This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2640297, IEEE Transactions on Network and Service Management

Analytical Model for Elastic Scaling of Cloud-based Firewalls

Khaled Salah*, Prasad Calyam+, Raouf Boutaba++
Khalifa University of Science, Technology and Research (KUSTAR), UAE*,
University of Missouri-Columbia, USA+
University of Waterloo, Canada, and the Division of IT Convergence Engineering, POSTECH, Pohang, Korea++
Email: *khaled.salah@kustar.ac.ae*, *calyamp@missouri.edu*, *rboutaba@cs.uwaterloo.ca*

***Abstract*—This paper shows how to properly achieve elasticity for network firewalls deployed in a cloud environment. Elasticity is the ability to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible. Elasticity for cloud-based firewalls aims to satisfy an agreed-upon performance measure using only the minimal number of cloud firewall instances. Our contribution lies in determining the number of firewall instances that should be dynamically adjusted in accordance with the incoming traffic load and the targeted rules within the firewall rulebase. To do so, we develop an analytical model based on the principles of Markov chains and queueing theory. The model captures the behavior of a cloud-based firewall service comprising a load balancer and a variable number of virtual firewalls. From the analytical model, we then derive closed-form formulas to determine the minimal number of virtual firewalls required to meet the response time specified in the SLA (Service Level Agreement). The model takes as input key system parameters including workload, processing capacity of load balancer and virtual machines, as well as the depth of the targeted firewall rules. We validate our model using discrete-event simulation, and real-world experiments conducted on Amazon Web Services (AWS) cloud. We also provide numerical examples to show how our model can be used in practice by cloud performance/security engineers to achieve proper elasticity under fluctuating traffic load and variable depth of targeted firewall rules.**

***Index Terms*—Cloud Computing, Firewalls, Cloud Firewalls, Scalability, Elasticity, Resource Management.**

I. Introduction

In a cloud environment, and as in any enterprise network, firewalls are typically used to filter traffic and enforce a given security policy. Today's cloud infrastructure and services offer a customer the ability to provision a customizable "virtual private cloud" (VPC) that comprises of many virtual machines that can be logically isolated, networked and configured into different subnets. A VPC provides higher control of the infrastructure to the customer, and provides flexible options to run isolated or single tenant hardware to support AWS instances. The VPC subnets can host a wide range of private and public services and applications, and a customer can interconnect their VPC to a corporate data center that is located at a different geographical site on the Internet.

The fact that the VPC is exposed with public IP addresses (from Amazon's public IP address pool), security becomes a major concern and there is a need for a suitable design when deploying the VPC so that it can securely handle elastic customer traffic needs. More importantly, the VPC has to remain operational under the threat of new cloud-based attacks that are always emerging [1]–[3]. In most cases, cloud service providers give minimal support towards security or protection of a VPC; it is the responsibility of the VPC customer to implement the proper performance scaling and adequate security measures by deploying the appropriate security appliances. As a consequence, the cloud customer has to undertake majority of control for management of network services i.e., the customer has to treat the VPC network as no different from a traditional non-cloud enterprise network, requiring the implementation of firewalls, load balancers, anti-viruses as well as intrusion detection systems.

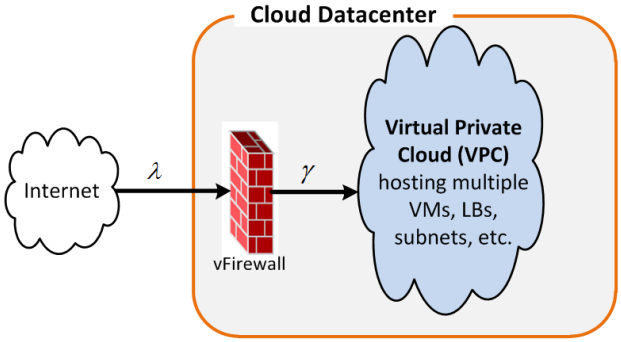

Figure 1 – Securing a VPC using a virtual firewall

To protect and secure a VPC deployment in a public cloud platform, typically virtual firewalls (vFirewalls or vFWs) are deployed as a first line of defense to filter lawful and unlawful packets, as shown in Figure. 1. A virtual firewall is basically a virtual machine that runs different software-based firewall functions such as e.g., access control lists. Unlike traditional firewalls, cloud-based firewalls need to be elastic in order to seamlessly integrate with the other elastic services related to e.g., compute (EC2), storage (S3) or service monitoring (CloudWatch). Elasticity is a key characteristic of cloud-hosted services and applications, whereby cloud resources are allocated and de-allocated based on the presented workload in order to satisfy given SLA (Service Level Agreement) performance metrics which may include response time, throughput and request loss ratio.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2640297, IEEE Transactions on Network and Service Management

Elasticity for cloud-hosted applications and services (such as web, FTP, and email service, multimedia streaming, customer relationship management (CRM), and many others) has been investigated to a great extent in the literature. However, the literature is lacking research work and investigation for implementing proper elasticity of virtualized network services residing in the cloud as those of routing, domain name services, intrusion detection, firewalls, etc. This article focuses on addressing the challenging issue of elasticity for a cloud-based firewall service. The article proposes an architectural design for an elastic scalable virtual firewall service to be deployed at cloud datacenters that support VPC services. Specifically, it focuses on how to tune elasticity to desired configurations by developing an analytical model that estimates the number of vFirewalls required to meet a certain SLA response time expected from a cloud-based firewall service. The firewall response time can be part of the overall end-to-end latency for an application or service hosted within the VPC specified in the SLA.

