Providing Elasticity to Intrusion Detection Systems in Virtualized Software Defined Networks

Martin Andreoni Lopez, Otto Carlos M. B. Duarte
Universidade Federal do Rio de Janeiro - UFRJ
GTA/COPPE/UFRJ - Rio de Janeiro, Brazil
Email: {martin, otto}@gta.ufrj.br

***Abstract*—This paper presents BroFlow, an Intrusion Detection and Prevention System based on Bro traffic analyzer, and on the global network-view feature of OpenFlow Application Programming Interface. BroFlow main contributions are: i) dynamic and elastic resource provision of machines under demand; ii) real-time detection of DoS attacks through simple algorithms implemented in a policy language for network events; iii) immediate reaction to DoS attacks and malicious packets, dropping flows close from their source; iv) strategic sensor positioning for attack detection in the network infrastructure shared by multi-tenants. A system prototype was developed and evaluated in the virtual environment Future Testbed Internet with Security (FITS). An evaluation of the system under attack shows that BroFlow guarantees the forwarding of legitimate packets at the maximal link rate, up to 90% reduction of the maximal network delay caused by the attack, and 50% of bandwidth gain compared with conventional firewalls approaches, even when the attackers are legitimate tenants acting in collusion.**

I. INTRODUCTION

Most of current cyber-attacks are originated by legitimate and authenticated internal users [1], thus, conventional security methods, such as firewall and access control mechanisms are totally inefficient. Therefore, the Intrusion Detection and Prevention System (IDPS) are mandatory to complement conventional security methods, protecting the system from either internal or external attacks [2]. Denial of Service (DoS) attacks consume big amounts of resources, hampering legitimate users achieve the appropriate quality of service (QoS) for their applications. DoS may even disrupt services, generating millions of dollar losses [3]. One kind of DoS attack, the SYN flooding, exploits the three handshake procedure vulnerability of TCP, keeping multiple open connections at the same time, consuming server resources. Another kind, the Distributed Denial of Service (DDoS) attacks are composed of thousands of bots which coordinate successful attacks with few packets per bot. The detection of a flooding attack formed by the sum of multiple distributed flows occurs close to the destination, hindering the blocking of the attack close to the source.

Software Defined Networking (SDN) paradigm separates the network operation into a logically centralized control plane and a distributed data plane, which provides programmability and global network view that allows a better security management. OpenFlow (OF) [4], an Application Programming Interface (API), is the most successful SDN implementation. OpenFlow provides a basic instruction set to modify, route, and block flows on the network. Consequently, it is possible to create security applications that promptly react against attacks, taking actions on network flows. Nevertheless, security concerns arise due to OpenFlow centralization [5].

In this paper we propose BroFlow, an elastic and distributed IDPS for defense against DoS attacks in virtualized Software Defined Networks. BroFlow is based on the OpenFlow [4] API and on the network traffic analyzer Bro [2]. BroFlow implements different anomaly detection algorithms against flooding DoS attacks. The BroFlow traffic sensors communicate through secure channels with an application in the POX Network Controller and thus, it performs the countermeasures to block DoS attacks. According to the system load, BroFlow adds or reduces physical resources on the fly. BroFlow uses Bro traffic analyzer and its policy language for network events making it easy to program. Unlike most current intrusion detection systems, BroFlow allows a prompt reaction to block DoS attacks, due to the OpenFlow features. BroFlow reacts directly into routing and forwarding of flows on the network and, hence, eliminates the malicious flows close to its source.

A BroFlow system prototype is implemented and evaluated into the *Future Internet Testbed with Security* (FITS), which is experimentation platform based on virtualization techniques. The results show the elasticity of the proposal to provide machines under a high packet rate flooding attack. The system shows a high efficiency to react under flooding attacks, reducing network delay up to 90%, guaranteeing proper packet forwarding with the maximal link rate up to 50% compare with conventional firewalls approaches.

The remainder of this paper is organized as follow. In Section II we describe related work. We detail the BroFlow architecture in Section III. Experimental results are shown in Section IV. Finally, Section V concludes the paper.

II. RELATED WORK

Software Defined Networking (SDN) provides a global view of the network to an intelligent and logically centralized controller, which simplifies network management. The OpenFlow API provides SDN capabilities to the network which allows the dynamic control of flows. This feature grants OpenFlow a suitable use for network security applications, such as SDN firewall [6] or anomaly detection [7], [8], being effective for detection and reaction of security threats.

