Computer Networks 92 (2015) 89–100

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

OPERETTA: An OPEnflow-based REmedy to mitigate TCP SYNFLOOD Attacks against web servers

Silvia Fichera, Laura Galluccio*, Salvatore C. Grancagnolo, Giacomo Morabito, Sergio Palazzo

University of Catania, Dipartimento di Ingegneria Elettrica Elettronica e Informatica, v.le A. Doria 6, Catania 95125, Italy

ARTICLE INFO

Article history:
Received 23 October 2014
Revised 21 July 2015
Accepted 27 August 2015
Available online 25 September 2015

Keywords:
OpenFlow
DDoS
SDN

ABSTRACT

TCP SYNFLOOD attacks are a type of Distributed Denial of Service (DDoS) attacks usually carried out against web servers. TCP SYNFLOOD rely on the normal TCP Three-Way Handshake mechanism to consume resources on the targeted server. In this way server resources are blocked and the server is made unresponsive. To this purpose, the attacker sends multiple fake SYN packets as if it wants to set several TCP connections up, but then it does not finalize the Three-Way Handshake. In this way, it blocks the resources of the attacked server, uselessly. In traditional networks, these attacks have been counteracted by means of firewalls and intrusion detection schemes. However, these solutions are not effective since can be violated. Software Defined Networks (SDNs) offer new features like network programmability which make solutions to TCP SYNFLOOD attacks more effective. In fact, in SDN networks intelligence for counteracting security menaces can be moved to a single network element, i.e., the Controller, which has complete information about the network and is in the best condition to identify ongoing attacks. However, in this way TCP SYNFLOOD attacks can turn into attacks to the Controller, which becomes a unique point of failure for the network. In this paper we propose OPERETTA, an OPEnflow-based Remedy to TCP SYNFLOOD Attacks. OPERETTA is implemented in the Controller which manages incoming TCP SYN packets and rejects fake connection requests. The OPERETTA protocol works in heterogeneous networks, as it can be implemented not only on a centralized Controller, but also on delocalized Controllers available in the access routers at the users' premises. OPERETTA has been tested using MININET and to this purpose prototypes of the relevant Control Plane functions have been implemented starting from the POX Controller. Numerical results show that OPERETTA achieves good performance in terms of resilience to TCP SYNFLOOD attacks and low level of CPU and memory consumption.

1. Introduction

TCP SYNFLOOD attacks are a type of Distributed Denial of Service (DDoS) attack that rely on the normal TCP Three-Way Handshake mechanism to consume resources on the targeted server. In this way server resources are blocked and the server is made unresponsive. As a consequence the attacked server is overwhelmed and cannot support the requested service.

The TCP SYNFLOOD attack was identified by Ziegler in 1994 [1]. Analyzing statistics on this type of attacks, it has been observed that it represents the 90% of the DoS attacks worldwide [2]; so it is a widespread and critical menace to be faced. In the TCP SYNFLOOD attack it is assumed that the victim node, upon receiving a connection request in the form of a SYN segment from a remote user, enters a listening state

* Corresponding author. Tel.: +39 095 738 2384.
E-mail addresses: silvia.fichera@dieei.unict.it (S. Fichera), laura.galluccio@dieei.unict.it (L. Galluccio), salvatore.grancagnolo@dieei.unict.it (S.C. Grancagnolo), giacomo.morabito@dieei.unict.it (G. Morabito), sergio.palazzo@dieei.unict.it (S. Palazzo).

http://dx.doi.org/10.1016/j.comnet.2015.08.038
1389-1286/

where its resources are frozen waiting for the confirmation ack from the initiating node. The victim waits for a certain amount of time and then gives up. If the victim's buffer allows to enqueue multiple packets, this mechanism will exhaust the victim available resources.

TCP SYNFLOOD attacks have been widely addressed in the literature as discussed in [3].

Methodologies used to counteract such a menace typically employ firewalls and intrusion detection schemes which can be ineffective and can be violated.

In SDN networks the decoupling between the data plane and the Control Plane allows to set up more effective solutions to TCP SYNFLOOD attacks. In fact, in these networks, the intelligence for counteracting security menaces can be moved to a single network element, i.e., the OpenFlow Controller, which has complete information about the network and is in the best situation to identify ongoing attacks. However, in this way TCP SYNFLOOD attacks can turn into attacks to the Controller, which becomes a unique point of failure for the network.

In this paper we propose OPERETTA, an OPEnflow-based Remedy to TCP SYNFLOOD Attacks. OPERETTA is implemented in the Controller which manages incoming TCP SYN packets and rejects fake connection requests. According to a traditional SDN architecture, OPERETTA logically consists of a single Controller, although physically OPERETTA can also be employed in a decentralized manner where multiple Controllers are available in the access routers at the users' premises to run control operations. This delocalization of the Controller function into multiple Controllers also achieves the objective of higher system reliability since, in case of DoS attacks, only a portion of the network, and not the entire network, is involved in the attack. OPERETTA has been tested using MININET and to this purpose prototypes of the relevant Control Plane functions have been implemented starting from the POX Controller. Numerical results show that OPERETTA achieves good performance in terms of resilience to TCP SYNFLOOD attacks and low level of CPU and memory usage.

The rest of this paper is organized as follows. In Section 2 we discuss the background literature in the field. In Section 3 we describe OPERETTA and in Section 4 we discuss the protocol performance. Finally, in Section 5 conclusions are drawn.

2. Background

In this section we provide a brief overview of the relevant literature in the field. More specifically, in Section 2.1 we discuss Denial of Service attacks in general and in Section 2.2 we focus on the TCP SYNFLOOD attack in particular. In Section 2.3 we recall Software Defined Networks with special focus on OpenFlow and in Section 2.4 we illustrate the few solutions to TCP SYNFLOOD attacks proposed so far for SDNs.

2.1. Denial Of Service

A Denial of Service (DoS) attack is an attempt made by a malicious user to compromise the regular network functioning. When this attempt comes from a group of hosts in a network coordinated by a certain malicious user, it is usually referred to as Distributed Denial of Service (DDoS). This is a relevant menace that has to be dealt with in the network.

A widespread example of DDoS attack is the one where an attacker infects vulnerable users by building a botnet. Accordingly, each infected user will send a massive amount of packets to the victim to exhaust its resources and make the latter unavailable to the legitimate users. However, different types of DDoS attacks can be considered [4]:

- Based on the *degree of automation* by distinguishing between manual, semi- automatic and automatic DDoS attacks.
- Based on the *exploited weaknesses* by distinguishing between the different weaknesses exploited to deny the service of the victim to its clients.
- Based on *source address validity* depending on whether IP spoofing for the source address is used or not.
- Based on *attack rate dynamics* depending on whether, during the attack, the participating agent transmits at constant rate or variable rate.
- Based on the *persistence of agent set* depending on whether the agent set is persistent (and thus can be traced) or not.
- Based on the *victim type* by distinguishing if the victim is the application, the host, the resource, the network or the infrastructure.
- Based on the *impact on the victim* by distinguishing between disruptive and degrading attacks.

It is evident that multiple types of attacks belonging to the above illustrated classes can be even combined to set up a more aggressive menace and stress the vulnerabilities of the network. Solutions to DDoS attacks typically rely on the use of firewalls to block malicious traffic [3]. Firewall action however should be combined with Intrusion Detection Systems (IDS) which collect information and send it to the analyzer to identify malicious users. As an example in [5] a TCP SYNFLOOD detection mechanism is activated which is based on the identification of an adaptive threshold taking into account the average value of the SYN packets received.

