Ann. Telecommun. (2016) 71:595–605
DOI 10.1007/s12243-016-0506-y

An elastic intrusion detection system for software networks

Martin Andreoni Lopez[1] · **Diogo Menezes Ferrazani Mattos**[1] · **Otto Carlos M. B. Duarte**[1]

Received: 29 June 2015 / Accepted: 26 February 2016 / Published online: 5 March 2016

Abstract Internal users are the main causes of anomalous and suspicious behaviors in a communication network. Even when traditional security middleboxes are present, internal attacks may lead the network to outages or to leakage of sensitive information. In this article, we propose BroFlow, an Intrusion Detection and Prevention System based on Bro traffic analyzer and on the global network view of the software-defined networks (SDN) which is provided by the OpenFlow. BroFlow main contributions are (i) dynamic and elastic resource provision of traffic-analyzing 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, dropping malicious flows close of their sources, and (iv) near-optimal placement of sensors through a proposed heuristic for strategically positioning sensors in the network infrastructure, which is shared by multi-tenants, with a minimum number of sensors. We developed a prototype of the proposed system, and we evaluated it in a virtual environment of the Future Internet Testbed with Security (FITS). An evaluation of the system under attack shows that BroFlow guarantees the forwarding of legitimate packets at the maximal link rate, reducing up to 90 % of the maximal network delay caused by the attack. BroFlow reaches 50 % of bandwidth gain when compared with conventional firewalls approaches, even when the attackers are legitimate tenants acting in collusion. In addition, the system reduces the sensors number, while keeping full coverage of network flows.

Keywords DoS attacks · Intrusion detection and prevention system · Software-defined networking · Network security

This work was supported by CNPq, CAPES, and FAPERJ.

✉ Martin Andreoni Lopez
martin@gta.ufrj.br

Diogo Menezes Ferrazani Mattos
diogo@gta.ufrj.br

Otto Carlos M. B. Duarte
otto@gta.ufrj.br

[1] Grupo de Teleinformática e Automação (GTA), Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil

1 Introduction

Intrusion Detection and Prevention System (IDPS) are mandatory to complement conventional security methods, protecting the system from either internal, or external attacks [17]. However, the resources inflexibility degrades IDPS performance while preventing infrastructure attacks [12], such as flooding denial of service (DoS) attacks. (DoS) attacks consume big amounts of resources, hampering the achievement of the appropriate quality of service (QoS) for legitimate users' applications [14].

Software-defined networking (SDN) provides a global view of the network to an intelligent and logically centralized controller, which simplifies network management. The information exchange between the network controller and the applications is done through the Northbound Application Program Interface (API). The switches received information through the Southbound API, with the most popular implementation OpenFlow API [9]. 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. Additionally, SDN manages multi-tenant network where multiple virtual networks run in parallel [8]. In these environment, it is mandatory to obtain the highest granularity for each virtual network (VN), in order to detect specific attacks in each VN.

In this article, we propose BroFlow, an elastic and distributed IDPS for SDN in virtualized environment. BroFlow is based on the OpenFlow [9] API and on the network traffic analyzer Bro [13]. As a study case, BroFlow implements different anomaly detection algorithms against flooding DoS attacks. BroFlow traffic sensors are deployed in virtual networks and in the infrastructure. This deployment allows a fine granularity in detection, when sensors communicate through secure channels with an application running on top of the POX[1] Network Controller and, thus, it performs the countermeasures to block DoS attacks. According to the system load, BroFlow replicates or tears down the resources, supplying in on demand. BroFlow uses Bro traffic analyzer and its policy language for network events. Unlike most current IDPS, BroFlow allows a prompt reaction to block attacks, by means of 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. BroFlow models and proposes a heuristic for optimization in IDPS sensor placement, reducing the number of sensor and maximizing the network coverage.

A BroFlow system prototype is implemented and evaluated into the *Future Internet Testbed with Security* (FITS), which is an 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 % compared with conventional firewalls approaches.

The remainder of this article is organized as follow. In Section 2, we describe related work. We detail the BroFlow architecture in Section 3. The sensor placement optimization is modeled in Section 4. Experimental results are shown in Section 5. Finally, Section 6 concludes the paper.

2 Related work

The dynamic control of flows feature grants OpenFlow a suitable use for network security applications, such as SDN firewall [7, 20] or anomaly detection [6, 10], being effective for detection and reaction to security thread.

Shanmugam et al. propose a distributed IDPS for cloud computing [15]. The paper uses SDN to distribute detection sensors and elastically provide resources according with the processing demand of each attack. This approach lacks of an optimization sensor placement and an evaluation of the detection methods.

Medhi et al. implement anomaly detection algorithms into NOX-OpenFlow controller [10]. The proposal inspects only the first packet of the connections, thus, it is effective against port scanning attacks, in which the packet header is only analyzed. Nevertheless, this approach is inefficient in more sophisticated attacks, such as worm attacks or virus propagation. Hu et al. design a firewall SDN application which presents a solution to firewall policy violation conflicts in OpenFlow-based networks [7]. Giotis et al. propose an OpenFlow-based anomaly detection architecture [6] using sFlow data collection. sFlow gathers flows information and communicates with an anomaly detector to identify potential threats. Yoon et al. adopts a combination of both approaches [20], that is, it combines a firewall with detection capabilities into a Floodlight network controller. However, these proposals do not consider virtualized multi-tenants environments.