Porras *et al.* [9] propose FortNOX, a special OpenFlow controller to ensure coherency of flow rule settings. As several applications run on the OpenFlow network controller, an application may configure flow rules which conflict with rules set by another application. As a solution, it is proposed a security extension in the NOX-OpenFlow controllers. In this extension, each application has its own policies with different priority. When applied in the controller flows, policies generated by security applications are prioritized. This solution, however, rejects non security applications rules in the Network Controller (NC). Hu *et al.* [6] design a firewall SDN application which presents a solution to firewall policy violation conflicts in OpenFlow based networks. Fresco is a similar approach and deals with OpenFlow rules setting [10]. Our work differs from Fresco as it provides security for virtual environment with several Virtual Network for SDNs.

Medhi *et al.* [8] implement anomaly detection algorithms into NOX-OpenFlow controller. The proposal inspects only the first packet of the connections, thus it is effective against port scanning attacks, which the packet header is only analyzed. Nevertheless, this approach is inefficient in more sophisticated attacks, such as worm attacks or virus propagation. Moreover, as the NC generates flow table entries for each new flow, when a DDoS attack creates several different flows, it overloads the NC increasing packet delivery time. Giotis *et al.* [7] propose an OpenFlow-based anomaly detection architecture. sFlow tools gather flows information,and communicate with an anomaly detector to identify potential threats. These proposals do not take into account virtualized multi-tenants environments.

XenFlow [11] prevents DoS attacks from a malicious tenant of a Virtual Network who share the same physical infrastructure. XenFlow proposes resources and traffic isolation in an OpenFlow based network. Eventhough, the isolation feature of XenFlow is fundamental for preventing DoS attacks that consumes shared resources, it is not effective for preventing flooding DoS attacks in a Virtual Network. DoS attacks into SDN are studied by Lim *et al.* [12] presenting a botnet DDoS detection scheme in OpenFlow networks. This proposal overloads the NC during the attack detection process.

SnortFlow [13] consists in an IDPS based on the Snort tool, which is an open source IDS based on signature detection, and OpenFlow API. The Snort Agent is localized into the management domain on the XEN hypervisor. This proposal lacks communication between the single Snort Agent and the Network Controller. Moreover, this work only evaluates the performance of the agent localization in the XEN hypervisor. The agent localization only in the management domain causes a coarse-grained rule implementation for each Virtual Network. Furthermore, the Snort tool only utilizes the signature detection method lacking the anomaly detection, which generates high false positive rate under small attacks variations.

III. The Proposed System

BroFlow employs a software programmable switch, Open vSwitch (OVS)[1], used as an OpenFlow switch. OVS presents a forwarding table which could be updated by an OpenFlow controller, it also offers several features, such as packets dropping, packet-header fields modifying, etc. The POX Network Controller (NC) configures and controls OpenFlow switches. We choose POX controller among others due to its programming simplicity and a fair trade-off between prototyping time and performance. Besides, our POX-based prototype, BroFlow application is easily portable to other controllers, even distributed ones [14]. Our system architecture considers a hybrid network virtualized environment, composed by XEN Virtual Machines running over an OpenFlow switching matrix. In these virtualization environments, the VMs are connected through OpenFlow switches, implemented by the programmable Open vSwitch (OVS).

[1]http://openvswitch.org/

BroFlow sensors are spread in the networks, then sensor localization results into an optimization problem. For attack detection, it is possible to establish a reduced number of sensors instead of placing sensors in every switch. This advantage is granted by combining Bro tool with the network global view provided by OpenFlow. We choose the Bro open source network traffic tool analyzer due it high policies description language, which defines events for the network activities, represented by a packet abstraction in a higher information level. With `Bro` language, the user can define its own policies. In addition, Bro inspects network traffic in real time, creating reports and alarms when a security policy is threaten.

