EL-SEC: ELastic Management of SECurity Applications on Virtualized Infrastructure

Nabeel Akhtar*, Ibrahim Matta*, Ali Raza* and Yuefeng Wang†
*Boston University, Boston, USA †Akamai Technologies, Inc., Cambridge, USA
{nabeel, matta, araza, wyf}@bu.edu

***Abstract*—The concept of Virtualized Network Functions (VNFs) aims to move Network Functions (NFs) out of dedicated hardware devices into software that runs on commodity hardware. A single NF consists of multiple VNF instances, usually running on virtual machines in a cloud infrastructure. The elastic management of an NF refers to load management across the VNF instances and the autonomic scaling of the number of VNF instances as the load on the NF changes. In this paper, we present EL-SEC, an autonomic framework to elastically manage security NFs on a virtualized infrastructure. As a use case, we deploy the Snort Intrusion Detection System as the NF on the GENI testbed. Concepts from control theory are used to create an Elastic Manager, which implements various controllers – in this paper, Proportional Integral (PI) and Proportional Integral Derivative (PID) – to direct traffic across the VNF Snort instances by monitoring the current load. RINA (a clean-slate Recursive InterNetwork Architecture) is used to build a distributed application that monitors load and collects Snort alerts, which are processed by the Elastic Manager and an Attack Analyzer, respectively. Software Defined Networking (SDN) is used to steer traffic through the VNF instances, and to block attack traffic. Our results show that virtualized security NFs can be easily deployed using our EL-SEC framework. With the help of real-time graphs, we show that PI and PID controllers can be used to easily scale the system, which leads to quicker detection of attacks.**

I. Introduction

Network Function Virtualization (NFV) has gained tremendous attention from the research community and industry. The idea of moving Network Functions (NFs) implemented by middleboxes (hardware appliances) from the user premises to a cloud infrastructure was proposed by *Sherry et al.* [1]. Their study of enterprise networks shows that middleboxes are a core part of the network infrastructure, where the number of middleboxes is at par with the number of routers in the network. Enterprises spend large sums of money buying proprietary hardware appliances, which are hard to upgrade and require a great deal of resources. Moreover, they showed that for most common middleboxes, *i.e.* firewalls, intrusion detection systems, and proxies, at least 32.6% of the failures are caused by overload or physical/electric failure. Moving middleboxes as Virtualized Network Functions (VNFs) running on a cloud infrastructure can greatly reduce the high capital and operational expenses while simplifying the configuration and deployment of VNFs. Moreover, failovers can be realized in a cloud environment through redundant resources [1]. However, moving NFs to a virtualized infrastructure introduces a new set of challenges, *e.g.*, state consistency, elastic management and monitoring of VNF instances. The resources in virtualized infrastructures need to be elastically managed using scaling to fulfill the system demand while keeping the cost low. Scaling can be classified as *vertical scaling* or *horizontal scaling*. Vertical scaling refers to the ability to add/remove allocated resources for existing VNF instances, such as CPU capacity, storage, and memory. Horizontal scaling refers to the ability to add/remove VNF instances. In this work, we focus on horizontal scaling, *i.e.,* VNF instances are added/removed based on the changing load on the system.

In this paper, we present EL-SEC, a framework for deploying security network functions on virtualized infrastructures. As a use case, we deploy the Snort Intrusion Detection System (IDS) as the NF on the GENI testbed [2]. In EL-SEC, the state of the VNF instances is shared with an *elastic manager* and an *attack analyzer*. The elastic manager balances the load across VNF instances and adds/deletes VNF instances to avoid overload conditions. The attack analyzer maintains a global state of the NF and intelligently creates a list of attackers that should be blocked. A *forwarding controller* updates the forwarding rules in the network to direct traffic to VNF instances, and to drop traffic from malicious hosts.
Our contributions in this paper are summarized below:

- We propose EL-SEC, a framework for deploying virtual network security functions on a virtualized infrastructure.
- As a use case, we deploy Snort IDS using EL-SEC on a virtualized infrastructure provided by the GENI testbed.
- We implement elastic management using concepts from control theory. Our results show that the system can be elastically managed using a PI or PID load controller and attacks are quickly detected.
- We use RINA [3], a clean slate internet architecture, to develop a distributed monitoring application for the EL-SEC. The application collects load and Snort alerts using a publish-subscribe architecture.
- We implement an attack analyzer that maintains a global state of the traffic by collecting Snort alerts from the VNF instances.

