines. In addition, legitimate clients run MAGI agents written to execute the Iperf[5] and Httperf[6] tools. Attackers run the MAGI Flooder agent, originally developed by DeterLab[7].

V. Experiments and Results

Figure 3 describes the topology used for the experiments. A dumbbell topology was selected because it is the worst-case scenario for the DoS defense provided by FlowFence, as legitimate and malicious clients are competing for the shared link to the server [18]. We assigned a capacity of 50 Mb/s to the shared link. The experiments use virtual machines to run clients, servers, routers, and the SDN controller. Two physical servers host the virtual machines and were connected using Gigabit Ethernet links. Each server is equipped with two Intel(R) Xeon(R) CPU X5690 3.47 GHz and 48 GB of RAM, and runs Debian Linux 3.2.0-4-amd64. The first server hosts the attacker nodes and the second one hosts the SDN routers, the controller, the legitimate client, and the server. Virtual machines were configured with a virtual CPU using one core, 256 MB of RAM, and running Debian Linux 3.2.0-4-amd64.

Three sets of experiments were performed. The first experiment measures the bandwidths received by a legitimate client during a flood attack, with and without the FlowFence defense, as a way of validating the prototype. The second experiment follows recommendations previously presented to test DoS defenses [18]; rather than using simple tests to

[3]POX is an open source SDN controller implemented in Python. Available at http://www.noxrepo.org/pox/about-pox/.

[4]Open vSwitch is a multilayer virtual switch. Available at http://www.openvswitch.org/

[5]http://sourceforge.net/projects/iperf/.

[6]http://www.hpl.hp.com/research/linux/httperf/.

[7]http://montage.deterlab.net/magi/.

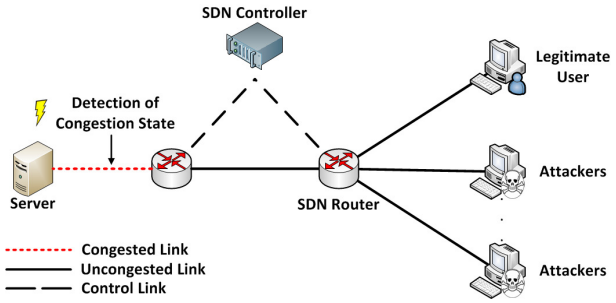

Figure 3. Dumbbell topology used in the experiments. All the participating nodes are virtual machines running on top of Xen Hypervisor. The legitimate user executes Iperf and Httperf tools and the attackers run the MAGI Flooder Agent.

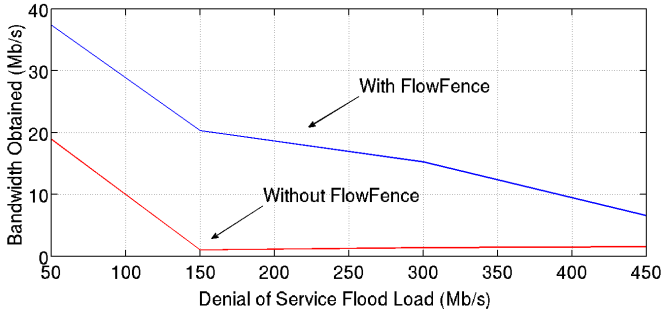

Figure 4. Results of the Iperf experiment. Using FlowFence a legitimate client receives more bandwidth than it would receive if FlowFence were not used. Under a total flooding of 450 Mbits/s, the legitimate client does not starve.

measure dropped packets or bandwidth, we used Httperf to measure the reply time for a client when requesting HTTP content under a DoS attack. This experiment evaluates the effectiveness of FlowFence in mitigating DoS flood attacks. The third experiment measures FlowFence reaction time.

All experiments were run for 60 seconds. DoS flood attack lasts during all the time. The legitimate client starts sending requests 30 seconds after the beginning of the experiment. Each attacker was configured to flood the link with a load of 50 Mb/s. The experiment scenario scales in number of attackers, and for each scenario, we run three rounds.

In the first experiment, Iperf was used to measure the bandwidth obtained by a legitimate client. The tool was configured to run tests during 30 seconds and deliver reports every second. Figure 4 presents the average bandwidth assigned to a legitimate client in the experiment. The results show that a legitimate client in an environment without FlowFence receives less bandwidth than it would access if FlowFence were not used. The performance of legitimate users degrades as the number of attackers increases. With three attackers and a load of 150 Mb/s, or more, a legitimate client ends up using a very low share of the bandwidth. When FlowFence is active, legitimate clients receive a larger share of bandwidth. The bandwidth share is reduced as the number of attackers increases, because they also receive a low amount of bandwidth. Nevertheless, even with 9 attackers generating a total load of 450Mb/s, which greatly exceeds the link capability, the legitimate client gets an assured share of bandwidth.