A. *BroFlow Sensors*

Our system owns two types of sensors, the BroFlow Virtual Network (VN) Sensors and the BroFlow Infrastructure Sensor, illustrated in Figure 1. Every sensor executes a Bro tool daemon, with minimal resources consumption. The VN sensor, which monitors either virtual switches or specific hosts, must be geographically distributed between the virtual switches. In each VN Sensor, are established specific and independent policies for each Virtual Network. This facility provided by BroFlow is important in a cloud virtualized environment, because the policy persists even when there is a virtual switch migration. Once the BroFlow sensor carries all policies, and it migrate together within the switch. In addition, a daemon monitors system resources consumed both by physical machines and the virtual machines. In this architecture, the physical machines could allocate several BroFlow Virtual Network Sensors which send notifications trough a secure communication channel with the network controller. All BroFlow sensors have attack detection and alarm-dispatch applications defined by two modules: Policy and Countermeasure Modules.

The BroFlow Infrastructure Sensor runs parallel to the Network Controller (NC) in order to protect the Physical Network (PN). An example of an infrastructure threat is the ARP flooding attack, known as ARP poisoning. This threat attacks switches with ARP packets to overload the routing tables with fake MAC addresses, causing a memory DoS of infrastructure switches. The Infrastructure Sensor detects and prevents these types of attacks, protecting the PN and the hosted VN. Furthermore, BroFlow Application, executed into the Network Controller, manages the alarms and countermeasures. The captured packets are sent to the event engine which, verifies, orders and converts them into events. Then, these events are sent to the policy interpreter. The event engine and the policy interpreter belong to the original Bro traffic analyzer tool. The BroFlow Policy Module decides which events generated by Bro represent an attack, and in that case, which action the BroFlow Application must take.

1) BroFlow Policy: Security Policies are composed by two modules: Network Event Inspection (NEI) and Attack Detection (AD). The NEI Module analyses in real time relevant information, provided by Bro tool, about the established flows during packet reception. All DoS attack detection policies are written in `Bro` language, and consist of: TCP-SYN, ICMP,

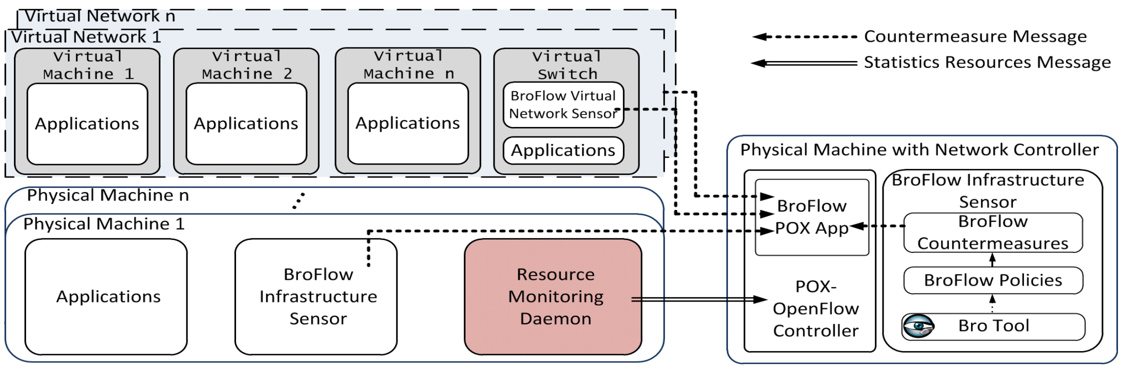


Figure 1. BroFlow system Architecture. Sensors analyses network traffic to protect the physical infrastructure and virtual networks from attacks. All sensors communicate with the BroFlow Application thought secure channels. Inspection modules provide elastic resources on demand.

and UDP. Thus, every time a packet related with these events is detected by NEI module, the AD module is invoked. The AD module implements algorithms, abstracted into policies of `Bro` language. In this paper we exemplify the use of Bro with two detection algorithms for packet flooding attack: ramp and adaptive thresholds [3]. Ramp algorithm sums packets during a specific period and raises an alarm when a threshold is reached. Adaptive threshold aims to decrease the false positive due to "flash crowds", that quickly exceeded the mean value. The algorithm detects variations in traffic statistics, based on traffic measures in consecutive T time intervals. Every time that a threshold is exceeded, a counter k is incremented. A counter bigger than one indicates a threshold exceeding for consecutive time intervals, featuring an anomaly considered as an attack.