SnortFlow [19] consists in an IDPS based on the Snort tool, which is an open source IDS based on signature detection, and OpenFlow. The Snort Agent is localized into the management domain on the XEN hypervisor. This work only evaluates the performance of the agent placement in the XEN hypervisor. The agent placed 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.

The problem of specific sensor placement is addressed by Chen et al., who propose a technique based on Genetic Algorithms (GA) for sensor placement [2]. The proposed algorithm minimizes the number of placed sensors and maximizes the rate of analyzed traffic. However, this proposal based on GA requires high processing time without warranting the solution convergence [5].

3 The proposed system

BroFlow employs a programmable software switch, Open vSwitch (OVS),[2] 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 packet dropping, packet-header fields modifying, etc. Each

[1]http://www.noxrepo.org/pox/about-pox/.

[2]http://openvswitch.org/

virtual switch runs on a virtual machine (VM), as it is deployed in FITS [11]. The POX network controller 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. Our system considers a hybrid network virtualized environment, composed of XEN virtual machines (VM) running over an OpenFlow switching matrix. As a hybrid environment (XEN+OpenFlow) [11], we consider XEN hypervisor for machine-based virtualization, and OpenFlow for network-based virtualization. In this virtualization environment, the VMs are connected through OpenFlow switches, implemented by Open vSwitch. Each virtual network (VN) consists a set of Virtual OpenFlow Switches, running on virtual machines (VM). All virtual networks could run over several physical machines (Fig. 1).

We choose the Bro [13] open source network traffic tool analyzer due to its high policies description language, which defines events for the network activities, represented by a packet abstraction in a higher information level. `Bro` language enables users to define their own policies. In addition, Bro inspects network traffic in real time, creating reports and alarms when a security policy is threaten.

The system architecture is based on the FITS [11] environment. FITS architecture is show in Fig. 2. FITS has a set of nodes geographically distributed among several institutions. A set of nodes is called an island. Islands are interconnected via virtual private networks (VPN) and generic routing encapsulation (GRE) tunnels to emulate layer-2 links over the Internet. FITS owns three kinds of nodes: manager, gateway, and operational nodes. Manager is responsible for coordinating the testbed, the gateway interconnects two islands, and the operational node hosts the experiment. Each operational node can instantiate virtual switches and interconnected them to support different virtual networks.

3.1 BroFlow sensors

Our system owns two types of sensors, the BroFlow virtual network (VN) sensors and the BroFlow infrastructure sensor, illustrated in Fig. 1. Every sensor executes a Bro tool daemon, with a minimal resources consumption. VN sensors are distributed between the virtual switches and monitoring either virtual switches or specific hosts. In each VN sensor, specific and independent policies for each virtual network are established. It is important in a cloud-virtualized environment, because the policy persists even when there is a virtual switch migration. Once the BroFlow sensor migrates, it carries its specific policies together. In addition, a daemon monitors system resources, which are consumed by both physical machines and virtual machines. In this architecture, the physical machines allocate several BroFlow virtual network sensors, which send notifications trough a secure communication channel to the network controller. The control communication between sensors and controller goes through an isolated network. In this way, we avoid security issues and possible delays during attacks. With distributed sensors, monitoring different networks, it is possible to observe behaviors that are not observed using a single point of observation [1].

The BroFlow infrastructure sensor runs parallel to the network controller in order to protect the physical network from malicious threats. An example of an infrastructure threat is the ARP flooding attack, known as ARP poisoning. ARP flooding overloads switches routing tables with fake MAC addresses, causing a memory DoS of switches. The infrastructure sensor detects and prevents these types of

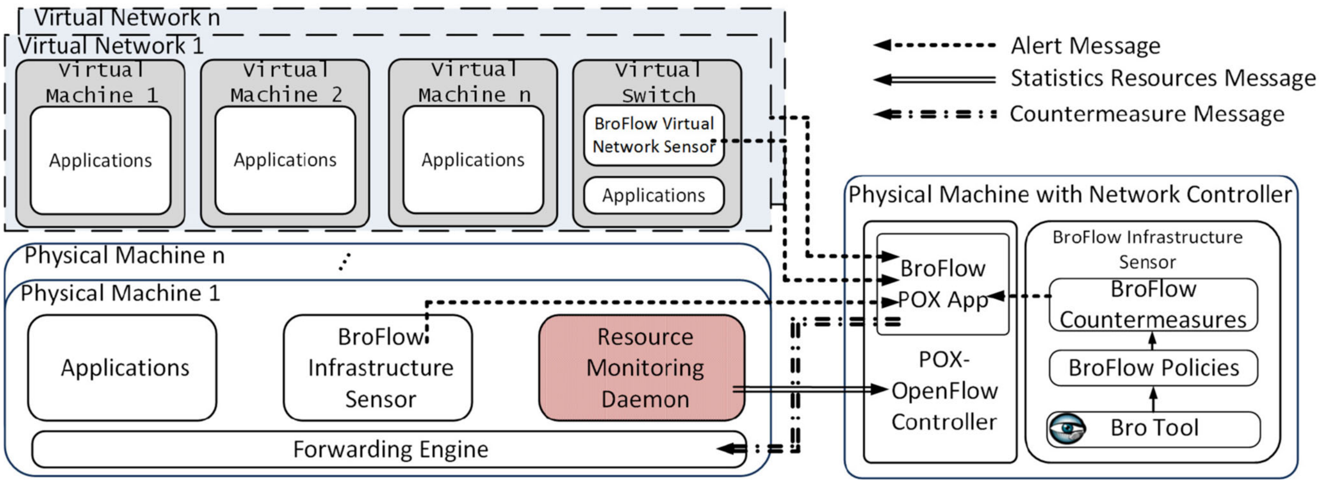


Fig. 1 BroFlow system Architecture. Sensors analyses network traffic to protect the physical infrastructure and virtual networks from attacks. All sensors communicate with the BroFlow application thought secure channels. Inspection modules provide elastic resources on demand. Countermeasures are applied to forwarding engines in order to stop a threat