Moreover, our implementation on the GENI testbed demonstrates that GENI is capable of supporting a wide range of experiments.

In the initial version of this work [4], we showed that control theory can be used to balance the load across two VNF instances. This paper expands [4] in many significant ways:

- Our previous work compares a PI-based load balancing controller to a load-oblivious Round Robin based controller. In this work, we propose an *elastic manager* that is not only responsible for load balancing but also responsible for auto-scaling VNF instances. The elastic manager is generalizable to any auto-scaler, *i.e.*, we are not limited to PI/PID controllers.
- In this work, we generalize the problem to more than two VNF instances.
- Our previous work did not include attack detection. In this work, we include an *attack analyzer* module to detect attacks on the system.
- We propose EL-SEC, a generic framework for deploying

Yuefeng Wang's work was done while he was at Boston University.

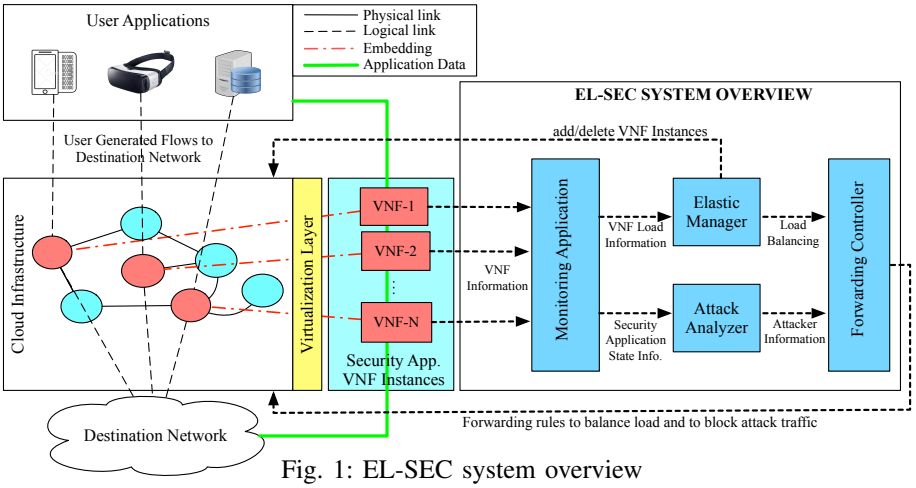


Fig. 1: EL-SEC system overview

security NFs on cloud infrastructures.

- We also implement a PID controller, along with a PI controller, to compare PI and PID control.
- We provide performance metrics related to auto-scaling and attack detection time. In our previous work, we only demonstrated load balancing using a PI controller.

Related work, such as *StatelessNF [5]*, *OpenNF [6]*, *Split/Merge [7]*, focuses on state consistency issues that arise due to flow migration and NF split/merge without giving details on how the resources are auto-scaled. Our work proposes and implements a complete VNF elastic deployment framework that addresses key challenges, including state consistency, auto-scaling, resource monitoring and flow migration.

The rest of the paper is organized as follows. Section II provides an overview of our EL-SEC framework. Section III describes a use case for EL-SEC where we deploy Snort IDS as VNF on the GENI testbed. Section IV explains the experimental setup. Section V presents our results. Section VI gives information on the reproducibility of our results and discusses experimental challenges. Section VII concludes the paper with a summary and future work.

II. EL-SEC Overview

This section provides an overview of EL-SEC and describes each component of our framework. EL-SEC aims to provide a framework for deploying security network functions on a cloud infrastructure. Moving resources from local premises to a cloud environment brings a new set of challenges, which includes constant monitoring of VNF instances, dynamically adding/deleting resources, balancing the load across the VNF instances, analyzing the system state to detect attacks and be resilient to system failures. EL-SEC facilitates security VNF deployment by addressing these key challenges. Figure 1 provides an overview of the system. User-generated flows traverse through security VNF instances (*e.g.*, firewall, intrusion detection system, etc.) running on a cloud infrastructure before reaching the destination network. The cloud infrastructure features an SDN enabled network, where the network can be programmed using common interfaces, such as OpenFlow [8]. To manage these security VNFs, we integrate EL-SEC with the system. In EL-SEC, a *Monitoring Application* gathers the state of the VNFs and provides this information to an *Elastic Manager* and *Attack Analyzer*. The *Elastic Manager* gets the VNF load information and balances the load across the VNF instances by providing a *Forwarding Controller* with load balancing directives. It also adds/deletes VNF instances based on the load on the current VNF instances. The *Attack Analyzer* gets information on the state of the security VNF instances (*e.g.*, traffic patterns, traffic alerts, etc.) from the *Monitoring Application* and identifies malicious hosts. The components of our EL-SEC framework are explained in detail next.