However use of firewalls does not guarantee immunity to DDoS attacks. In fact, commonly the firewall mechanism relies on identification of legitimate/malicious source IP addresses or source TCP ports. By using the IP spoofing or changing the source TCP port at each attack, the attacker can cheat the firewall, so easily bypassing the defense mechanism.

Accordingly, alternative solutions to the problem of DDoS attacks should be identified.

In the following we will focus on a specific class of such attacks, i.e., TCP SYNFLOOD attacks.

2.2. TCP SYNFLOOD attacks and countermeasures

The TCP SYNFLOOD attacks, as discussed in [6], represent one of the most common type of network attacks. The mechanism behind this, is the following.

In the TCP SYNFLOOD attack it is assumed that the victim node, upon receiving a connection request in the form of a SYN segment from a remote user, enters a LISTEN state. This state can be unfortunately entered also when none or only part of the IP address is received. If a malicious node

issues a SYN packet to request a connection to a victim using a spoofed source address, the corresponding SYN ACK reply from the victim will be discarded by the network. In fact, usually the spoofed IP address belongs to a host that is either unreachable or has an anomalous response to the SYN ACK segment from the victim. If a spoofed address belongs to a normal operating node, the latter will reply with RST segment to close the unexpected and undesired connection.

After entering the LISTEN state, the victim keeps this half open connection state waiting for an ACK as a response to the SYN ACK it sent. The victim waits for a certain amount of time and then gives up. If the victim's buffer allows to enqueue multiple packets, this mechanism will exhaust the victim available resources.

TCP SYNFLOOD attacks have been widely addressed in the literature as discussed in [3].

TCP connections are established by exploiting the well known Three-Way Handshake methodology. A connection initiator sends a SYN packet to request the setup of the connection to the peer entity. The latter answers back using a SYN ACK packet and keeps waiting for the ACK from the connection initiator. In this condition a connection queue of finite size at the destination host keeps track of connections waiting to be completed. This queue typically empties quickly since the ACK is expected to arrive in short time [7]. An attacker can simply not send the related ACK and/or keep generating TCP SYN packets using IP spoofing. In both cases the connection establishment process will never complete and this incomplete connection opening will consume server resources. If a large number of malicious coordinated users send fake TCP SYN packets, the destination will freeze a lot of resources resulting in a Denial of Service to legitimate traffic.

Two main types of solutions to TCP SYNFLOOD attacks can be envisaged [8] and differ in the type of defense action.

A first type of solutions implies modifications in the end-host TCP implementation so as to change algorithms and data structures used for connection lookup and establishment. As an example, this type of approaches implies variations in the SYN-RECEIVED timer, SYN caches, or SYN cookies [9].

More effective and complex solutions use a combination of individual defense techniques among those listed above.

Another class of approaches aimed at solving TCP SYNFLOOD attacks requires a network action meaning that these schemes either reduce the probability to incur into favorable conditions for executing the malicious attack or use additional network middle-boxes, such as filters, active monitors, or proxies to isolate network servers [10].

The Stream Control Transmission Protocol (SCTP) [11] is a transport-layer protocol proposed and standardized by IETF that can be used on top of IP networks for end-to-end communications. SCTP is similar to TCP in many ways but exhibits additional advantages. In fact it supports unicast connection-oriented communication and provides reliable transport with in-sequence packet delivery and rate-adaptive congestion control. As compared to TCP, SCTP is stream-oriented thus being able to handle multiple simultaneous and multiplexed streams rather than a single stream as supported by TCP. Also multihoming is guaranteed by SCTP. Concerning security issues, SCTP is more immune to man-in-the middle attacks since it employs a cookie mechanism during the initial connection procedure which also avoids the connection half opening and thus the TCP SYNFLOOD attacks. However, SCTP also exhibits some drawbacks related to the need for changes in the client server applications and transport stack of the end nodes. Moreover, SCTP was proposed when the TCP protocol was still well established so any application would require to be modified to support SCTP. A side condition of this is also that huge companies such as Microsoft do not plan to integrate SCTP in their frameworks thus hampering its diffusion. Finally, SCTP is not compliant with NAT which is indeed widespread among the residential users that access the Internet through their ISPs networks. In this paper instead, we aim at proposing a methodology for coping with SYNFLOOD attacks which is compatible with traditional TCP and requires no modifications at the client and server sides. Moreover the use of SDN approaches has been accepted as a promising approach to networking by big companies, like Microsoft [12]. In the next section we will see how the use of innovative networking paradigms, such as SDN, allows to find new strategies to counteract SYNFLOOD attacks.

2.3. Software defined networks

The evolution of the Internet paradigm in the recent years is bringing an innovation in the traditional *ossified* IP paradigm towards a software-oriented approach. More specifically, Internet is intrinsically a hardware dependent architecture, so it lacks in flexibility in resource allocation and programmability in the network devices. To gain more flexibility, the SDN paradigm has been proposed. SDN makes the network programmable [13]. It allows to modify network structure and behavior by simply reprogramming it, without updating the firmware on the network device.

Using a software-oriented approach is thus possible to define traffic flows and decide how to manage them into the network. This allows to achieve a higher degree of flexibility. SDN is based on two main concepts strongly related: *abstraction* and *modularity*.

Abstraction allows to define components from the interface, also characterizing components' behavior: developers are free to make their own choices about architecture. Different modules are defined in an interchangeable way and are based on the same interface.

Modularity in SDNs allows to give more flexibility since adding new functionalities can be performed by simply writing code to solve a new problem, avoiding thus the introduction and configuration of new devices.

The most innovative aspect in SDN is related to decoupling the Control Plane and the Data plane. More in depth, this decoupling implies that the Data Plane deals only with forwarding and the Control Plane simply defines rules to manage packets. The Control Plane consists of the Network Operating System and all the applications that run on it. The Control Plane takes decisions on traffic control and forwarding. The entity that manages the Control Plane is the Controller and represents the network brain, responsible for managing all network devices.

The Data Plane consists of the network hardware (routers and switches). The Data Plane only implements packet management according to Controller's commands.