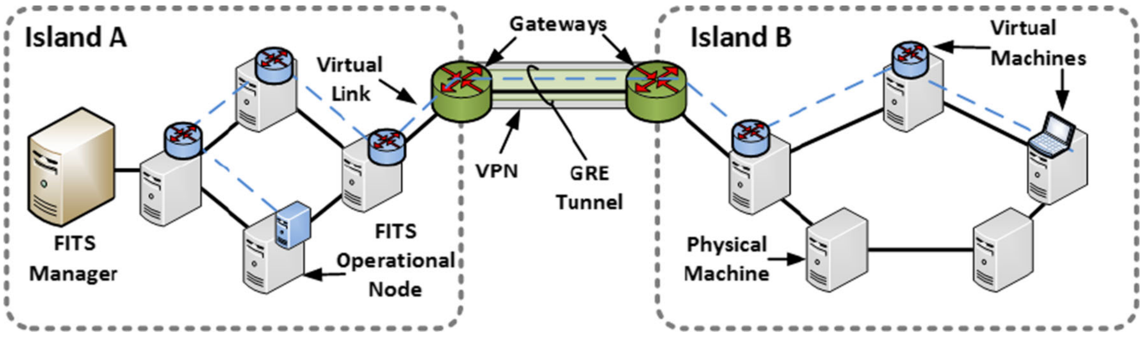


Fig. 2 Future Internet Testbed with Security (FITS) architecture. Gateway interconnect Islands, Operational nodes provide virtualization functions and manager node coordinates the testbed

attacks, protecting the physical network and the hosted virtual networks. Furthermore, BroFlow application, manages the alarms and countermeasures. We use the event engine and the policy interpreter from the original Bro traffic analyzer. The BroFlow policy module decides which events generated by Bro represent an attack and which action the BroFlow application must take.

3.1.1 BroFlow policy

BroFlow security policies are composed by two modules: network event inspection (NEI) and attack detection (AD). The NEI Module analyses in real time the relevant information, provided by the Bro tool, about the established flows during packet reception. Detection policies are written in `Bro` language and consist of DoS flooding: TCP-SYN, ICMP, 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. Two anomaly detection algorithms for packet flooding attack are implemented: ramp and adaptive thresholds [16]. The ramp algorithm sums packets during a specific period and raises an alarm when a threshold is reached. The 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. When the 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.

3.2 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. The countermeasures are applied in the forwarding engine in order to stop or mitigate a threat. When an attack is detected by the policy module, it sends an alarm message to the NC. 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 compose 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, the suspicious flows are explicit.

3.3 BroFlow application

The BroFlow application runs on the top of POX-OpenFlow network controller and receives alarms derived from sensors, executing the required countermeasures to answer those alarms. Thereby, when an alarm message is received from sensors, the BroFlow application match message content with its flow table. After that, the application indicates to the network controller the countermeasure to be taken in all network switches. In our prototype, the countermeasures are corresponding OpenFlow action: *drop* to block a flow, and *output* to forwarding packets to a specific switch port, deviating a flow to another host. Countermeasures are applied under a quarantine regime, that is, 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 was closed, ceasing to use system resources.

3.3.1 Elasticity under demand

Elasticity consists of the growth and reduction of resources according to the workload [3]. To prove this concept, we use the flexibility provided by SDN, to dynamically deviate traffic flows to be inspected in the BroFlow sensors. Considering the traffic workload, it is possible to instantiate or disconnect new sensors and balance flows, in order to analyze network traffic. Our system deviates the malicious flows to several BroFlow sensors when a machine is overloaded. For packet mirroring between the BroFlow Sensors, we use *Generic Routing Encapsulation* (GRE) tunnel. The packet inspection is done after the decapsulation, assuring packet integrity. Therefore, the flows distribution consider the packet source and system resources availability in each virtual machine. A flow of a new source is allocated in the less loaded processing machine and flows from the same source are allocated together in the same machine preventing attacks from going unnoticed. The system resources of each physical machine are monitored by *libvirt* library running as a daemon. This library monitors the physical machine resources, identifying which amount of resources are consumed by the virtual machines. This daemon is located in the privileged domain, called Domain 0, and monitors resources such as bandwidth, CPU, and memory. Statistics of all physical machines are aggregated in the 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 physical machine containing BroFlow sensors are analyzed together, in order to detect when a flow redistribution is possible, allowing to deactivate a machine in case of unload, ensuring the elasticity of the proposal.

4 Modeling and optimization of strategic IDPS sensor location

BroFlow sensors are spread in both physical and virtual networks, then sensor placement results into an optimization problem. In BroFlow, it is possible to establish a reduced number of sensors instead of placing sensors in every switch. This section models formally the problem of the sensor network placement. A heuristic is proposed minimizing the sensors number and maximizing the network coverage reached for each sensor. We assume that each node has enough processing power to analyze all incoming network traffic, as it is for example, in a datacenter.