B. BroFlow Countermeasures

The countermeasure module performs the communication with the BroFlow Application in the POX OpenFlow Network Controller (NC). This module, translates the information generated by the BroFlow Sensors and forwards the alarms messages to the BroFlow Application. When an attack is detected by the policy module it sends an alarm message to the OpenFlow Controller. We establish a Secure Socket Layer (SSL) channel in dedicated network interfaces, ensuring authenticated and encrypted communication. The messages are JSON formatted and include flows information, IP addresses and ports, source and destination, the destination MAC address and the countermeasure to be taken. These fields are all information that Bro traffic analyzer tool obtains from a monitored packet. As these fields do not composes an OpenFlow (OF) complete flow, OF fills the other fields with wild-cards values. Although this definition installs general flows in switches, the use of wild-card fields do not generate ambiguity, as long as a TCP connection is normally defined by four specific fields: source and destination addresses, source and destination ports. Thus, as the four fields that identify the TCP connection are well defined, there is not ambiguity in the suspicious flows.

C. BroFlow Application

BroFlow application runs on the top of POX-OpenFlow Network Controller. This application receives alarms derived from several BroFlow sensors and executes the required countermeasures to answer those alarms. Thereby, when an alarm message is received from sensors countermeasure module, the BroFlow Application verifies in its table which flow match the alarm message content. After that, the application indicates the network controller the countermeasure to be taken in all network switches. In our prototype, the countermeasures are corresponding OpenFlow action: *drop* for packet dropping and *output* to forwarding the packet to a specific switch port. Therefore, with these two OpenFlow actions over the network switches, we defined countermeasures to *block* a specific flow or set of flows, and to *deviate* a flow to another host, in order to avoid the DoS attacks. All the countermeasures are applied under a quarantine regime. Every time a countermeasure is applied in the switches, a timer is activated. When a timer bursts, the countermeasure is cleaned and all the analyses are established again. Hence, it is possible to detect if the attack finishes, ceasing to use system resources.

1) Flow management Module: Bro tool machine can remains overloaded with a high packet rate of a flooding DoS attack. As a consequence, our system can deviate the malicious flows to be analyzed in several parallel machines. We create a module into the BroFlow Application in the Network Controller, responsible for flow distribution between BroFlow Sensors. For packet mirroring between the BroFlow Sensors, we use *Generic Routing Encapsulation* (GRE) tunnel. Although GRE increases the total amount of data in the network, this approach alleviates inspection loads in each individual physical machine. The packet inspection is done after the decapsulation, assuring packet integrity. This mirroring method allows to place BroFlow Sensors into different Virtual Networks hosted on different Physical Machines. Therefore, the flows distribution takes into account the system resources availability in each virtual machine; and the packet source. A flow of a new source is allocated in the lowest processing machine, and flows from the same source are allocated together in the same machine preventing attacks of going unnoticed.

2) Resources Management Module: Located in the privileged domain, called Domain 0, of every Physical Machines (PM) in the network, this module monitors system resources such as bandwidth, processing and memory of each PMs. Resources monitoring is performed by XEN data gathering, through the *libvirt* library running as a daemon. Statistics of all PMs are aggregated in the Network Controller (NC). Thereby, the NC has information about the resources availability of each analyzed machine. In case of an overload, this module analyzes the available resources in the physical machines and decides where to instantiate a new BroFlow Sensor. Likewise, all PM containing BroFlow Sensors are analyzed together, in order to detect when a flow redistribution is possible, allowing to deactivate a machine, ensuring the elasticity of the proposal.

IV. RESULTS

We developed a prototype in *Future Internet Testbed with Security*[2] (FITS), an interuniversity testbed for Future Internet proposals. FITS consist of spreads nodes between Brazilian and European institutions to develop experimentation in new generation networks. FITS is based in the XEN and OpenFlow (OF) mechanisms to provide a pluralist architecture, allowing a coexistence of parallel multiples networks running different applications. In FITS [15] the control plane executes in the XEN VMs and the packet forwarding is performed by OF.

A. Countermeasure Evaluation

The first experiment analyzes TCP-SYN packet flooding DoS attack in VN hosted by physical switches. The considered scenario is constituted by a physical switch hosting four virtual switches belonging to four different VN. Every Virtual Machine, corresponds to a virtual switch, runs a BroFlow sensor, which analyzes the traffic of its network. In addition, an Infrastructure Sensor is installed in the physical router.