A. Monitoring Application

The Monitoring Application gathers the state of the VNF instances and shares it with different components of EL-SEC. The security VNF needs to share two important pieces of information with EL-SEC: i) a measure of load (*e.g.*, CPU load, traffic load or average packet delay) on the VNF instances, and ii) the application state of the VNF instances (*e.g.*, traffic patterns or intrusion alerts). The load of the VNF instances is used by the *Elastic Manager* to distribute load and add/delete VNF instances. The security application VNF instances share their state with a central entity (*Attack Analyzer*) to accurately and quickly detect attacks.

B. Elastic Manager

The Elastic manager is the heart of the system. It is responsible for balancing load and elastically scaling the VNF instances needed by the security NF. It gets the load of the VNF instances from the *Monitoring Application*. Note that simple round-robin balancers can be used to distribute load across the VNF instances. However, as we have shown in our previous work [4], load balancers based on control theory perform much better when compared with traditionally used round-robin techniques. The Elastic Manager calculates the ratio of traffic that should be diverted to different VNF instances. This information is shared with the *Forwarding Controller*, which then updates the traffic forwarding rules.

As the load on the system changes, the *Elastic Manager* is responsible for adding/removing VNF instances such that the minimal number of VNF instances are used while keeping the VNF instances from getting overloaded.

C. Attack Analyzer

The Attack Analyzer aims to keep the global application state of the VNF instances. Since traffic is distributed across the VNF instances, it is important to keep a global state of the VNF instances to accurately and quickly detect attack traffic and to maintain the application state of the NF in case of VNF instance's failure. Different techniques have been proposed for

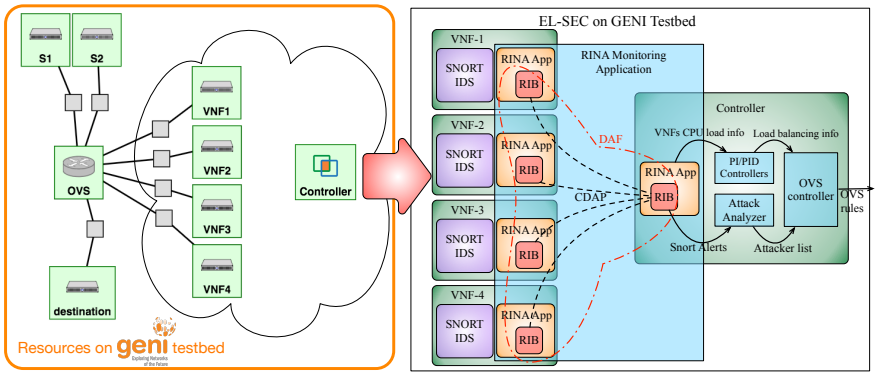


Fig. 2: EL-SEC use case: IDS on the GENI testbed

the stateful implementation of an NF. These techniques range from logging packet-level state information (*e.g.*, Stateless-NFs [5]) to event-based logging (*e.g.*, FTMB [9]). The EL-SEC framework is capable of supporting different techniques for VNF state management. Note that the Attack Analyzer is a centralized entity that has a global picture of the VNF instances and it can run different kinds of analysis (*e.g.*, machine learning) on traffic patterns to accurately and quickly detect and stop attack traffic.

D. Forwarding Controller

The *Forwarding Controller* is used to update the forwarding rules to either balance load across the VNF instances or to drop traffic from attacking hosts. The *Forwarding Controller* gets VNF load balancing directives from the *Elastic Manager* and the list of attackers from the *Attack Analyzer*. Note that for an SDN enabled network, the *Forwarding Controller* can be an SDN Controller.