Let $v \in V$ be a vertex, and $s \in S \subseteq V$, where S is the subset of nodes which are implemented as an IDS sensor. We use x^v as the variable that determines if vertex v holds a sensor

$$x^v = \begin{cases} 1, & \text{if } v \in S \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Each vertex sends a traffic unit to each other vertices. Therefore, the total amount of traffic T in the network equals $T = \frac{|V|(|V|-1)}{2}$. The traffic from the source i to destination j uses a single path, and we represent whether vertex v is in the path of i, j traffic by $t_{ij}^v = 1$, otherwise $t_{ij}^v = 0$. Thus, the total amount of traffic passing through vertex v is $t^v = \sum_{i \neq j} t_{ij}^v$. We also define an auxiliary variable $y_{i,j}$ to determine weather there is a sensor in the path from the source i to destination j

$$y_{i,j} = \begin{cases} 1, & \exists\, v \in V \mid x^v \wedge t_{i,j}^v \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

In this way, the total amount of traffic analyzed by the IDS sensors is

$$T_x = \sum_{i \neq j \in V} y_{ij}, \tag{3}$$

The objective function $F(x)$ represents the global cost to minimize composed by two objective functions: the number

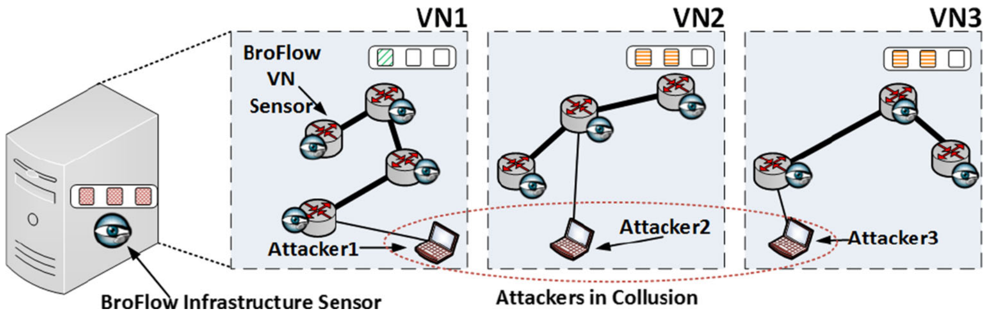
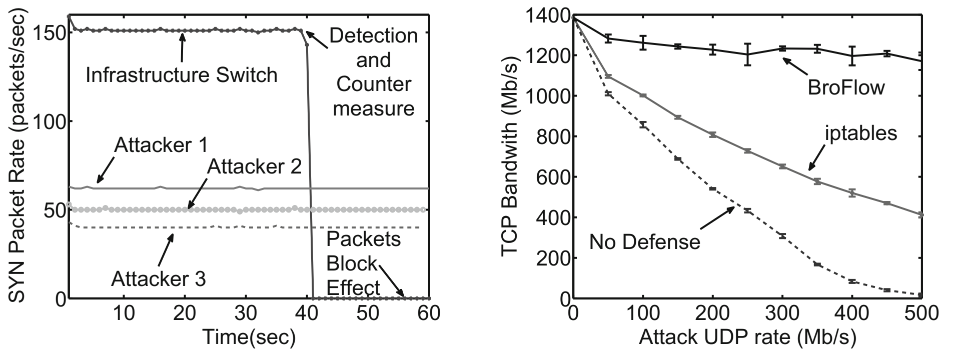


Fig. 3 Topology used in the first experiment, flooding attack against the infrastructure. One physical machine hosts three virtual networks. Each VN host a BroFlow sensor and one attacker acting in collusion. Sensor in the VN cannot detect each attack at the VN, but the attack is detected by the BroFlow infrastructure sensor as the sum of all malicious flow in the infrastructure



(a) Result of Experiment 1. Infrastructure threat detection. (b) BroFlow compared with iptables and no defense under a UDP attack at different rates.

Fig. 4 In Fig. 4a three attackers acting in collusion, detected by Infrastructure Sensor around 40 seconds. In Fig. 4b 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

of sensors in the network and the total network traffic coverage analyzed by all sensors. Then, we want to minimize the global cost

$$\min F(x) = F_{\text{sensor}}(x) - F_{\text{traf}}(x), \tag{4}$$

where $F_{\text{sensor}}(x)$ is the relation between the sensor nodes and the total of vertices in the network, expressed by

$$F_{\text{sensor}}(x) = \frac{1}{|V|}\sum_{v\in V} x^v, \tag{5}$$

and $F_{\text{traf}}(x)$ is the traffic percentage in relation to the total amount of network traffic that each sensor node analyze, given by

$$F_{\text{traf}}(x) = \frac{T_x}{T} = \frac{1}{T}\sum_{i\neq j\in V} y_{ij} \tag{6}$$

such that $0 \leq F_{\text{sensor}}(x) \leq 1$ e $0 < F_{\text{traf}}(x) \leq 1$. Besides, $\sum_{v\in V} x^v \leq |V|$, and $x^v \in [0, 1]$. This problem is reduce to a set covering problem (SCP) a nondeterministic polynomial time NP-hard problem.