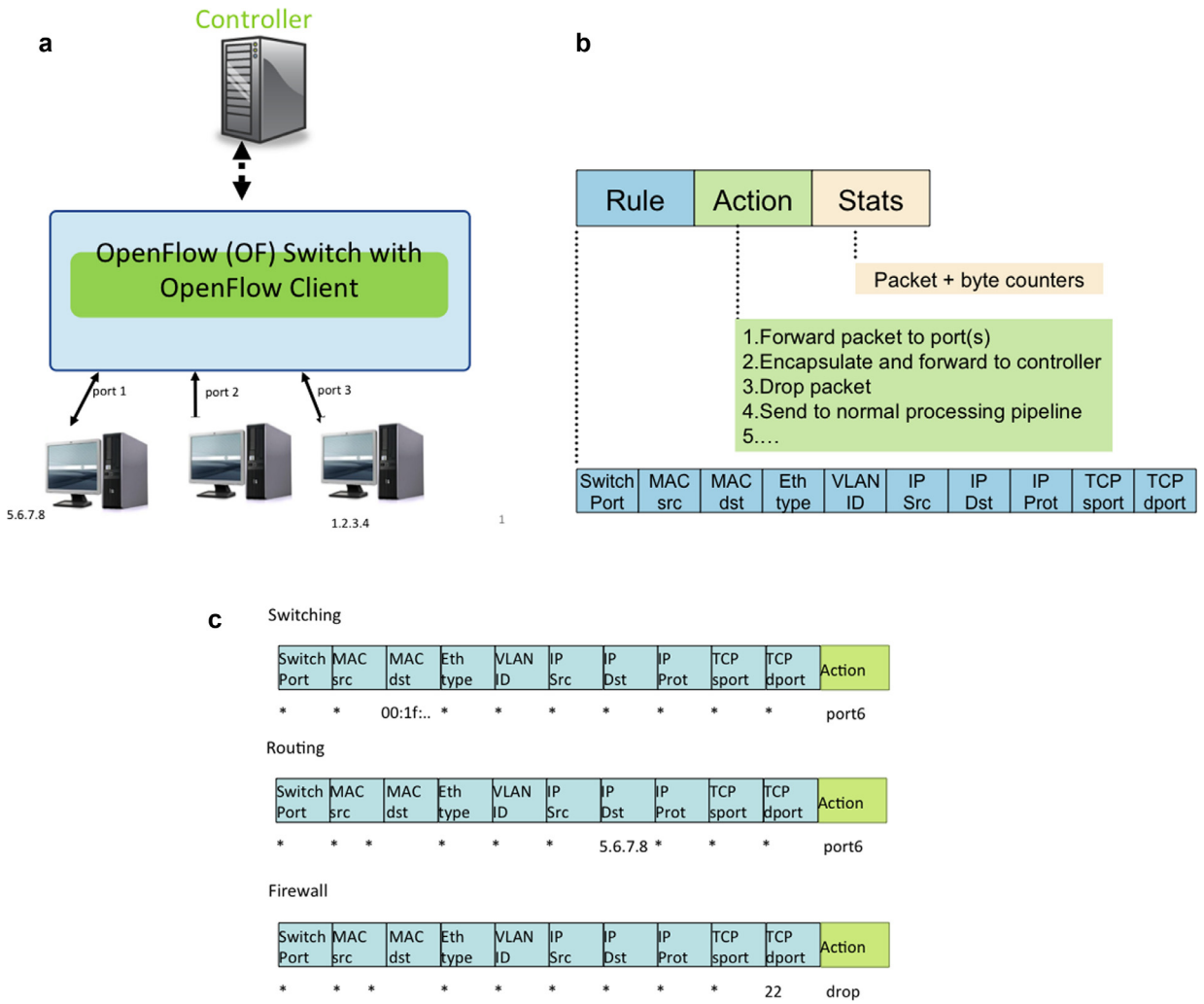


Fig. 1. Example of OpenFlow functioning (from OpenFlow/SDN tutorial, Srini Seetharaman, Deutsche Telekom, Silicon Valley Innovation Center).

SSL security is used to support collaboration between the Control and the Data Plane. The OpenFlow protocol is an example of such a SSL-enabled communication protocol [13].

OpenFlow provides a set of API or hooks [14,15] that the Controller uses to install a flow entry in the flow table of a switch. An OpenFlow switch contains a flow table which is similar to a forwarding table entry, used to manage an incoming packet.

When a switch receives a new packet, it checks if this packet matches any entry in the flow table. If the entry is found, the related rule is applied to the packet; otherwise, the Controller is asked how to treat the packet. The Controller analyzes the packet header fields, establishes the best route between source and destination, and installs a new entry on the switch flow table. A flow entry is composed by 3 fields:

- ***Rule***: it defines the flow. The Rule is checked against the packet header;
- ***Action***: it specifies how packets belonging to the flow should be treated, e.g., where should be forwarded, whether should be dropped, and whether and in case how should be modified. More actions for each flow are allowed.
- ***Stats***: it tracks the number of packets and bytes per flow and considers also the time expired from when the last packet associated to the flow was received.

An example of SDN functioning is the following.

In Fig. 1(a) a typical SDN architecture implemented through OpenFlow is illustrated. In particular an OpenFlow Switch has different ports on which hardware devices are attached. At the software level, an OpenFlow client is controlled by a remote Controller. In Fig. 1(b) an example of flow entry including Rules, Actions and Stats is shown. In Fig. 1(c) an example of how to implement different rules is provided. In particular switching rules, routing rules and firewall rules are shown.

In the next section we will recall the relevant literature on the topic of DDoS in SDN. In particular we will discuss the few solutions proposed so far which allow to exploit the typical features of SDN to cope with TCP SYNFLOOD attacks.

2.4. DDoS in SDNs

In the context of SDN research, a few solutions to DDoS attacks have been proposed. As an example, in [16] a set of switches coordinated by a NOX Controller are required to collect information from traffic flows. The NOX Controller is an open source development platform for SDN control applications. The metrics of interest are average packets per flows, average bytes per flow, average flow duration, etc. These metrics are used to identify potential DDoS attacks by recognizing if nodes belong to possible fake groups.

SDN itself has however intrinsic vulnerabilities which can be exploited by an attacker. As an example *scalability* is associated to the use of a centralized Controller in the network. This creates a bottleneck which implies that an attacker can send multiple simultaneous flow requests to quickly saturate the Control Plane. Also problems of *responsiveness* are met since there is a need to quickly access a huge amount of information to promptly identify possible DDoS attacks. Unfortunately an excessive stress in terms of polling at network switches can cause long latency, thus not satisfying the security requirements in terms of responsiveness.

In order to cope with the above issues, AVANT-GUARD has been proposed as a possible solution [17]; AVANT-GUARD focuses specifically on counteracting TCP SYNFLOOD attacks by adding an extension of the data plane denoted as *connection migration* aimed at reducing the number of interactions between data and Control Plane consequent to the DDoS attacks. Another extension to the data plane services denoted as *actuating trigger* is introduced by the Control Plane applications with the aim of performing appropriate actions to add conditional flow rules to be applied only as a consequence of an alert condition triggered by the data plane monitored statistics.

Another solution to DDoS attacks is proposed in [18] by NEC and Radware. The DoS Secured Virtual Tenant Network is a fully virtualized network solution implemented through an Anti-DoS software module that uses the OpenFlow features to monitor and collect packet level statistics from the switches in the network so identifying possible ongoing attacks.

However, all the above discussed approaches require monitoring of multiple statistics at the OF network switches. This implies a significant network overhead and scalability issues which instead could be avoided as detailed in the following.

3. OPERETTA

In this section we describe OPERETTA, an OPEnflow-based Remedy to TCP SYNFLOOD Attacks. In particular, in Section 3.1 we illustrate OPERETTA in a nutshell; then, in Section 3.2 the protocol behavior is described in more details.

3.1. OPERETTA in a nutshell

OPERETTA is an OpenFlow-based approach aimed at mitigating the TCP SYNFLOOD attacks. It is implemented in the Controller and utilizes standard OpenFlow features.

The OPERETTA protocol can be applied in any OpenFlow network. In particular it can be implemented at the Controller that is responsible for dealing with the packets exchanged during the TCP handshake but can be also executed when the Controller functionalities are delocalized in the access router at the users' premises. This possibility will be granted in the medium term as it is in the strategy of ISPs to deploy some functions under their control in the user's access routers. In Fig. 2 we illustrate the proposed scenario. In this figure we see both a set of delocalized OPERETTA controlled networks that coexist with standard networks and an OPERETTA Controller in the core network.