Figure 2 shows the experiment with three attackers sending SYN packets at different rates, 45, 50, 55 packets per seconds. It was defined, as a test criterion the threshold in 100 SYN packets per second which represents the maximal SYN packet rate allowed in each network, then the threshold in each VN is never passed. Nonetheless, as the tenants act in collusion, the maximal established threshold is exceeded more than 50%. It is because the connection aggregated rate per second of the VNs is totally forwarded the physical switch that hosts these VNs. Thus, the threshold established in the VN is not extrapolated individually, but the aggregated threshold is considered an attack for the Infrastructure Sensor. At the detection moment, approximately at 40 seconds, an alarm message is generated by BroFlow sensor located at the physical switch. This value allows the implementation of the adaptive threshold algorithm to avoid flash crowds. This algorithm increment a counter k, when the mean rate value of the previous T time interval of 10 seconds is exceeded. Then, if the average packet rate is exceeded only one time, the countermeasure is not launched, assuming a false positive, but if the average rate is exceeded four times consecutively, the alarm is sent to the BroFlow application. Hence, from the Infrastructure Sensor viewpoint, there is a DoS Attack to the physical infrastructure. The adaptive threshold values adopted are: estimated average per interval $\mu_i = 100$; exponentially-weighted moving average (EWMA) factor $\beta = 0.98$; amplitude factor $\alpha = 0.5$; time interval $T = 10$ seconds; successive threshold violation $k = 4$, being the same values adopted by Siris and Papagalou [3].

The next experiment analyzes the selective block effect, when a flow is considered malicious over others flows in the network. Figure 3 indicates the moment when an attack is blocked, and the legitimate flow is not affected. When an intrusion is detected, the Intrusion Detection Module sends an alert to the network controller which updates its flow table, blocking the malicious flows in all OpenFlow Switches. Detection occurs approximately at 20 seconds, and the countermeasure is applied under a quarantine regime. After 15 seconds, the quarantine finished, avoiding system idle resources. If the attack is still occurring the countermeasure is reapplied.

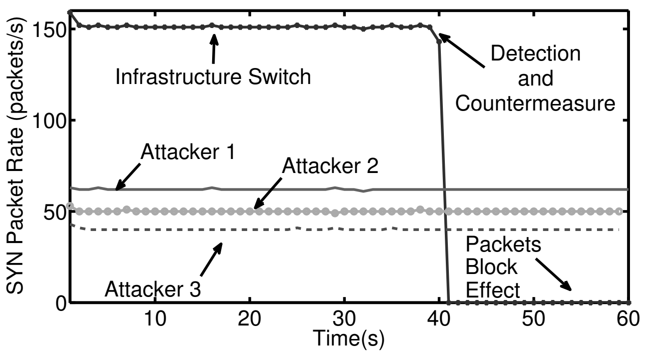


Figure 2. Infrastructure threat detection. Three attackers acting in collusion, detected by Infrastructure Sensor around 40 seconds.

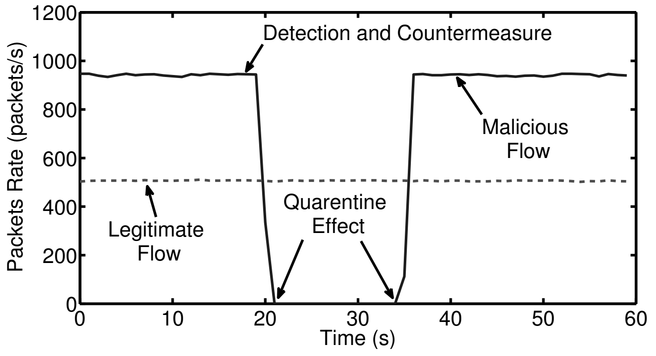


Figure 3. Selective flow block and countermeasure quarantine effect. Once an attack is detected, around 20 seconds, the flow is blocked and 15 seconds later the flow is permitted, avoiding idle resources.