To solve this problem, we use a greedy algorithm, which sorts the vertex list according to t^v, the amount of traffic that pass through the vertices. Thus, the algorithm chooses first central nodes that concentrate the largest amount of the traffic. We use as metric the amount of traffic $H(x) = T_x/T$ covered by the current sensors. Hence, after the selection of a sensor node, it is verified if the solution $H(x)$ reaches a target amount of traffic, and stops. Otherwise, the algorithm selects another node. The process is repeated until the sensor set covers the target amount of traffic, or when it is not possible to add more sensors. Greedy algorithms make locally optimal choices that eventually reaches a global optimal. Nevertheless, the computational cost execution is low comparing with other solutions.

We also used the Simulated Annealing (SA) optimization method, which has a proven convergence to a global minimal in an undetermined time. Every iteration selects a number of sensors and their positions at random and generate a new candidate solution that could be accepted in case it has a lower objective function $F(x)$ than the previous iteration. If the solution is worse than the previous, SA accepts the new solutions by a decreasing probability according to iteration. Thus, the solution may be accepted even if the objective function is higher to the previous one. This behavior is necessary to avoid the solution converge to a local minimal. The perturbation used to select the number of sensors and the positions follow a Cauchy distribution.

Even knowing that BroFlow uses specific control messages, it could be analyzed the possible impacts on the network links while the network is in operation transferring data to the nodes selected as a sensor. This problem can be modeled as another restriction in the formulation. Nevertheless, we leave this problem as a future work.

5 Results

We developed a BroFlow prototype in *Future Internet Testbed with Security*[3] (FITS), an interuniversity testbed for Future Internet proposals. FITS consists of distributed 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 [4], allowing a coexistence of parallel multiple networks running different applications. In FITS [11] the control plane executes in the XEN VMs and the packet forwarding is performed by OF. In addition, all experiment results are presented with a confidence interval of 95 %.

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

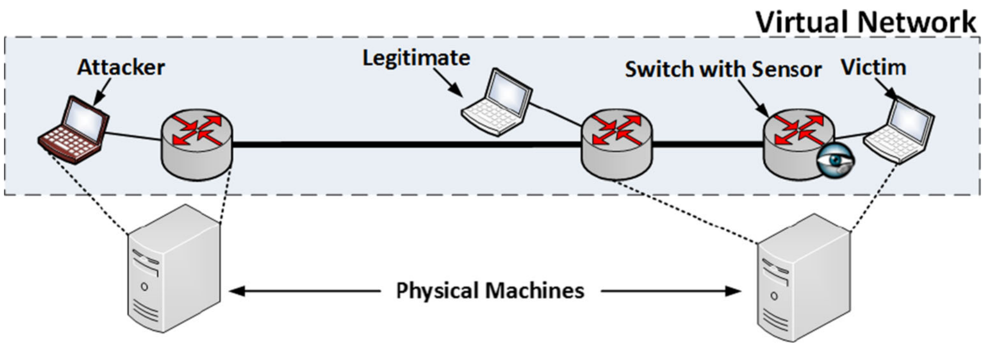


Fig. 5 Topology of the Second experiment. Comparison of different defense method in a Virtual Network. In the switch with sensor, it is compared BroFlow sensor with iptables and system without defense. In addition, this topology is used to evaluate the performance of BroFlow

5.1 Countermeasure evaluation

The first experiment analyzes the effect of a TCP-SYN packet flooding attack to the infrastructure when host several virtual networks. A physical machine hosting three different virtual networks constitutes the scenario, as shown in Fig. 3. All virtual switches run on a virtual machine. In every virtual network there is one attacker and one BroFlow sensor for virtual network, which analyzes the traffic of its network. Moreover, an infrastructure sensor is installed in the physical machine.

Figure 4a shows the experiment with three attackers sending SYN packets at different rates. It was defined, as a test criterion, the threshold in 100 SYN packets per second that represents the maximal SYN packet rate allowed in each network. As it can be seen in Fig. 3, in each in each VN the threshold is never passed, boxes are not full. Nonetheless, as the tenants act in collusion, the maximal established threshold is exceeded more than 50 %. The connection aggregated rate per second of the VNs is totally forwarded the physical machine 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, in Fig. 3, the box in the infrastructure sensor is full. At the detection moment, approximately at 40 s, avoid flash crowds, an alarm message is generated by BroFlow sensor located at the physical machine.

Each BroFlow sensor runs an adaptive threshold algorithm implemented in `Bro` language. This algorithm increments a counter k, when the mean rate value of the previous T time interval of 10 s 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 $T = 10$ s; successive threshold violation $k = 4$, being the same values adopted by the authors in [16].

The SDN network global view allows a DoS attacks to be blocked close to its source. This behavior is evaluated in the experiment 2, as shown in Fig. 5. This experiment has three virtual switches working in two physical machine, with a virtual link of 1500 Mb/s. 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 sharing the link with the attacker performs a TCP bandwidth measurement. We implement one BroFlow sensor in the last virtual switch, the switch with sensor, close to the victim. BroFlow is compared with iptables and with the system without any defense. Figure 4b 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