III. Use case: Instantiating EL-SEC

In this section, we provide a use case of EL-SEC where we implement an Intrusion Detection System (IDS), namely Snort [10], as the VNF on a cloud infrastructure. We implemented the system on the Global Environment for Network Innovations (GENI) testbed [2]. The overview of the system is shown in Figure 2. Sources *S1* and *S2* are used to generate traffic to a *destination*, passing through an OpenFlow Virtual Switch (*OVS*). Traffic is duplicated on the *OVS* switch and sent to *off-path* VNF instances (*VNF1-4*) running Snort IDS. The *Controller* node is the “brain" of the system and it implements the EL-SEC framework. A distributed *Monitoring Application* running on the Recursive InterNetwork Architecture (RINA) [3], [11] is used to share VNF state information with other components of the EL-SEC system. RINA processes are running on the VNF nodes (along with Snort) to gather VNF state information and provide it to the *Controller* node. The *Elastic Manager* employs a control theoretic method, *i.e.*, Proportional Integral (PI) or Proportional Integral Derivative (PID) controller, to balance the load across the VNF instances. It informs the *OVS controller* of the fractions of traffic that should be directed to the VNF instances. The *OVS controller* then updates OpenFlow rules on the OVS switch to distribute duplicated traffic accordingly. The *Attack Analyzer* obtains Snort alerts from the RINA monitoring application. It analyzes the Snort alerts and informs the *OVS controller* about malicious traffic. The *OVS controller* updates OpenFlow rules on the OVS switch to drop all traffic from malicious hosts.

Note that this is our own instantiation/implementation of the EL-SEC framework using PI/PID controllers, the RINA monitoring application, Snort IDS, the OVS OpenFlow-based controller and Attack Analyzer. The modular structure of EL-SEC enables experimenters to extend/update different components of the EL-SEC system to satisfy their needs. EL-SEC provides a general framework that is capable of supporting a wide variety of on-path and off-path security virtual network functions (VNFs). Each component of the implemented system is explained in detail next.

A. GENI Testbed

GENI (Global Environment for Network Innovations) [2] is a nationwide suite of infrastructure that enables research and education in networking and distributed systems. GENI supports large-scale experimentation with advanced protocols for data-centers, clouds, mobile and SDN networks, *etc*. Since we needed to deploy the system on an edge-cloud system, GENI was a perfect candidate for it.

B. Snort IDS Application

Snort IDS [10] is an open-source network intrusion detection system. It has the ability to perform real-time traffic analysis on IP networks. It is one of the most widely deployed IDSes and it has been previously deployed on virtualized infrastructures to detect attacks [12]. We installed Snort on each VNF instance. We ran Snort in IDS mode to analyze traffic against the open-source Snort community rule set [10]. Snort performs deep packet inspection on incoming packets and generates alerts whenever it detects abnormal traffic. Snort’s deep packet inspection creates load on the VNF instances, thus change in incoming traffic changes load on the VNF instances running Snort. As the load on the system changes, we use EL-SEC to elastically manage system resources.

C. RINA Monitoring Application

The Monitoring Application collects VNF state information and shares it with other components of the EL-SEC system. We used the Recursive InterNetwork Architecture (RINA) [3], [11] to implement the Monitoring Application for EL-SEC. RINA is a clean-slate network architecture that overcomes inherent weaknesses of the current internet, *e.g.*, security and support for mobility and quality of service. For our system, we created a RINA monitoring Distributed Application Facility (DAF) consisting of monitoring processes running on each VNF instance. The RINA process on the controller node uses the DAF to get the state (CPU load and Snort alerts)

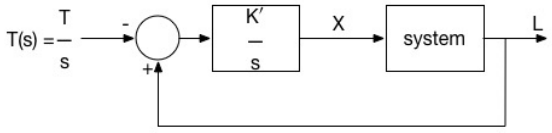


Fig. 3: Block diagram of the PI-controlled VNF system. System load L and target load $T(s) = \frac{T}{s}$ of VNF i is used to compute X, *i.e.* ratio of traffic diverted to VNF $i+1$

of the VNF instances. Details about RINA and the RINA monitoring application are explained in our earlier work [4]. Each monitoring application process on the VNF VMs periodically publishes its VNF load information and any Snort alerts generated. The RINA application process running on the controller VM subscribes to this information and passes the average over the last few measurements to the *Elastic Manager* for auto-scaling and load balancing. It also passes Snort alerts to the *Attack Analyzer* as soon as it gets the alerts. The *Elastic Manager* and *Attack Analyzer* are explained next.