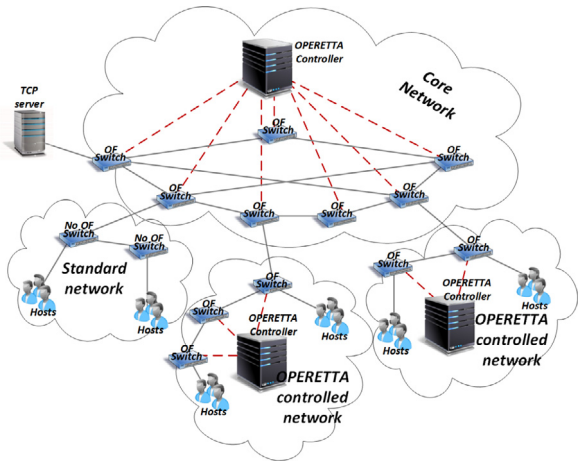


Fig. 2. Proposed scenario.

In OPERETTA the Controllers and the switches operate in such a way that a TCP server receives a request for the establishment of a TCP connection (the SYN) only after the host requesting such establishment has demonstrated to be legitimate. To achieve such objective the first attempt to open a TCP connection is always rejected. If the host proves to behave correctly, then appropriate rules are installed in the flow tables that enable successive SYN to reach the server. On the contrary, hosts showing malicious behaviors are banned and their MAC address is put in a black list for a time interval τ.[1] Note that similarly a black list could have been used for IP addresses as well; however, we preferred to rely on MAC addresses because attacks spoofing IP addresses are much more frequent than those spoofing MAC address. Actually, IP verification solutions based on BCP38 can be used in this context [10]. However, note that there is a large number of IP addresses which might be consistent with a given topology.

To this purpose, rules in the access switches are set in such a way that SYN packets are sent to the Controller, by default. The Controller acts as a proxy and responds with a SYN ACK to the requesting host. If the expected following ACK is received, then a RST messages is sent to the requesting host and a rule is installed in the access switch such that the following SYN packets are allowed to reach the TCP server. In fact, according to the TCP protocol the requesting host will trigger a new Three-Way Handshake upon receiving the RST message.

[1] We assume that the Option field in the IP packet is used to report the MAC address of the initial source node.

On the contrary, if the ACK is not received, the requesting host is considered malicious and, if this occurs too many times, the corresponding MAC address is inserted in a black list and a rule is installed in the switch such that the following SYN packets received by the banned MAC address are discarded.

3.2. OPERETTA protocol functioning

The OPERETTA protocol functioning is shown in Fig. 3.

When an incoming TCP packet is received by an OF switch, the device checks its flow table. If it is a SYN packet but does not match any flow table entry, it will be forwarded to the Controller. The Controller will check in a white list if the source MAC address is known. In the positive case, the Controller will install the forwarding rule for the corresponding flow in the OF switch flow table.

The choice to consider the MAC address instead of the IP address is aimed at reducing covert attacks. In fact, usually TCP SYNFLOOD attackers employ IP spoofing and change their IP address at any attack.

A SYN counter is used to characterize the flow coming from a specific node identified through a certain MAC address.

This counter represents the number of SYN packets not yet ACKed. The counter is compared to a threshold K which is chosen in such a way to identify a tradeoff between performance and efficiency. Indeed, choosing a large value for the threshold K would result in a delay in the attack identification and thus scarce efficiency. On the contrary, by choosing a low value of K, this would cause a too early identification of a possibly misbehaving user which might instead be a legitimate user.. In fact, the ACK can be lost due to many reasons (e.g., congestion, time out elapsed, loss of connectivity, etc.). Accordingly, if the threshold is low, the risk to expose the Controller to many attacks is reduced, thus improving performance. However this approach can lead to misinterpreted losses and delay which can be incurred and seen as a clue of attack so, invoking unnecessary TCP SYN-FLOOD mitigation procedure. On the contrary, a high threshold value would cause inefficiency in the DDoS remedy procedure because the countermeasures to TCP SYNFLOOD attacks could be actuated too late.

If the SYN counter for the considered flow is lower than the threshold K, i.e., a limited number of SYN packets have been received by the corresponding node, the Controller will pretend to be the designated receiver and will send SYN/ACK packets back to the source, according to the standard TCP Three-Way Handshake.

When the incoming packet is instead an ACK associated to a previous SYN packet, the source will be considered as a legitimate node, and the Controller will send a RST packet to the source while also installing in the OF switch a rule to forward the packet to the destination. So, when the valid source will try again to establish a TCP connection with the same destination, the SYN packet will be

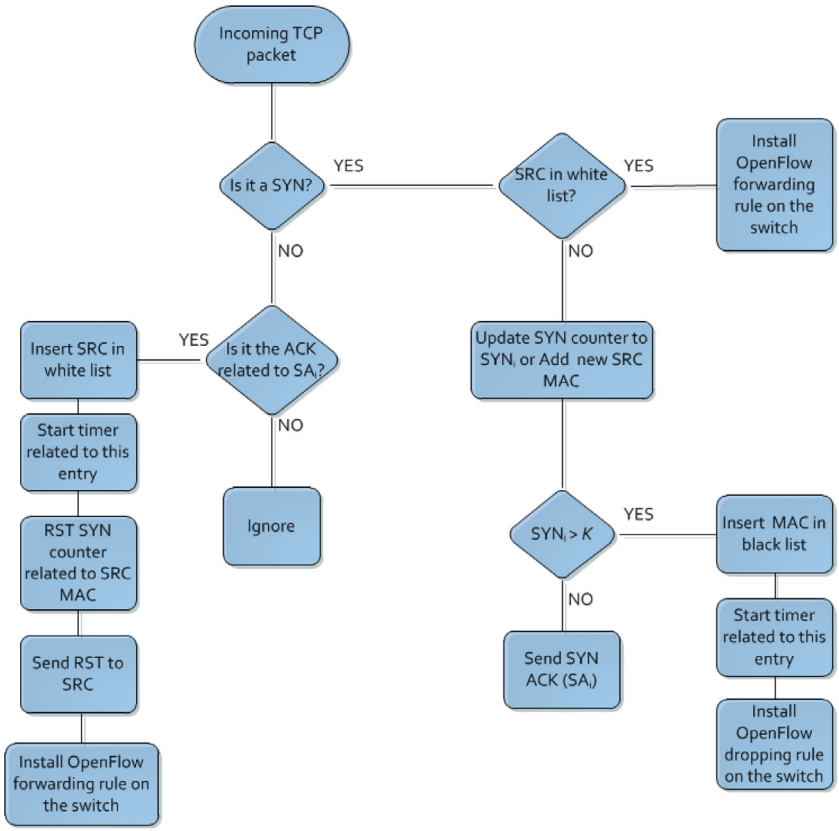


Fig. 3. OPERETTA protocol functioning.

directly forwarded by the switch, without any new Controller interrogation.

If the source MAC address is not in the white list, the source MAC is added to a temporary list and the related SYN counter updated. When the SYN counter achieves a threshold, the source is considered malicious and all packets coming from it are dropped by the switch. The source MAC will thus be inserted in a black list and the OF Controller will install the corresponding dropping rule in the switch.

When the source is marked as legitimate the Controller sends a RST packet to release the resources blocked by this fictitious open connection. Our algorithm breaks up the TCP end-to-end semantic, so the legitimate user must be able to recover the connection that has been reset by the Controller. In case of Internet traffic, since it is likely to be HTTP traffic over the TCP protocol, the RST flag notifies the TCP peer that the connection should be aborted. Observe that differently from the FIN handshake which gracefully terminates a TCP connection, a RST segment causes the connection to be abnormally closed, and this lets the HTTP protocol recover the connection to the real destination [19].