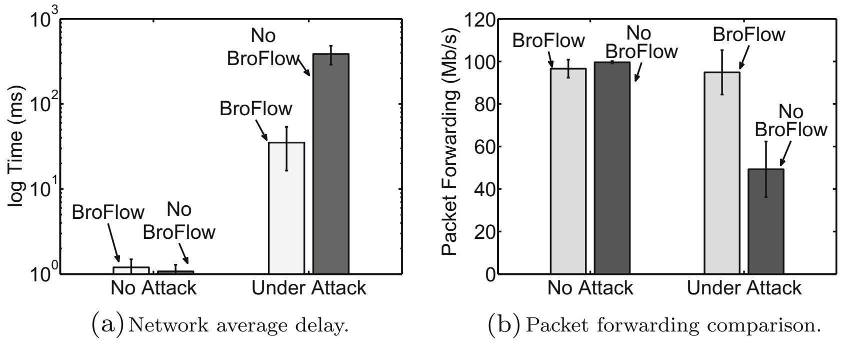
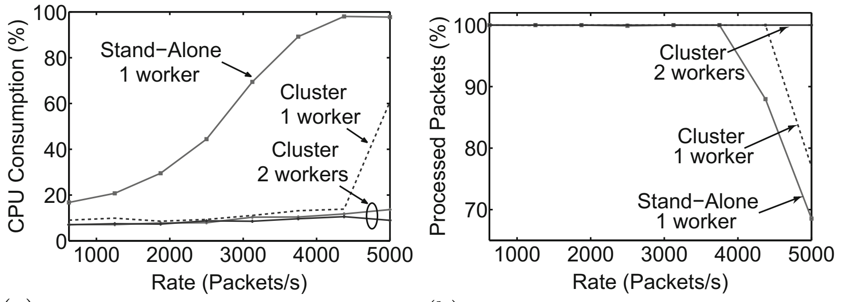


Fig. 6 BroFlow performance evaluation under and without attack. **a** The network average delay is reduced up to 90 % under attack scenario, time in logarithm scale. **b** The packet-forwarding rate of the system is minimum affected under attack with BroFlow

Fig. 7 CPU consumption and analyzed packet comparison. Bro traffic analyzer tool running in stand-alone and cluster with one and two cores configuration



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

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 Fig. 4b shows, the fall between 0 and 50 Mb/s, in case of BroFlow, it is due to the time taken to detect an attack.

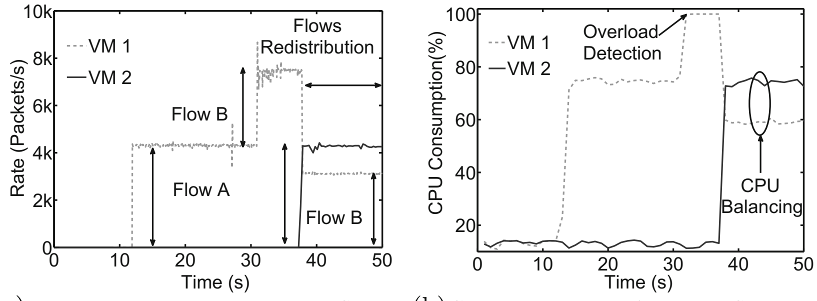
With the same topology of the Fig. 5, we evaluate the performance of our system. Then, Fig. 6 shows the system performance under a flooding attack. Figure 6b shows either the overload introduced by BroFlow as well as its efficiency blocking a DoS attack. Figure 6a compares the average packet forwarding 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 6b 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 maximal network average delay keeps almost unaffected with the BroFlow system.

5.2 Evaluation of resources consumed by bro

Once a threat is detected, with the use of SDN capability, our system is able to mirror the malicious traffic to be inspected in different analyzer machines. If the rate of the attack is high, it overloads the analyzer 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 analyzer machine depends on the security policy and the threat types. We generated increasing packet rates to analyze the amount of CPU spent by the machine and the perceptual of analyzed packets by Bro. The analyzed packet is the relation between packets send and packets received and analyzed by Bro tool.

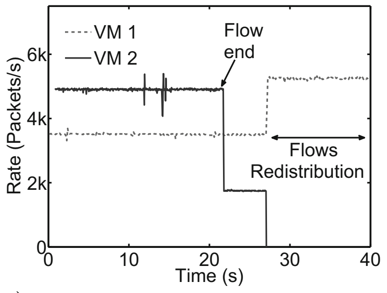
Bro natively executes as single-threading, using only one CPU core [13]. As it evolves, it executes as multi-threading, using several CPU cores, in which each process is called a *worker*. With the help of the `PF_RING` library instead of the `libpcap` library [13] is possible to increase the sniffing capacity. We use the same terminology of Bro creators [18], in which the workload is spread across many cores, or even many physical computers. We call cluster the group of *workers* that can be multiple cores in one machine or several physical machines used to handle the workload. In this experiment, we evaluate the use of both technologies, stand-alone and cluster, 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 *worker*, one core, and with two *workers*, two cores. Figure 7a shows cluster configuration running two *workers*. It is important to highlight that

Fig. 8 Analysis of the CPU consumption and Packet reception of the BroFlow sensor machines in an overload scenario

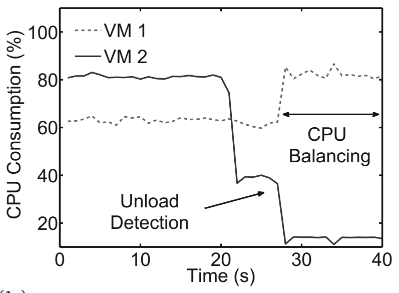


(a) Packet rate received by BroFlow Sensors in an overload scenario. (b) CPU consumption of BroFlow Sensors in an overload scenario.