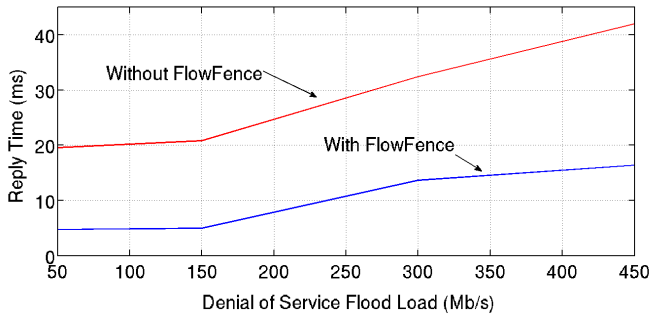

Figure 5. Results of the Httperf experiment. Using FlowFence a legitimate client requesting HTTP content receives a reply faster than it would happen if FlowFence was not present. The reply time slightly increases as the number of attackers and the volume of the DoS traffic increase.

It is noteworthy that this result is not the same as the theoretical bandwidth that the client should receive according to proposed sharing algorithm. It happens because the router still has to enqueue the incoming messages from both legitimate and malicious traffic. Also, it is important to note that as FlowFence is a hybrid DoS defense it protects the network elements and links, making it an effective defense against flood attacks that go from simple UDP floods to more sophisticated attacks, such as the Coremelt Attack [19].

In the second experiment, Httperf was used to measure reply time for a legitimate client when it requests HTTP content under a DoS attack. The size of the requested content was of 1KB. The tool was configured to try 10 requests per second and to perform a total of 100 requests. Figure 5 presents the results of this experiment, the average reply time for the 100 requests, with and without FlowFence.

The results show that reply time is greater when no defense is present, and reply time increases as the number of attackers and the volume of flooding increase. When the volume is 450 Mb/s, the time is approximately 40ms. It is important to note that the HTTP content is very small. The goal of this experiment is to measure the time that it takes for a legitimate client to send a small request and receive an answer over a very congested link. With FlowFence, the time slightly increases, reaching around 15 ms, when the flood load is 450 Mb/s. The low increment is explained because the legitimate client receives a larger share of bandwidth and the bandwidth control is applied in both routers in the topology. Also, with FlowFence, no additional headers were needed. All the notifications and commands are sent through the control plane. All these messages are handled at the application layer and travel using the traditional TCP/IP stack. This experiment shows that FlowFence avoids starvation of legitimate clients even under heavy DoS attacks.

The last experiment measures FlowFence response time. Response time is relevant because if a defense takes too long to detect and mitigate a lack of network resources, many legitimate transactions may fail. This experiment deploys a scenario in which 9 attackers, generating a 450Mb/s load, are present. Using timers in the applications, the time required for each step that FlowFence takes to detect and control the DoS was measured. Table I shows the results of the test. Results show that it takes 7 seconds, at all, for FlowFence to detect

and control the DoS attack. The activity that requires more time is the DoS detection, which takes up to 4800 ms. It happens because the router is sampling the output interface occupation level per second. The sliding window has a width of four samples to prevent false positives caused by small bursts of legitimate traffic. It only takes 10 ms for the controller to collect the router flow statistics. The time spent to calculate the bandwidth is 30ms. These short periods of time show that FlowFence does not impose a high computational load in the controller, which could lead to scalability problems. Additionally, the queue setup time takes 330 ms, this is caused by the execution of several commands, in Open vSwitch, to establish and assign bandwidth to each queue. Finally, it takes 1829 ms for the controlling actions to have effect on the bandwidth assignment, and to be perceived by the legitimate client. This is a reasonable time, as the routers still forward all packets that are flooding the interface, before the defense starts applying bandwidth controls.

Table I. TIME REQUIRED FOR EACH FLOWFENCE STEP TO DETECT AND REACT TO A DOS FLOODING ATTACK.

Activity	Required Time (ms)
DoS Detection	4800
Flow stats collection	10
Bandwidth calculation and queues command	30
Queue setup time	330
Flow redirection time	1
Bandwidth control effects	1829
Total time	7000

VI. CONCLUSION

In this paper, we propose the FlowFence, a congestion avoidance system for denial of service mitigation on Software Defined Networking. FlowFence identifies congestion conditions by monitoring the usage of output interfaces at the network routers. FlowFence fast reacts to a congestion scenario, using bandwidth control over every flow that traverses the congested interfaces. To keep FlowFence simple, it does not require additional headers in the conventional TCP/IP stack and it does not employ intrusion detection system to classify malicious flows. The use of the SDN logically centralized control allows identifying the flow path and discarding packets close to their origin, avoiding the unnecessary use of network bandwidth resource. The simple system quickly detect a congestion scenario and mitigate the impact caused by badly behaved flows. FlowFence penalizes with lower bandwidth shares those flows that exceed a fair share of the resource; the penalization is proportional to the difference between the actually used bandwidth and the estimated fair bandwidth usage. The results obtained through experiments with our developed prototype show that FlowFence is simple, fast and efficient, avoiding congestion scenarios with a low response time. As a future work, we will consider larger experiment topologies, and we will also port FlowFence application from the POX to the Beacon controller, which outperforms POX.