In Fig. 4 we show the exchange of messages between a legitimate user and the switch and OPERETTA Controller; similarly, in Fig. 5 we show the exchange of messages between a malicious user and the OPERETTA switch and Controller.

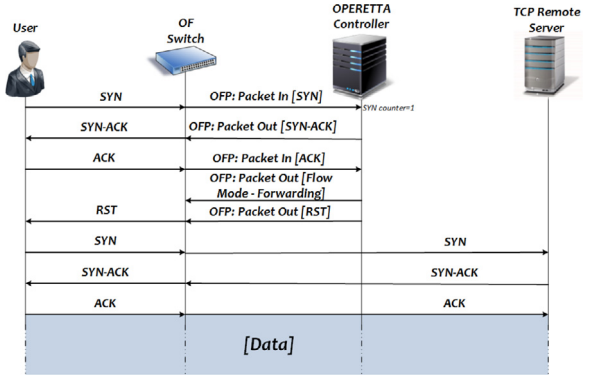


Fig. 4. Message exchange between a legitimate user and the OPERETTA controlled network.

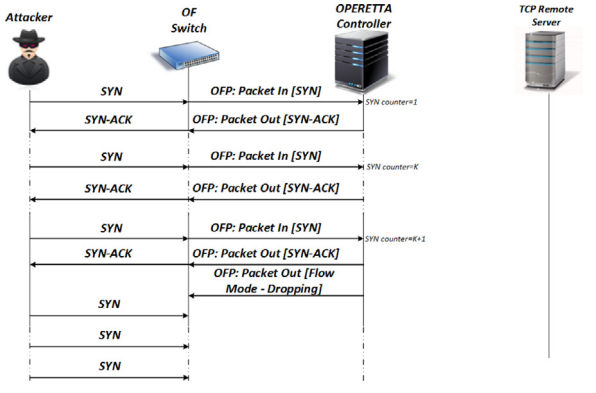


Fig. 5. Message exchange between a malicious user and the OPERETTA controlled network.

4. Performance Assessment

In this section we describe the implementation details of our experimental testbed and illustrate the scenarios being considered and the model of attack implemented.

4.1. OPERETTA implementation

In order to assess the OPERETTA protocol we have implemented a prototype adding some functionalities to the `POX Controller` (`branch Dart`); in particular we worked on Layer 3 functionalities using the `l3_learning module` [20].

The OPERETTA functionalities were implemented on the `_handle_PacketIn_` function where we also added the features to generate SYN/ACK and RST packets at the Controller.

We also created a class `Firewall` in which both the black and white lists were included, as well as the SYN counter.

To perform TCP SYNFLOOD attacks in our tests we used a Python script with SYN packets originated using `Scapy` [21]; we sent them to a selected destination identified through an IP address and a selected TCP destination port. We assumed that attackers perform IP Spoofing and act accordingly.

The connection requests by legitimate users are generated with `iperf` [22].

To evaluate the connection setup recovery time we used `SimpleHTTPServer` [23].

The SYN counter threshold has been set to 10 as a trade-off between performance and efficiency in the OPERETTA protocol.

The scenarios were implemented in `Mininet` (version 2.1.0+) [24].

The hardware used to perform simulations is a Samsung Laptop equipped with Intel(R) Core (TM) i3-2365M, CPU 1,4GHz and 4 GB RAM running Kubuntu 13.10, 64 bit.

We assumed to have a remote TCP server and 102 hosts, whereof 2 hosts are legitimate users and the other 100 are potential attackers; the internetworking is made up of 11 OpenFlow switches, 4 of them compose the access network and the others form the core network. Each host is connected to the network with a 100 Mbps link while switches are connected with a 1 Gbps link. Also the TCP server uses a 1 Gbps link.

We have run two emulation campaigns:

- ***Campaign 1***: The number of attackers is set to 50 while the number of fake SYN packets sent by each node has been changed between 50 and 250.
- ***Campaign 2***: A fixed number of fake SYN packets is sent by each attacker (100 SYN packets during the simulation time) but the number of attackers changes from 0 to 100.

Two topologies have been considered:

- ***Topology 1***: A single Controller is considered as shown in Fig. 6. In this case, the OPERETTA logic is deployed only in the central Controller. This solution is compliant with most current OpenFlow deployments, thus resulting in a potential transparent technology migration for the final users. In this topology the OPERETTA Controller is connected to all network switches. The attack is blocked in

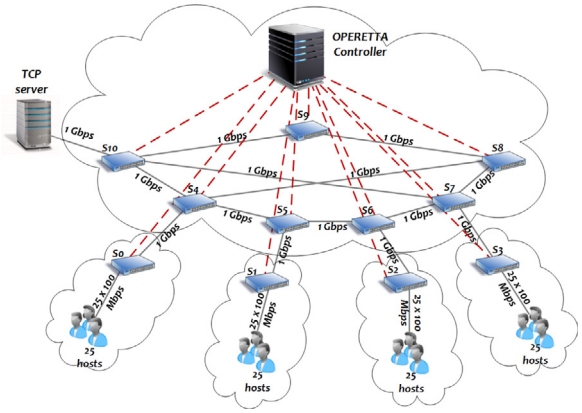


Fig. 6. Topology 1, centralized Controller.

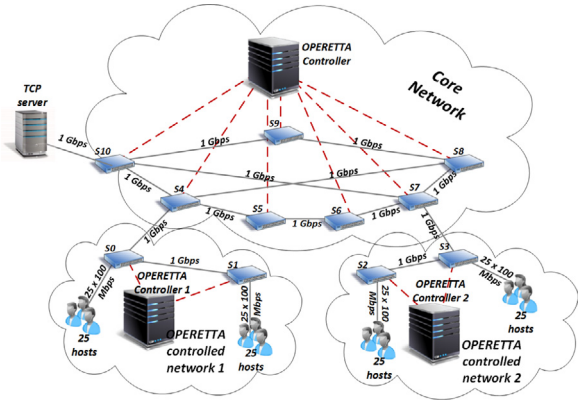


Fig. 7. Topology 2, delocalized Controllers.

the access network by a switch managed by the central Controller.

- ***Topology 2***: Multiple delocalized Controllers are employed as shown in Fig. 7. In this second case delocalized flow analysis is performed to unload the Controller. To do this, we propose that a Controller can be located in the users' premises, to check the compliancy of the source to the right TCP protocol functioning. Malicious traffic is blocked by the user switch and the OPERETTA main Controller only manages filtered traffic. Observe that this second scenario copes with scalability issues, because there is no single point of failure at the centralized Controller. This approach is innovative [25] since it calls for a radical change not only in Internet architecture but also in the users network; all local network devices should be replaced with appliances in which the OPERETTA Controller is integrated.

Simulation results compare the performance achieved by the proposed OPERETTA protocol to what can be obtained using a standard POX Controller, i.e., a Controller with a `l3_learning` module with no TCP SYNFLOOD defense mechanism.

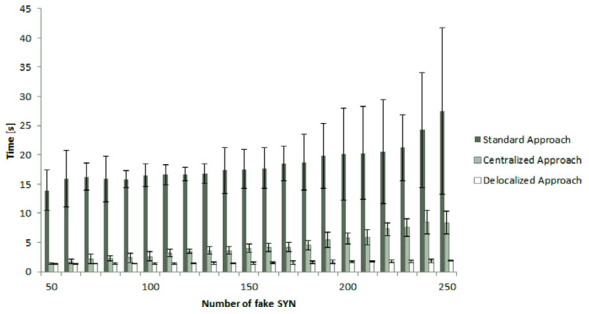


Fig. 8. Average delay to set up an HTTP connection with 50 attackers.