D. Elastic Manager

The *Elastic Manager* provides scaling of resources and balances load across VNF instances. To elastically manage the resources for the cloud infrastructure, we used control-theoretic methods, namely a Proportional Integral (PI) or Proportional Integral Derivative (PID) controller, to distribute load across the VNF instances. The RINA Monitoring Application provides the PI/PID controller with the current load on the system. Based on the current load and previous load values, the PI/PID controller distributes the load across the VNF instances with the goal of minimizing the number of VNF instances needed while avoiding overloading any instance. As traffic changes, the CPU load on the VNF instances running Snort also changes. The PI/PID controller adds/removes VNF instances as the CPU load on the VNF instances changes. The PI and PID controllers are explained in detail next.

1) PI Controller: The Proportional Integral (PI) controller is a control-theoretic auto-scaler. The block diagram of the PI-controlled system is shown in Figure 3. Initially, when the load on the system is low, all traffic is diverted/duplicated to a single VNF instance running Snort IDS, *i.e.*, VNF1. The current CPU load $L_i(t)$ of VNF i is provided to the PI controller by the RINA monitoring application. The target CPU load T_i represents the maximum load allowed on VNF i.

When the current load $L_i(t)$ increases beyond its target load T_i, a fraction of the flows are diverted to VNF $i+1$ such that the load on VNF i does not exceed T_i. Assuming the "load error" at VNF i at time t is $e_i(t) = L_i(t) - T_i$, the PI control equation is given by:

$$x_{i+1}(t) = x_{i+1}(t-1) + K_i \; e_i(t) \qquad i \geq 1 \qquad (1)$$

where K_i is the controller's gain for VNF i, $x_{i+1}(t)$ is the fraction of new flows directed to VNF instance $i+1$ given previous instances $1, 2, ..., i$ have reached their target load.

2) PID Controller: The Proportional Integral Derivative (PID) controller has an additional derivative term of the "load error". The derivative term predicts system behavior and thus improves the settling time and stability of the system. The PID control equation is given by:

$$x_{i+1}(t) = x_{i+1}(t-1) + K_i^p e_i(t) + K_i^d \Big(e_i(t) - e_i(t-1) \Big) \quad (2)$$

Parameter	Description	Value
RINA Monitoring Application		
δt	time between consecutive VNF state information message	200 ms
C	number of measurements taken to calculate average CPU load	20
PI Controller		
T	Target CPU load on VNF instances	50 %
K	Integral gain	0.1
PID Controller		
T	Target CPU load on VNF instances	50 %
K^i	Integral gain	0.1
K^d	Derivative gain	0.1
Ryu Controller		
t_i	idle timeout	4 sec
t_h	hard timeout	4 sec
Traffic Generation (nping)		
r_p	number of packets per second sent for a flow	20/sec
s_p	average packet size for a flow	1400 B
t_f	average flow lifetime	150 sec

TABLE I: System Parameters

where K_i^p and K_i^d are the proportional and differential controller gains, respectively, for VNF i.

E. Attack Analyzer

The *Attack Analyzer* uses the application state information of the security VNF instances to detect attacks on the system. For our implementation of the *Attack Analyzer* on the GENI testbed, we use a log-based approach. Snort-alerts are logged at the *controller* node via the RINA monitoring application. The *Attack Analyzer* processes the Snort alert logs to detect attacks and generate a list of attackers (malicious hosts).

The Snort alerts provide a high-level state information of the VNF instances but do not provide packet-level details of the system that are sometimes necessary for security VNFs. However, the EL-SEC framework can support systems like StatelessNF [5] where packet-level state information can be collected from the VNF instances via the RINA monitoring application.