The SDN network global view, allows a DoS attack to be blocked close to its source. This behavior is evaluated with three virtual switches working in one physical machine. The attacker performs an UDP flood at different rates, from 0 to 500 Mb/s, to a victim at two-hop distance. At the same time, a legitimate VM, which shares the link with the attacker, performs a TCP bandwidth measurement. We implement one BroFlow Sensor in the last virtual switch close to the victim. We compare our solution with iptables and with the system without any defense. Figure 4 shows that reception of TCP packets without defense at 500 Mb/s is lost, because UDP attack fills all link capacity. An improvement is reached with iptables, whereas it blocks the malicious flow in the last hop. Nevertheless, with BroFlow leveraging the SDN global view, once the sensor detects the attack it blocks the malicious flow in all virtual switches, even in the closest switch to the source. As Figure 4 shows, the fall between 0 and 50 Mb/s, in case of BroFlow, it is due to the time taken to detect an attack. All the experiments performed are presented with average value and a confidence interval of 95%.

Figure 5 shows the BroFlow performance evaluation under a flooding attack. Figure 5(b) shows either the overload

[2]http://www.gta.ufrj.br/fits.

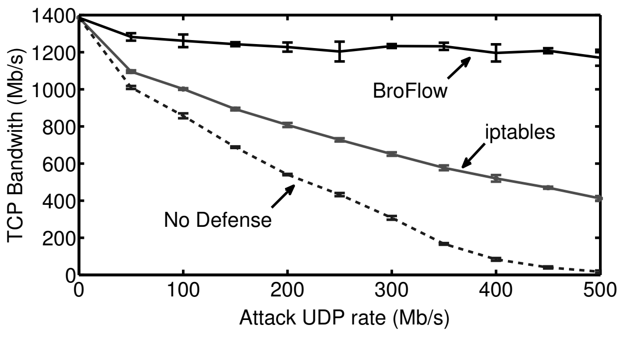
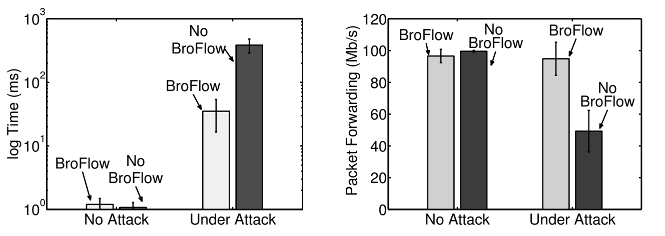


Figure 4. BroFlow compared with the iptables and with the system with no defense under a UDP flood attack at different rates. A gain of 50% is obtained by the BroFlow, where a global countermeasure is applied in all switches blocking malicious flow close to its source.



(a) Network average delay. (b) Packet forwarding comparison.

Figure 5. BroFlow performance evaluation under and without attack. a) Network average delay is reduced up to 90% under attack scenario, time in logarithm scale. b) Packet forwarding rate of the system is minimum affected under attack with BroFlow.

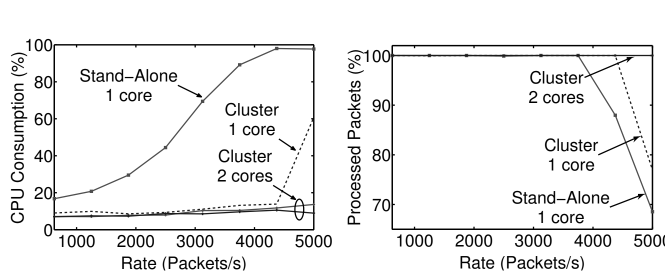
introduced by BroFlow as well as its efficiency blocking a DoS attack. Figure 5(a) compares the average packet forwarding delays with and without an attack. The delay added by BroFlow is insignificant when there is no attack. On the other hand, BroFlow decreases the average delay due to the packet dropping under attack. Figure 5(b) shows the packet forwarding rate. BroFlow practically does not overload the system without attack, reaching the maximal rate of 100 Mb/s. During the DoS attack, the forwarding rate falls 50% of the maximal rate, while the maximal average keeps almost unaffected with the BroFlow system.

B. Evaluation of Resources Consumed by Bro

IDPSs analyze in real time all mirrored traffic, and thus, it overloads the IDPS machine. Hence, we evaluate consumed resources of the Bro traffic analyzer tool to determine which aspect bandwidth or processing is the most critical for the system when a DoS attack takes place. It is important to highlight that the processing required for the analysis machine depends on the security policy and the threat types. Deep Packet Inspection (DPI) technique demands a considerable amount of processing. We generated increasing packet rates, and we analyzed the amount of CPU spent by the machine and the perceptual of analyzed packets by Bro.