4.2. Numerical results

In this section we will discuss the performance achieved by OPERETTA in the two scenarios and for the two different topological settings mentioned above, when varying the type of attack carried out. To this purpose in Section 4.2.1 we will discuss the delay performance achieved; in Section 4.2.2 we will detail the related resource consumption in terms of memory and CPU processing. In Section 4.2.3 a discussion on the effectiveness of the OPERETTA approach in terms of SYN overhead at the Controller is presented. In Section 4.2.4 we will estimate the time needed by OPERETTA to block an ongoing attack. Finally in Section 4.2.5 a discussion on the limits to the application of OPERETTA is carried out.

4.2.1. Delay

In this section we evaluate the delay needed to set up a complete HTTP session. In particular we compare the OPERETTA approach to the performance that can be achieved when a standard POX Controller is employed.

In Fig. 8 we illustrate the results when an attack occurs and 50 malicious nodes simultaneously perform TCP SYN-FLOOD attacks while two legitimate users want to interact with the attacked server. Observe that in this scenario the standard POX Controller experiences an increasingly high delay if compared to the OPERETTA approach. This is because the standard POX Controller becomes victim of the malicious nodes as it is flooded by switches with rule requests messages related to the SYN packets generated by the attackers. In this case, the Controller, once overloaded by requests, saturates its resources and does not promptly answer back.

On the other hand, the OPERETTA Controllers in both configurations react promptly and the delay remains quite stable especially in the delocalized case. Observe also that a small standard deviation is obtained by OPERETTA as compared to the large fluctuations in the traditional mechanisms. This witnesses the higher stability of the OPERETTA approach as compared to standard methodologies. Similarly, in Fig. 9 we show the delay performance in case a variable number of attackers is employed, each generating a fixed amount of SYN packets. Also in this case the higher stability of OPERETTA can be observed, again exhibiting better performance in case of the delocalized topology. Note that if the number of attackers is variable, the delay incurred by OPERETTA is higher than in the case of 50 fixed attackers sending a variable number of SYN packets. This is because an increase in the number of attackers implies more interrogations to the

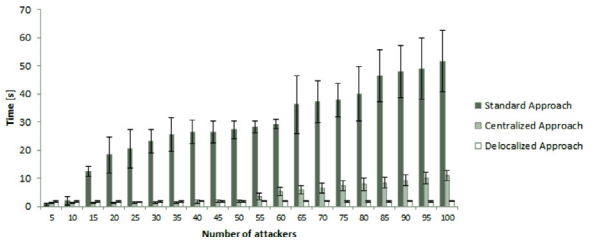


Fig. 9. Average delay to set up an HTTP connection with a variable number of attackers each sending 100 SYN packets.

Controller and, consequently, larger set up delay. On the other hand, due to the threshold mechanism used in OPERETTA, an increase in the amount of SYN packets does not compromise the TCP SYNFLOOD countermeasure performance. On the contrary, observe how the standard POX Controller results overwhelmed when the number of attackers and/or SYN packets is high.

4.2.2. Memory and processing resources

In this section we report results related to the use of system resources such as memory and CPU. In particular, in Fig. 10 we report the average CPU consumption as a percentage of the overall resources available when 50 attackers send a variable number of SYN packets. Observe that OPERETTA exhibits high stability both in case of centralized and in the case of delocalized Controllers. Note that delocalizing the Controllers allows to improve performance of about 100%. In Fig. 11 we report the CPU usage in case of a variable number of attackers. As expected, an increase in the number of attackers slightly impacts on the CPU usage in case of centralized topology while the system is more immune in case of delocalized approaches. Finally in Figs. 12 and 13 we report the percentage of memory resources used in the standard configuration as well as in the centralized and decentralized scenarios. The standard POX Controller results more memory demanding; on the contrary the OPERETTA

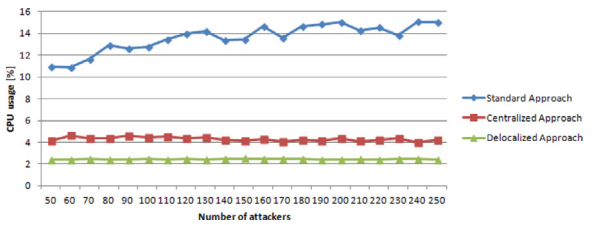


Fig. 10. Average CPU consumption with 50 attackers.

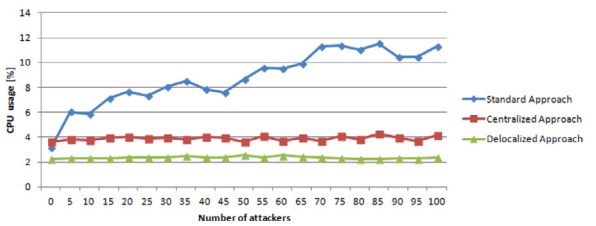


Fig. 11. Average CPU consumption with a variable number of attackers each sending 100 SYN packets.

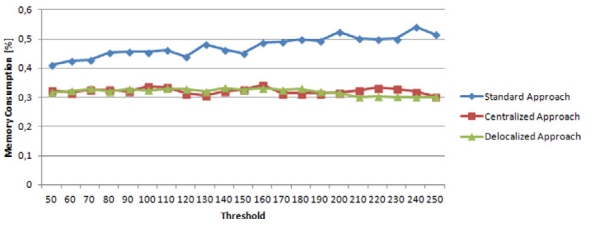


Fig. 12. Average memory consumption in case of 50 attackers.

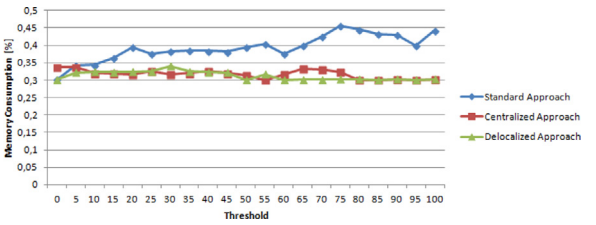


Fig. 13. Average memory consumption in case of a variable number of attackers each sending 100 SYN packets.

strategies do not suffer either for an increase in the number of attackers or SYN packets. In particular when the number of attackers is lower than 50 and each of them sends for example 100 SYN, the memory usage remains constant. Then there is a variation along with an increase in the CPU use.

4.2.3. SYN overhead

In this section we focus the attention on the percentage of SYN packets processed by the Controller. To this purpose we consider again the three above mentioned cases. In Fig. 14 in particular we note that, in an OpenFlow-based network, during a DDoS attack using IP spoofing, the Controller is forced to manage a huge amount of TCP SYN packets related to the Three-Way Handshake procedure. The majority of these TCP SYN packets are fake SYNs. In the case of standard POX Controller, we observe a high overhead at the Controller which could cause the service to be denied to the entire network. In the same figure we also report the number of fake SYN received by a centralized OPERETTA Controller. Observe that, according to the OPERETTA approach, the number of TCP SYN packets analyzed by the Controller is restricted by the threshold K. In this case having chosen for example a threshold equal to 10, we find a reduction of about 10% in the number

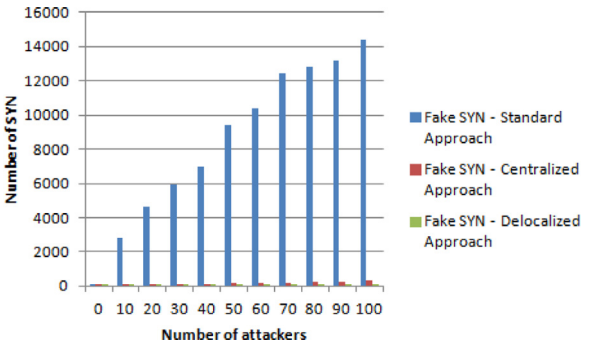


Fig. 14. Fake SYN Packets at the Controller.