Fig. 9 Analysis of the CPU consumption and Packet reception of the BroFlow Sensors in an unload scenario



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



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

under this technique, we only analyzed the CPU consumption of the *workers* threads that inspect traffic. It was not considered the proxy and manager process because they can be executed in separate machines.

Figure 7a shows Bro CPU consumption in a VM. In stand-alone configuration, the system saturates using all the CPU resources. Thus, after 3600 packets per second, CPU consumption reaches 100 %. Notwithstanding, for the same packet rate, in one *worker* cluster configuration, the system stays behind this limit and with two *workers* configuration, the performance is better. The CPU consumption increase was almost negligible under the maximal tested rate of 5000 packets per second. Figure 7b shows the difference between sent and received packets analyzed. Comparing these values with the ones in Fig. 7a, we observe that analyzed packets suffer a decrease when CPU consumption is maximum, this effect is notorious with one *worker* stand-alone configuration, reaching the analysis 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 % is analyzed, although in two-*workers* cluster configuration, under the same rate, no packets are dropped.

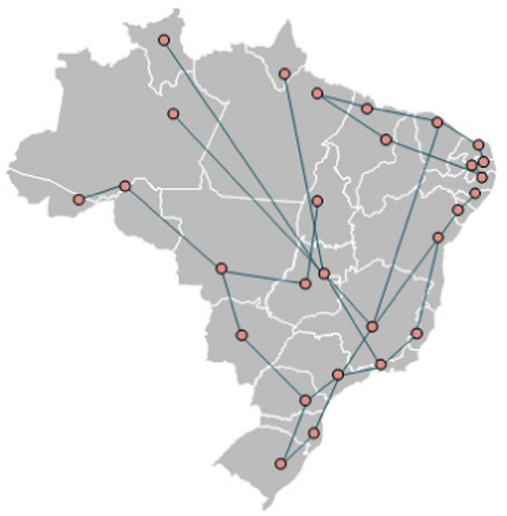


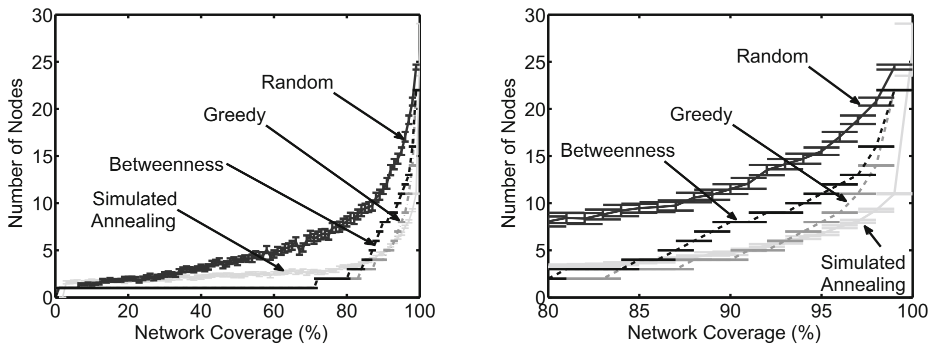
Fig. 10 Brazilian *Rede Nacional de Ensino e Pesquisa* (RNP) real topology, with 31 vertex and 34 edges

5.3 Elasticity under attack

Figure 8 shows when second flow starts, it overloads the BroFlow sensor machine, approximately at 30 s. 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 without overloading the BroFlow sensor. This test performs a temporal analysis of the machines in overload case. Two flows are initiated in one machine, Fig. 8a, causing a CPU overload as shown in Fig. 8b. 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. BroFlow sensors are deactivated and flows are redistributed by the NC if exist idle resources. Using the elasticity technique, we can see that our system is capable to support high rates of attacks, or several attackers against the system. In addition, as our sensors, nodes support all incoming traffic, we consider out system capable to scale to high rates of attack or attackers.

Figure 9 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 flow 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, Fig. 9a. Then, the NC analyze the CPU consumption of the two machines, as shown in Fig. 9b, redistributing flows in order to deactivate a sensor.

Fig. 11 Efficiency of the proposed placement method in relation with the number of sensors node to cover all network traffic



(a) Heuristic evaluation in a topology of 31 nodes. (b) Deeper detail of the network coverage vs. number of sensors.

5.4 Optimal sensor placement

To evaluate the proposed heuristic, we use a real topology from *topology zoo*.[4] The analyzed topology is the Brazilian Internet backbone network, *Rede Nacional de Ensino e Pesquisa* (RNP), that has 31 vertex with 34 edges distributed geographically in the Brazilian states. The real topology can be seen in Fig. 10.

We have evaluated our metric with two meta-heuristic solutions, greedy and simulated annealing, and with a random choice. In addition, it is implemented the choice by the betweenness centrality. Betweenness centrality is shown in Eq. 7, where σ_{ij} is the total number of shortest paths from node i to node j and $\sigma_{ij}(v)$ is the number of those paths that pass through v. The betweenness centrality consider all the nodes and paths in the network, while our proposal is relatively to each node.

$$g(v) = \sum_{i \neq v \neq j} \frac{\sigma_{ij}(v)}{\sigma_{ij}} \tag{7}$$

Figure 11 shows in percentage the covered traffic in relation with the number of IDS sensors. The system determine the best position that each node should be located warranting the maximal network coverage. With low network coverage, the behavior all approaches is similar. The random choice shows the worst behavior followed by the simulated annealing. Then, the simulated annealing improved it results over the random choice. The betweenness and the greedy proposal present an exact result until approximately 80 %. Figure 11b shows in higher detail the results. The random choice is still being the worst approach followed by the betweenness. Both approaches with our metric showed the best efficiency with high values of network coverage, minimizing the number of sensors nodes used. The simulated annealing has better behavior despite of it high computational cost. With 95 % of network coverage, the random solution use 15 nodes and the greedy and simulated annealing reduce it by two time using 7 nodes. Yet with 99 % the simulated annealing gain by more than two time compared with the random choice, choosing 11 nodes against 21 of the random choice.