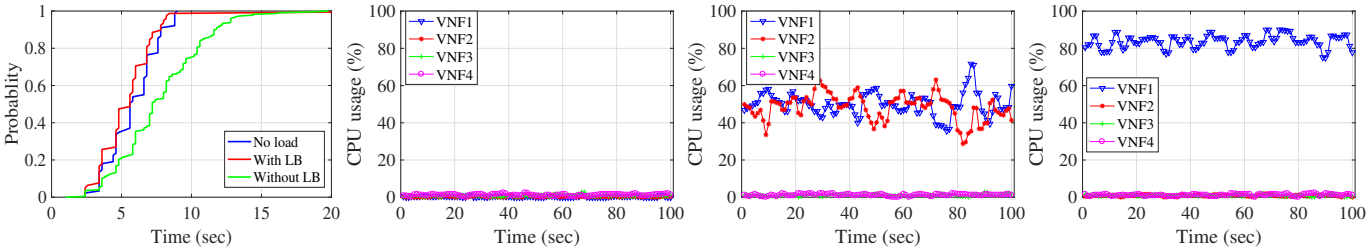
F. OVS Controller

The Open vSwitch (OVS) Controller is responsible for adding forwarding rules on the OVS switch for incoming flows. We use the Ryu OpenFlow controller [13] in our implementation. The OVS controller is responsible for two types of forwarding rules: i) rules to distribute load across the VNF instances, and ii) rules to block IP addresses associated with attackers. The OVS controller is provided load balancing directives by the *Elastic Manager* implementing the PI/PID controller. For each new flow, the OVS switch asks the OVS Controller about where to forward the flow. If the flow is for the destination node, the OVS Controller installs rules on the OVS switch that duplicate the packets of the flow and send them to one of the VNF instances running Snort IDS.

The Attack Analyzer provides the list of attackers to the OVS Controller. The OVS controller then installs forwarding rules to drop all packets originating from the malicious hosts.

IV. Experimental Setup

This section explains the experimental setup and the parameters used in our implementation. The reader can reproduce the experiment by following the detailed steps given in the tutorial, along with source code and experiment traces at [14].



(a) Time to detect port scanning attack (b) No load on the system (c) With Load Balancing (T = 50%) (d) Without Load Balancing

Fig. 4: Load on VNF instances when the system in under port scanning attack

1) Applications on VNF instances: There are two applications running on each VNF instance, the Snort IDS and the RINA monitoring application. Initially, we run Snort IDS on each VNF and provide it with the Snort community rule set to perform deep packet inspection on the incoming packets. Next, we run the distributed RINA monitoring application on the VNF instances and Controller node. The RINA monitoring application processes running on the VNF instances publish their CPU load and Snort alerts, and the RINA monitoring application process on the Controller node subscribes to these updates. Parameters of the RINA monitoring application are shown in Table I. The RINA monitoring application processes on the VNF instances take the average of $C = 20$ CPU load measurements and publish it every $\delta t = 200ms$.

2) PI/PID controller: The PI/PID controller running on the Controller node receives CPU load information from the RINA monitoring application. Next we start the PI or PID controller. The parameters for the PI/PID controllers are shown in Table I. The target load T is set to be 50% and values for the controller gains (K, K^i and K^d) are set to 0.1. Due to lack of space, we do not include the analysis of stability of the PI/PID controllers, but it can be shown that these controller gain values, given other system parameters, are sufficient for stability.

3) Attack Analyzer: Next we run the *Attack Analyzer* on the *controller* node. Our simple implementation of the *Attack Analyzer* receives Snort alerts through the RINA monitoring application and parses them for attacks. In this paper, the Attack Analyzer is configured to only analyze alerts for port-scanning attacks. It generates a list of hosts responsible for the attacks, and the OVS controller uses this list to block all traffic from these hosts. Note that the Attack Analyzer can be configured to detect other types of attack as well.

4) Configure OVS switch and OVS Controller: Next we configure the OVS switch and connect it to the Ryu [13] OVS controller. For each incoming flow, the OVS switch asks the Ryu Controller about the forwarding port for the flow. The flow's idle timeout (t_i) and hard timeout (t_h) values are shown in Table I. Flows expire after t_i seconds of inactivity, and after t_h seconds regardless of activity.