Bro natively executes as single-threading, using only one CPU core [2]. Within Bro evolves, it executes as multi-threading, using several CPU cores, with the use of cluster technique and the help of the `PF_RING` library instead of the `libpcap` [2]. We evaluate the use of both technologies under a DoS attack. In the conventional configuration, called stand-alone, the Virtual Machine (VM) is configured to have access only to one core, avoiding idle resources. In cluster configuration it was performed an execution with only one core and with two cores. Figure 6(a) shows cluster configuration running two cores. Bro simulates a cluster but still uses single-thread technique in each processor.

Figure 6(a) shows Bro CPU consumption in a VM. In stand-alone configuration the system saturates using all CPU resources. Thus, after 4000 packets per second, CPU consumption reaches 100%. Notwithstanding, for the same packet rate, in one core cluster configuration, the system stays behind this limit and with two cores configuration. In this configuration, CPU consumption increase was almost negligible under the maximal tested rate of 5000 packets per second. Figure 6(b) shows the difference between sent and received packets analyzed. Comparing these values with the ones in Figure 6(a), we observe that analyzed packets suffer a decrease when CPU consumption is maximum, this effect is notorious with one core stand-alone configuration, reaching to analyses at most only 70% of the packet under the maximal tested rate of 5000 packets per second. The results show an improvement when one core cluster configuration is used, under the maximal tested rate of 5000 packets per second, in which approximately 80% of packets are analyzed. Nevertheless, in two-cores cluster configuration, under the same rate, there is no packets dropped.



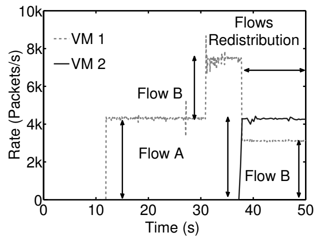
(a) Bro tool process consumption comparison. (b) Analyzed packet by Bro tool comparison.

Figure 6. CPU consumption and analyzed packet comparison. Bro traffic analyzer tool running in stand-alone and cluster with one and two cores configuration.

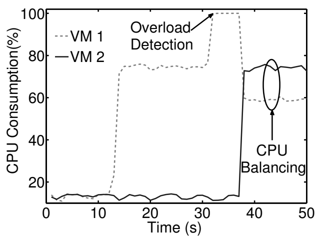
C. Elasticity Under Attack

For this experiment, we generated constant packet rates that are inspected by an only active virtual machine. Then we create a new flow in order to overload the inspection module in the BroFlow Sensor machine. When the daemon running on the PM detects the overload, it sends a message to the Network Controller (NC). In this experiment, the overload is considered the CPU consumption of the machine where Bro is executed. Whereas the NC receives the overload message, it executes the balancer algorithm to decide if a new BroFlow sensor machine must be instantiated. After the new machine activation, the flows are redistributed taken into account its origin and the resources available in the sensors machines.

Figure 7 shows when second flow starts, it overloads the BroFlow Sensor machine, approximately at 30 seconds. Once an overload is detected, there is a time interval to the new machine instantiation, until all flows are redistributed. Then, after the flow load balance, all the packets are being analyzed

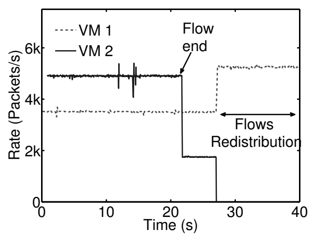


(a) Packet rate received by BroFlow Sensors in an overload scenario.

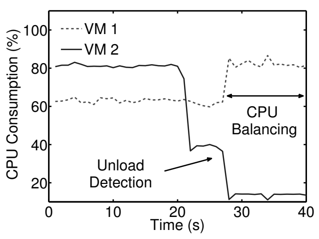


(b) CPU consumption of BroFlow Sensors in an overload scenario.

Figure 7. Analysis of the CPU consumption and Packet reception of the BroFlow Sensor machines in an overload scenario.



(a) Packet rate received by BroFlow Sensors in an unload scenario.



(b) CPU consumption of BroFlow Sensors in an unload scenario.