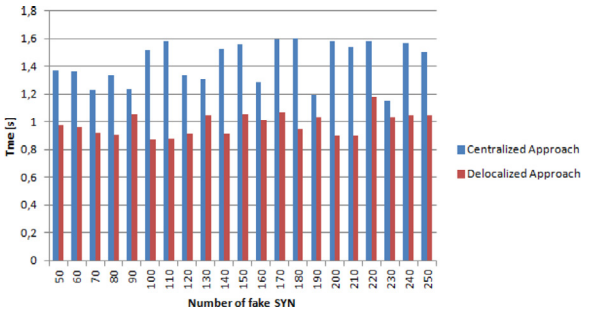


Fig. 15. Average time needed to block an attack in case of 50 attackers.

of received fake SYN packets. Indeed these packets will be dropped by the OPERETTA switches without overwhelming the Controller. In Fig. 14 we also show the number of TCP SYN packets handled by the central Controller in case the delocalized strategy can be applied. We note that this introduces a reduction in the number of fake SYN packets managed by the central Controller of a factor equal to 50% as compared to the centralized OPERETTA case. This improvement is due to the fact that the central Controller is somehow protected by the delocalized Controllers.

4.2.4. Time needed to block an attack

In this section we estimate the time needed to block an attack. In Fig. 15 we show the results when a TCP SYNFLOOD attack is performed by 50 attackers simultaneously. We notice that the centralized approach needs much more time to block the attack compared to the delocalized approach. This is because in the centralized approach just one Controller has the load of the entire network. It is relevant to note that, when keeping constant the number of attackers, the average time needed to block an attack is constant as well. This is due to the fact that this is not influenced by the number of SYN packets per attacker because in our implementation, the attack is always blocked when a number of SYN packets equal to *threshold* + 1 are received. On the other hand, in Fig. 16 we see that, in case the number of attackers is variable, the average time needed to block an attack is affected by an increase in the number of attackers. Also in this case OPERETTA centralized approach takes more time to block the attack than the delocalized approach.

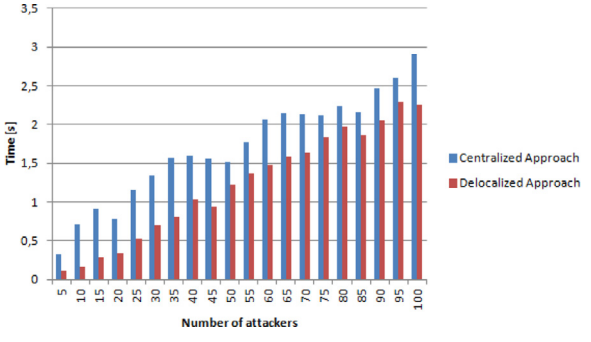


Fig. 16. Average time needed to block an attack in case of a variable number of attackers each sending 100 SYN packets.

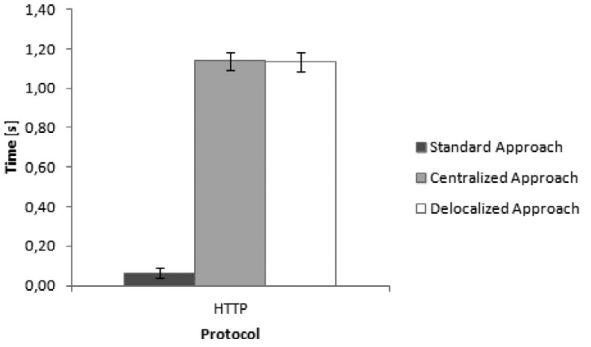


Fig. 17. Delay experienced at the legitimate hosts for setting up HTTP and TCP connections with no attackers.

4.2.5. OPERETTA: the other side of the coin

So far, we have shown that the OPERETTA protocol is responsive and reactive in case of TCP SYNFLOOD attacks. However, the OPERETTA protocol performs poorly in case there are no ongoing attacks. This is shown in Fig. 17 where we represent the average delay experienced by legitimate hosts in our scenarios, when considering no ongoing TCP SYNFLOOD attacks.

As a solution to this issue, we propose to employ a hybrid approach where the Controller activates the OPERETTA operations only in the in case it detects that a TCP SYNFLOOD attack is ongoing. To this purpose, note that a large number of solutions have been proposed to detect TCP SYNFLOOD attacks (see [26–28], for example). The performance of the proposed approach will depend on the mechanism utilized to detect the TCP SYNFLOOD attack. Selection of the best solution is outside the scope of this paper.

Observe that in any case a standard POX Controller obtains lower delay as compared to OPERETTA, both in the centralized and delocalized configurations. In fact, OPERETTA increases the delay significantly because of the use of the TCP SYNFLOOD migration procedure. This means that the OPERETTA procedures should be invoked by the Controller only when ongoing DDoS attacks are detected in the network. On the contrary, standard POX Controller procedures should be executed when all users are identified as legitimate. Accordingly, a combination of an adaptive OPERETTA Controller with a standard POX Controller would give the optimal performance as discussed above.

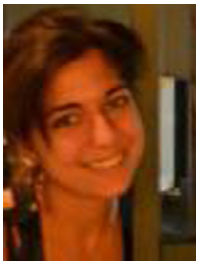
5. Conclusions

In this paper we have introduced OPERETTA, an OpenFlow Remedy to TCP SYNFLOOD attacks. OPERETTA relies on the use of an SDN approach to protect the OF Controller from TCP SYNFLOOD attacks. OPERETTA achieves four main targets, i.e., it provides a robust methodology to identify TCP SYNFLOOD attacks while also providing a flexible network mechanism for coping with these attacks which does not introduce significant CPU processing or memory overhead. Observe that OPERETTA works in heterogeneous networks, as it can be implemented not only in case of a centralized Controller, but also in the condition where delocalized Controllers are available in the access routers at the users' premises as foreseen by the evolution of the SDN paradigm. In particular in case

of use of delocalized Controllers, OPERETTA is more immune to TCP SYNFLOOD attacks as it allows to limit the scope of a possible critical attack only to a portion of the network, so avoiding to paralyze a central Controller and, thus, the entire network.