5) Traffic Generator: Background traffic is needed to generated load on the system. Traffic is generated using the nping application [15], which is an open-source tool for network packet generation and response time measurement and analysis. Parameters for the traffic generator are given in Table I. Each flow is randomly generated at source *S1* or *S2*. Each flow is randomly assigned a source IP address (which is different from the IP addresses of hosts *S1* and *S2*). The destination IP address is that of the *destination* node. Packets are sent at the rate (r_p) of 20 per second, and the average packet size (s_p) is 1400 bytes. The average lifetime for a flow (t_f) is 150 seconds. At a given time, multiple flows are generated to put the required load on the system.

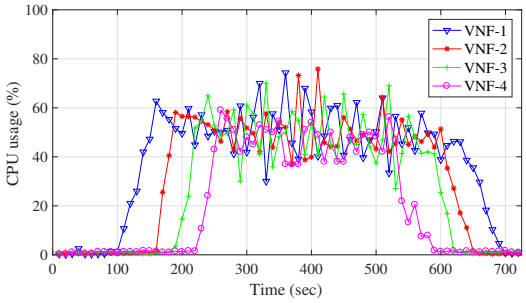
6) Attack Generator: The Attack Generator uses the port scanning application nmap [16] to perform a port-scanning attack on the destination node. All traffic, including for port scanning, is duplicated to the VNF instances. Whenever Snort IDS running on a VNF instance detects the port-scanning attack, it generates an alert, which gets communicated to the *Attack Analyzer* by the RINA monitoring application.

V. Results

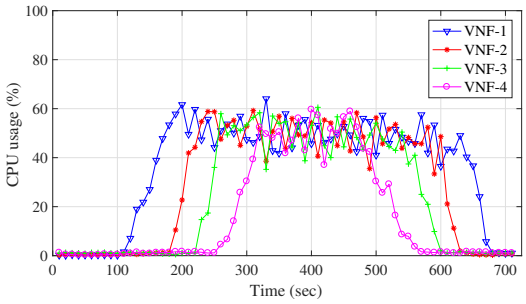
This section presents the performance results obtained from our EL-SEC system implementation on the GENI testbed. We use different performance metrics to test our implementation. We measure the time taken to detect and stop a port-scanning attack with and without load on the system. Moreover, we look at the effect of Elastic Management on the attack detection time. We also provide a comparison of the PI vs. PID controller.

Figure 4a shows the CDF of the time taken to detect a port-scanning attack under different CPU load on the VNF instances running Snort IDS. The *No load* scenario is shown in Figure 4b. In this scenario, there is no background traffic when the port-scanning attack is performed, so there is very little load on the VNF instances. Note that the port-scanning attack by itself does not result in much CPU load due to Snort IDS processing. The *With LB* scenario is shown in Figure 4c. In this scenario, background traffic is generated using the Traffic Generator described in IV-5. The Snort IDS performs deep packet inspection on the incoming packets and this generates significant load on the system. Using the PI controller, the load is balanced between VNF1 and VNF2. The CPU load on VNF1 and VNF2 is around the target load of 50%. The *Without LB* scenario is shown in Figure 4d, where the background traffic is not balanced, so all the traffic is sent to VNF1. The CPU load on VNF1 is around 90%. As seen in Figure 4a, the time taken to detect the attack is significantly larger when there is a high load on the VNF hosts. However, with the elastic management of resources and the load balancing (Figure 4c), the time taken to detect the attack is comparable to that of the unloaded system (Figure 4b). In the loaded scenario (Figure 4d), the Snort IDS is unable to process packets at line rate, thus packets are queued, and when the queue reaches its capacity, packets are dropped. This leads to significant performance degradation due to overload of the VNF instances running Snort IDS.

Next, we show how our system scales when we increase the load on the system. Figure 5 shows load balancing under the PI and PID controllers. Initially, there is no load on the system. The target load is set to 50% ($T = 50\%$). As seen in Figure 5a for the PI controller and Figure 5b for the PID



(a) PI controller (T = 50%)



(b) PID controller (T = 50%)