[4]www.topology-zoo.org

6 Conclusion and future work

In this paper, we presented BroFlow, an Intrusion Detection and Prevention System (IDPS) for virtualized 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 are deactivated in case of unload. Therefore, the architecture provides resources according to demand. Furthermore, it was we created a heuristic to locate IDS sensor, optimizing the maximal network coverage with the lowest number of sensor.

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. We have proved that our proposal significantly reduces the sensors number, while keeping full coverage of network flows with an optimization problem.

As future works, we will correlate different BroFlow sensors alarms, taking into account the rules establishment into switches. In addition, we plan to integrate new metrics in our heuristic, such as network bandwidth, node loads, among others, in our optimization problem.

Acknowledgment We also thank Lyno Henrique Gonçalves Ferraz, Antonio Lobato and Ulisses Figueredo for their significant contributions to obtain the results.

References

1. Che X, Lopez J, Maag S, Morales G (2015) Testing trust properties using a formal distributed network monitoring approach. Ann Telecommun 70(3–4):95–105
2. Chen H, Clark JA, Tapiador JE, Shaikh SA, Chivers H, Nobles P (2009) A multi-objective optimisation approach to IDS sensor placement. In: Computational intelligence in security for information systems. Springer, pp 101–108
3. Coutinho E, de Carvalho Sousa F, Rego P, Gomes D, de Souza J (2015) Elasticity in cloud computing: a survey. Ann Telecommun 70(7–8):289–309
4. Fernandes NC, Moreira MDD, Moraes IM, Ferraz LHG, Couto RS, Carvalho HET, Campista MEM, Costa LHMK, Duarte OMB (2011) Virtual networks: isolation, performance, and trends. Ann Telecommun 66(5-6):339–355
5. Ferraz LHG, Mattos DMF, Duarte OCMB (2014) A two-phase multipathing scheme based on genetic algorithm for data center networking. In: IEEE global communications conference - GLOBECOM, pp 2270–2275
6. Giotis K, Argyropoulos C, Androulidakis G, Kalogeras D, Maglaris V (2014) Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. Comput Netw 62:122–136
7. Hu H, Han W, Ahn GJ, Zhao Z (2014) FLOWGUARD: building robust firewalls for Software-Defined Networks. In: Proceedings of the 3rd workshop on hot topics in SDN. ACM, pp 97–102
8. Mattos DMF, Duarte OCMB (2014) XenFlow: Seamless migration primitive and Quality of Service for virtual networks. In: IEEE global communications conference - GLOBECOM
9. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J (2008) OpenFlow: enabling innovation in campus networks. SIGCOMM Comput Commun 38:69–74
10. Mehdi SA, Khalid J, Khayam SA (2011) Revisiting traffic anomaly detection using software defined networking. In: Recent advances in intrusion detection. Springer, pp 161–180
11. Moraes IM, Mattos DM, Ferraz LHG, Campista MEM, Rubinstein MG, Costa L. H. M, de Amorim MD, Velloso PB, Duarte OC, Pujolle G (2014) FITS: a flexible virtual network testbed architecture. Comput Netw 63:221–237
12. Nam J, Jamshed M, Choi B, Han D, Park K (2015) Scaling the performance of network intrusion detection with many-core processors. In: ACM/IEEE symposium on architectures for networking and communications systems (ANCS), pp 191–192
13. Paxson V (1999) Bro: a system for detecting network intruders in real-time. Comput Netw 31(23–24):2435–2463
14. Piedrahita AFM, Rueda S, Mattos DM, Duarte OCM (2015) FlowFence: a denial of service defense system for software defined networking. In: Global information infrastructure and networking symposium -GIIS, p 6
15. Shanmugam PK, Subramanyam ND, Breen J, Roach C, Van der Merwe J (2014) DEIDtect: towards distributed elastic intrusion detection. In: Proceedings of the 2014 ACM SIGCOMM workshop on distributed cloud computing - DCC, pp 17–24
16. Siris VA, Papagalou F (2006) Application of anomaly detection algorithms for detecting SYN flooding attacks. Comput Commun 29:1433–1442
17. Sommer R, Paxson V (2010) Outside the closed world: on using machine learning for network intrusion detection. In: IEEE symposium on security and privacy, pp 305–316
18. Weaver N, Sommer R (2007) Stress testing cluster Bro. In: Proceedings of the DETER community workshop on cyber security experimentation and test. USENIX Association, pp 9–9
19. Xing T, Huang D, Xu L, Chung CJ, Khatkar P (2013) SnortFlow: a OpenFlow-based intrusion prevention system in cloud environment. In: 2nd GENI research and educational experiment workshop, pp 89–92
20. Yoon C, Park T, Lee S, Kang H, Shin S, Zhang Z (2015) Enabling security functions with SDN: a feasibility study. Computer Networks 1(1)