ACKNOWLEDGMENTS

The authors would like to thank CNPq, CAPES, FAPERJ, and Colciencias for their financial support.

REFERENCES

[1] *Worldwide Infrastructure Security Report: 2014 Report*, Arbor Networks, Jan. 2015, accessed in July 2015. [Online]. Available: http://www.arbornetworks.com/resources/infrastructure-security-report

[2] W. Wenye and L. Zhuo, "Cyber Security in the Smart Grid: Survey and Challenges," *Comp. Net.*, vol. 57, no. 5, pp. 1344–1371, Apr. 2013.

[3] S. Zargar, J. Joshi, and D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[4] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[5] I. M. Moraes, D. M. F. Mattos, L. H. G. Ferraz, M. E. M. Campista, M. G. Rubinstein, L. H. M. Costa, M. D. de Amorim, P. B. Velloso, O. C. M. B. Duarte, and G. Pujolle, "FITS: A Flexible Virtual Network Testbed Architecture," *Computer Networks*, vol. 63, pp. 221 – 237, 2014.

[6] Q. Yan and F. Yu, "Distributed denial of service attacks in software-defined networking with cloud computing," *Communications Magazine, IEEE*, vol. 53, no. 4, pp. 52–59, Apr. 2015.

[7] J. Kwon, D. Seo, M. Kwon, H. Lee, A. Perrig, and H. Kim, "An incrementally deployable anti-spoofing mechanism for software-defined networks," *Computer Communications*, vol. 64, pp. 1 – 20, 2015.

[8] J. Silva Delgado, D. Mendez Peñuela, L. Morales Medina, and S. Rueda Rodriguez, "Automatic Network Reconfiguration Because of Security Events," in *Proceedings IEEE Colombian Conference on Communications and Computing 2014*, Colombia, Jun. 2014, pp. 1–6.

[9] S. Lim, J. Ha, H. Kim, Y. Kim, and S. Yang, "A SDN-oriented DDoS Blocking Scheme for Botnet-based Attacks," in *2014 Sixth International Conference on Ubiquitous and Future Networks*, China, Jul. 2014.

[10] H. Bedi, S. Roy, and S. Shiva, "Mitigating Congestion-based Denial of Service Attacks with Active Queue Management," in *2013 IEEE Global Communications Conference*, Atlanta, USA, Dec. 2013, pp. 1440–1445.

[11] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks," in *The 2013 ACM SIGSAC Conference on Computer & Communications Security*, Germany, Nov. 2013, pp. 413–424.

[12] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow," in *IEEE 35th Conference on Local Computer Networks (LCN)*, Denver, USA, Oct. 2010, pp. 408–415.

[13] X. Liu, X. Yang, and Y. Xia, "NetFence: Preventing Internet Denial of Service from Inside Out," in *Proceedings of the ACM SIGCOMM 2010 conference*, New Delhi, India, Aug. 2010, pp. 255–266.

[14] X. Liu, X. Yang, D. Wetherall, and T. Anderson, "Efficient and Secure Source Authentication with Packet Passports," in *II Conference on Steps to Reducing Unwanted Traffic on the Internet*, USA, Jul. 2006.

[15] D. M. F. Mattos and O. C. M. B. Duarte, "XenFlow: Seamless migration primitive and quality of service for virtual networks," in *IEEE Global Communications Conference (GLOBECOM 2014)*, Dec. 2014.

[16] D. M. F. Mattos, L. H. G. Ferraz, and O. C. M. B. Duarte, "Virtual machine migration," in *Cloud Services, Networking and Management*, N. L. S. da Fonseca and R. Boutaba, Eds. Hoboken, EUA: Wiley-IEEE Press, Apr. 2015.

[17] J. Mirkovic, T. Benzel, T. Faber, R. Braden, J. Wroclawski, and S. Schwab, "The DETER Project: Advancing the Science of Cyber Security Experimentation and Test," in *IEEE International Conference on Technologies for Homeland Security (HST)*, 2010.

[18] J. Mirkovic, S. Fahmy, P. Reiher, and R. Thomas, "How to Test DoS Defenses," in *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, Mar. 2009, pp. 103–117.

[19] A. Studer and A. Perrig, "The coremelt attack," in *Computer Security - ESORICS 2009*, M. Backes and P. Ning, Eds. Springer Berlin, 2009, vol. 5789, pp. 37–52.