Fig. 5: Load balancing with PI and PID controllers

controller, if the CPU load on a VNF instance exceeds the target load T, a fraction of the incoming traffic is diverted to other VNF instances. Initially, all (duplicated) traffic is sent to VNF-1. When the load on VNF-1 exceeds T, a fraction of the traffic is sent to VNF-2. The same happens as the load on VNF-2 and VNF-3 exceeds the target, and so a fraction of the traffic is diverted to VNF-3 and VNF-4, respectively. Then, we gradually decrease the load on the system by decreasing the rate of incoming flows. As expected, initially the CPU load on VNF-4 gradually goes down as other VNF instances (VNF-1, VNF-2, and VNF-3) are able to process a larger percentage of incoming flows. As the rate of incoming flows goes further down, the CPU load on VNF-3, VNF-2 and finally VNF-1 also gradually goes down. The results show that both PI and PID controllers can be used to elastically manage load on the VNF instances. However, as expected, the PID controller (Figure 5b) has smaller oscillations around the target load (T) compared with the PI controller (Figure 5a), thus giving better performance.

VI. Reproducibility and Discussion

To reproduce the results shown in this work, we created a detailed tutorial, along with source code and experimental traces at [14]. We also include real-time monitoring graphs for a live demonstration of the EL-SEC system performance.

Our implementation on the GENI testbed shows that GENI is capable of supporting a wide range of experiments. Because of the distributed nature of the work, EL-SEC cannot be deployed in simulators or single-machine based emulators (*e.g.*, mininet [17]). GENI provides the distributed virtualized environment that can be used to fully implement and test the capabilities of EL-SEC.

VII. Conclusion and Future work

EL-SEC is a general framework that can be used to support security network functions on a virtualized infrastructure. EL-SEC uses a modular approach, where different components of the EL-SEC system are combined to implement a desired behavior. As a use case, we implement an intrusion detection system (IDS) on the GENI testbed using EL-SEC. The Snort IDS application is used as the security VNF. RINA, a clean slate internet architecture, is used to develop a distributed monitoring application to communicate Snort and VNF state information to the controller node. A PI or PID controller is used to elastically manage the load on VNF instances, and to add/delete VNF instances depending on the system load. An Attack Analyzer processes alerts from all the Snort instances running as VNFs and generates a list of attackers whose traffic is stopped by the OVS controller.

Our results further show that control theoretic load managers (*e.g.*, PI or PID) can be used to elastically manage resources on the virtualized infrastructure. Moreover, we show that elastic management of resources enables quicker detection of attacks.

Experimenters can use EL-SEC to deploy different security VNFs on a virtualized infrastructure. Different components of EL-SEC can be extended to support a variety of experiments. We aim to extend EL-SEC to include support for "serverless computing" [18]. EL-SEC can also be extended to support machine-learning based elastic management of resources. The Attack Analyzer can also use techniques from machine learning to learn from traffic patterns and efficiently detect attacks.

Acknowledgement

We would like to thank Marzieh Babaeianjelodar and Yaoqing Liu for their help during the initial phase of this work.

References

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *ACM SIGCOMM*, 2012, pp. 13–24.
[2] GENI, http://www.geni.net/.
[3] Boston University RINA Lab, http://csr.bu.edu/rina/.
[4] N. Akhtar, I. Matta, and Y. Wang, "Managing NFV using SDN and control theory," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 1113–1118.
[5] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless network functions: Breaking the tight coupling of state and processing," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA, 2017, pp. 97–112.
[6] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," in *SIGCOMM*, 2014, pp. 163–174.
[7] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," in *10th USENIX Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 227–240.
[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, 2008.
[9] J. Sherry, P. X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. a. Martins, S. Ratnasamy, L. Rizzo, and S. Shenker, "Rollback-recovery for middleboxes," ser. SIGCOMM '15. ACM.
[10] SNORT, https://www.snort.org/.
[11] Y. Wang, I. Matta, and N. Akhtar, "Application-Driven Network Management with ProtoRINA," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2016), April 2016*, March 2015.
[12] V. Mishra, V. K. Vijay, and S. Tazi, *Intrusion Detection System with Snort in Cloud Computing: Advanced IDS*. Singapore: Springer Singapore, 2016, pp. 457–465.
[13] Ryu Controller, http://osrg.github.io/ryu/.
[14] EL-SEC Webpage, tutorial, source code, http://cs-people.bu.edu/nabeel/ELSEC/.
[15] nping, https://nmap.org/nping/.
[16] nmap, https://nmap.org/.
[17] Mininet, http://mininet.org/.
[18] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with openlambda," in *USENIX HotCloud 16)*, Denver, CO, 2016.