Figure 8. Analysis of the CPU consumption and Packet reception of the BroFlow Sensors in an unload scenario.

without overloading the BroFlow Sensor. This test performs a temporal analysis of the machines in overload case. Two flows are initiated in one machine, Figure 7(a), causing a CPU overload as shown in Figure 7(b). To avoid the overload a new machine is instantiated and all the flows are redistributed, balancing the CPU consumption. In contrast, we evaluate the system on the opposite scenario, when the system is unloaded. The consumed resources are constantly analyzed and informed to the NC in order to redistribute flows with the goal to deactivate BroFlow Sensors, in order to avoid idle resources.

Figure 8 shows the temporal analysis of the BroFlow Sensors in a unload scenario, the test begin with two Virtual Machines receiving packets in a constant rate. After some time, one of these flows is deactivated, causing a noticeable decrease in the CPU consumption of one BroFlow Sensors. When unload is detected, the NC redistribute the flows, so the machine with the lowest CPU consumption will not receive any flows, and thus can be deactivated. Two machines are inspecting packets when one flow is over, Figure 8(a). Then, the NC analyze the CPU consumption of the two machines, Figure 8(b), redistributing flows in order to deactivate a Sensor.

V. Conclusion and Future Work

In this paper we presented BroFlow, an Intrusion Detection and Prevention System (IDPS) for Denial of Service (DoS) attacks in virtualized Software Defined Networks (SDN). BroFlow joins the simplicity of policy elaboration of Bro tool with the network global view and control agility provided by OpenFlow. BroFlow contributions are evidenced with a prototype implementation running over FITS platform. Upon elasticity techniques, several BroFlow Sensors can be instantiated dynamically in case of overload, taking into account system resources. Moreover, sensors can be deactivated in case of unload. Therefore, the architecture provides resources according to demand. The prototype shows a good performance reacting to different DoS attacks, reducing up to 90% the network delay caused by the attack. Trough simple detection algorithms, the system blocks packets close the source attack, allowing the network availability in more than 50% compared with conventional firewalls approaches. As future work, we will correlate different BroFlow sensors alarms, taking into the account the rules establishment into switches.

Acknowledgment

This work was supported by CNPq, CAPES, and FAPERJ. We also thank Antonio Lobato and Ulisses Figueredo for their significant contributions to obtain the results.

References

[1] D. M. Lynch, "Securing against insider attacks," *Information Systems Security*, vol. 15, no. 5, pp. 39–47, 2006.

[2] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE Symposium on Security and Privacy*, May 2010, pp. 305–316.

[3] V. A. Siris and F. Papagalou, "Application of anomaly detection algorithms for detecting SYN flooding attacks," *Computer communications*, vol. 29, pp. 1433–1442, May 2006.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Computer Communication*, vol. 38, pp. 69–74, Apr. 2008.

[5] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable Software Defined Networks," in *Proceedings of the second workshop on Hot Topics in SDN*. ACM, Aug. 2013, pp. 55–60.

[6] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "FLOWGUARD: building robust firewalls for Software-Defined Networks," in *Proceedings of the third workshop on Hot topics in SDN*. ACM, Aug. 2014, pp. 97–102.

[7] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments ," *Computer Networks*, vol. 62, pp. 122 – 136, Apr. 2014.

[8] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Recent Advances in Intrusion Detection*. Springer, Sep. 2011, pp. 161–180.

[9] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proceedings of the first workshop on Hot topics in SDN*. ACM, aug 2012, pp. 121–126.

[10] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular composable security services for Software-Defined Networks," in *Proceedings of Network and Distributed Security Symposium*, Feb. 2013.

[11] D. M. F. Mattos and O. C. M. B. Duarte, "XenFlow: Seamless migration primitive and Quality of Service for virtual networks," IEEE Global Communications Conference - GLOBECOM, Dec. 2014.

[12] S. Lim, J. Ha, H. Kim, Y. Kim, and S. Yang, "A SDN-oriented DDoS blocking scheme for botnet-based attacks," in *Sixth International Conferece on Ubiquitous and Future Networks*, Jul. 2014, pp. 63–68.

[13] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar, "SnortFlow: A OpenFlow-based intrusion prevention system in cloud environment," in *2nd GENI Research and Educational Experiment Workshop*, Oct. 2013, pp. 89–92.

[14] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: a distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. USENIX Association, 2010, pp. 1–6.

[15] I. M. Moraes, D. M. 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. Duarte, and G. Pujolle, "FITS: A Flexible Virtual Network Testbed Architecture," *Computer Networks*, vol. 63, pp. 221–237, Apr. 2014.