References

[1] B. Ziegler, Hacker tangles panix web site, Wall Street J. (September 12, 1996).
[2] D. Moore, G. Voelker, S. Savage, Inferring internet denial of service activity, in: Proceedings of USENIX Security Symposium, 2001.
[3] J. Mirkovic, P. Reiher, A taxonomy of DDoS attack and DDoS defense mechanisms, ACM SIGCOMM Comput. Commun. Rev. 34 (2) (April 2004).
[4] E. Damon, J. Dale, E. Laron, J. Mache, N. Land, R. Weiss, Hands-on denial of service lab exercises using SlowLoris and RUDY, in: Proceedings of INFOCSECD, 2012.
[5] M. Bogdanoski, T. Shuminoski, A. Risteski, Analysis of the SYN Flood DoS Attack, Int. J. Comput. Netw. Inf. Secur. (2013).
[6] Radware, security report, global application & network, 2013, http://www.atsweb.it/fileadmin/ATSWeb/downloads/Radware_2013_ERT_Report.pdf (accessed 2015).
[7] Cisco, defining strategies to protect against TCP SYN denial of service attacks, 2006, http://www.cisco.com/en/US/tech/tk828/technologies_tech_note09186a00800f67d5.shtml (accessed 2015).
[8] W.M. Eddy, Defenses against TCP SYN flooding attacks, Internet Protoc. J. (2006).
[9] J. Lemon, Resisting SYN flood DoS attacks with a SYN cache, in: Proceedings of the BSD Conference, 2002.
[10] P. Ferguson, D. Senie, 2000, RFC 2827 Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing, http://www.ietf.org/rfc/rfc2827.txt (accessed 2015).
[11] IETF RFC 2960 Stream Control Transmission Protocol.
[12] http://www.microsoft.com/en-us/server-cloud/solutions/software-defined-networking.aspx.
[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. (2008).
[14] Open Networking Foundation - ONF White Paper, Software-Defined Networking: the new norm for networks, 2012, https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf (accessed 2015).
[15] OpenFlow Switch Specification Version 1.1.0 Implemented (Wire Protocol 0x02), 2011, http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf (accessed 2015).
[16] A. Passito, R. Braga, E. Mota, A lightweight DDoS flooding attack detection using NOX/OpenFlow, in: Proceedings of 35th Annual IEEE Conference on Local Computer Networks, 2010.
[17] S. Shin, V. Yegneswaran, P. Porras, G. Gu, AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks, in: Proceedings of ACM CCS, 2013.
[18] Radware & NEC Corporation of America Whitepaper, Denial-of-Service (DoS) Secured Virtual Tenant Networks (VTN) Value-added DoS protection as a service for Software Defined Network (SDN), 2012, http://www.necam.com/docs/?id=0078a286-d99d-4d2c-ab54-d18814b59dd7 (accessed 2015).
[19] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, 1999, RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, https://www.ietf.org/rfc/rfc2616.txt (accessed 2015).
[20] POX Wiki, https://openflow.stanford.edu/display/ONL/POX+Wiki (accessed 2015).
[21] Scapy Documentation, http://www.secdev.org/projects/scapy/doc/index.html (accessed 2015).
[22] Iperf Documentation, http://iperf.fr/ (accessed 2015).
[23] Simple HTTP Server Documentation, http://docs.python.org/2/library/simplehttpserver.html (accessed 2015).
[24] B. Lantz, B. Heller, N. McKeon, A network in a laptop: rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, 2010.
[25] A. Tootoonchian, Y. Ganjali, HyperFlow: a distributed Control Plane for OpenFlow, in: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, 2010.
[26] H. Wang, D. Zhang, K.G. Shin, Detecting SYN flooding attacks, in: Proceedings of IEEE Infocom, 2002.
[27] A. Hussain, J. Heidemann, C. Papadopoulos, A framework for classifying denial of service attacks, in: Proceedings of ACM SIGCOMM, 2003.
[28] T. Peng, C. Leckie, K. Ramamohanara, Survey of network-based defense mechanisms countering the DoS and DDoS problems, CM Commun. Surv. 39 (1) (2007).



Silvia Fichera received her laurea degree in Telecommunications Engineering from University of Catania, Catania, Italy, in 2014. She was also trainee at Ecole Polytechnique Fédérale de Lausanne (EPFL). Now she is with Scuola Superiore di Studi S. Anna, Pisa, Italy.

Laura Galluccio received her laurea degree in Electrical Engineering from University of Catania, Catania, Italy, in 2001. In March 2005 she got her Ph.D. in Electrical, Computer and Telecommunications Engineering at the same university under the guidance of Prof. Sergio Palazzo. Since 2002 she is also at the Italian National Consortium of Telecommunications (CNIT), where she worked as a Research Fellow within the VICOM (Virtual Immersive Communications) and the SATNEX Projects. Since November 2010 she is Assistant Professor at University of Catania. Her research interests include ad hoc and sensor networks, protocols and algorithms for wireless networks, and network performance analysis. From May to July 2005 she has been a Visiting Scholar at the COMET Group, Columbia University, NY under the guidance of Prof. Andrew T. Campbell. She is member of the Sigmobile, IEEE and N2Women.

Salvatore Grancagnolo received his laurea degree in Telecommunications Engineering from University of Catania, Catania, Italy, in 2014. Formerly he was with Consorzio nazionale Interuniversitario per le Telecomunicazioni. Now he is consultant at Altran Italia SPA.

Giacomo Morabito was born in Messina, Sicily (Italy) on March 16, 1972. He received the laurea degree in Electrical Engineering and the PhD in Electrical, Computer and Telecommunications Engineering from the Istituto di Informatica e Telecomunicazioni, University of Catania, Catania (Italy), in 1996 and 2000, respectively. From November 1999 to April 2001, he was with the Broadband and Wireless Networking Laboratory of the Georgia Institute of Technology as a Research Engineer. Since April 2001 he is with the Dipartimento di Ingegneria Informatica e delle Telecomunicazioni of the University of Catania where he is currently Associate Professor. His research interests focus on analysis and solutions for wireless networks.

Sergio Palazzo was born in Catania, Italy, on December 12, 1954. He received his degree in electrical engineering from the University of Catania in 1977. Until 1981, he was at ITALTEL, Milano, where he was involved in the design of operating systems for electronic exchanges. He then joined CREI, which is the center of the Politecnico di Milano for research on computer networks. Since 1987 he has been at the University of Catania, where is now a Full Professor of Telecommunications Networks. In 1994, he spent the summer at the International Computer Science Institute (ICSI), Berkeley, as a Senior Visitor. He is a recipient of the 2003 Visiting Erskine Fellowship by the University of Canterbury, Christchurch, New Zealand. Since 1992, he has been serving on the Technical Program Committee of INFOCOM, the IEEE Conference on Computer Communications. He has been the General Chair of some ACM conferences, including MobiHoc 2006 and MobiOpp 2010, and currently is a member of the MobiHoc Steering Committee. He has also been the TPC Co-Chair of the IFIP Networking 2011, IWCMC 2013, and European Wireless 2014 conferences. Moreover, in the recent past, he has been the Program Co-Chair of the 2005 International Tyrrhenian Workshop on Digital Communications, focused on "Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements", the General Vice Chair of the ACM MobiCom 2001 Conference, and the General Chair of the 2001 International Tyrrhenian Workshop on Digital Communications, focused on "Evolutionary Trends of the Internet". He currently serves the Editorial Board of the journal Ad Hoc Networks. In the recent past, he also was an Editor of IEEE Wireless Communications Magazine (formerly IEEE Personal Communications Magazine), IEEE/ACM Transactions on Networking, IEEE Transactions on Mobile Computing, Computer Networks, and Wireless Communications and Mobile Computing. He was a Guest Editor of Special Issues in the IEEE Journal of Selected Areas in Communications ("Intelligent Techniques in High-Speed Networks"), in the IEEE Personal Communications Magazine ("Adapting to Network and Client Variability in Wireless Networks"), in the Computer Networks journal ("Broadband Satellite Systems: a Networking Perspective"), in the EURASIP Journal on Wireless Communications and Networking ("Ad Hoc Networks: Cross-Layer Issues", and "Opportunistic and Delay Tolerant Networks"). He also was the recipient of the 2002 Best Editor Award for the Computer Networks journal. His current research interests include mobile systems, wireless and satellite IP networks, intelligent techniques in network control, multimedia traffic modeling, and protocols for the next generation of the Internet.