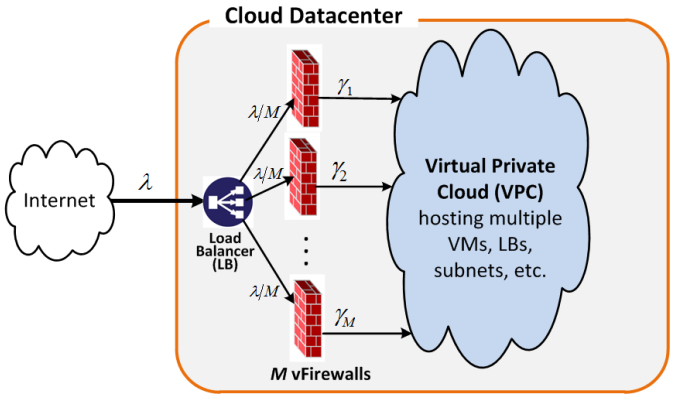

Figure 2 – An architecture of an elastic cloud-based firewall service

Figure 2 shows a typical architecture for an elastic cloud-based firewall service that can be deployed at a cloud datacenter. The firewall service comprises of a Load Balancer (LB) which fronts a variable number of vFirewalls (a.k.a. vFWs) that get allocated and de-allocated based on the received traffic from the Internet. A vFirewall is basically a software-based firewall running on a compute virtual machine (VM) of a certain processing capacity. The primary function of the LB is to distribute the incoming workload λ of arriving packets evenly among M vFWs so that each vFW will receive λ/M of the workload. In the figure, γ_j is the departure rate or throughput of a vFW$_j$.

In this article, we present an analytical model and provide closed-form solutions and formulas that can be used to support and enable elasticity for cloud-based firewalls. These formulas provide an answer to the question on the number of vFW$_s$ required for a given network workload. The answer to this key question is at the heart of any proper elasticity implementation. The answer to this key question is at the heart of any proper elasticity implementation. In this article, we do not claim to provide a full-fledge implementation to elasticity for vFW$_s$, but we focus on this key question. Our analytical formulas and solutions can be an integral part to any elastic scaling framework, design or implementation. The implementation of a full-fledged elastic cloud-based firewall system is beyond the scope of this article and is left as a future work.

Figure 3 shows the main design components for implementing a cloud-based firewall system. The components and their roles can be described briefly as follows:

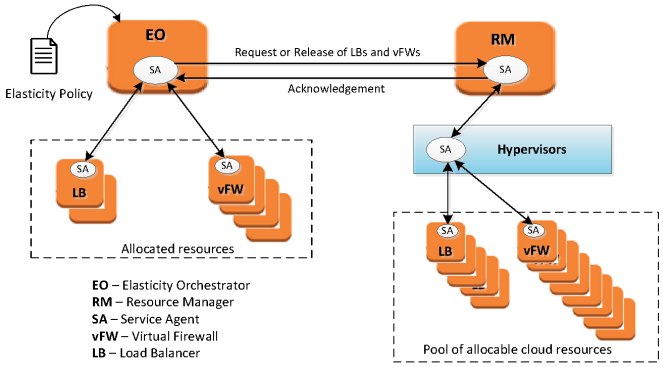

Figure 3 – Key design components for an elastic cloud-based firewall system

- **Service Agents (SA).** The measurements of the various cloud node states and network load have to be performed in real time by service agents which are placed in various parts of the VPC network including LB, vFW, hypervisors, and routers. Such agents will measure and monitor various aspects of node system parameters and network conditions. For example, the vFW agents will gather fine-grained statistics on the highly frequent triggered rules within the rulebase from each vFW.
- **Elasticity Orchestrator (EO).** The EO is responsible for the provisioning (scaling out) and de-provisioning (scaling in) of new vFWs. The EO carries out such functionality by gathering information from the agents, and subsequently deciding on the number of needed vFWs. The orchestrator runs periodically an algorithm to ensure the created vFWs are meeting the workload demand, and would adjust the allocated vFWs accordingly, based on the formulas and the analytic solutions given in this article. The length of the adjustment period and other parameters including the desirable SLA requirements are included as part of the "elasticity policy".
- **Resource Manager (RM).** The RM is the third key component of elasticity. This node is also known in some systems as the Virtual Infrastructure Manager (VIM) which is responsible for monitoring, automation, and management of cloud resourcesparticularly, in keeping track of used and unused cloud resources. The EO interacts with the Resource Manager at the time of provisioning new vFWs, and also at the time of releasing already allocated vFWs. As shown in Figure 3, the RM interacts with the hypervisors deployed on physical machines throughput the cloud datacenter, to manage and allocate virtual resources.

There is also a number of design and management aspects to consider for implementing and supporting elasticity for cloud-based applications and services [38]–[40]. Many of these

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2640297, IEEE Transactions on Network and Service Management

aspects need to be incorporated to implement a complete elastic cloud-based firewall system.

This paper is a significant extension of our preliminary work presented in [4]. Our main contributions of this paper can be summarized as follows:

- We propose a mathematical model that captures the behavior and dynamics of cloud-based firewalls. The model can be used to study the performance of cloud-based firewalls and also to determine the minimal number of cloud firewalls needed to achieve elasticity.
- We derive important closed-form formulas and present a complete algorithm to implement elasticity. Moreover, we propose a deployment topology of a cloud-based firewall service comprising LB and vFWs.
- We conduct a major experimental work and measurements to validate our analytical model. The experiments were conducted in a real-world VPC environment on the popular Amazon Web Services cloud infrastructure.
- We provide several numerical examples to show how proper elasticity can be achieved, and we offer guidelines for researchers as well as cloud performance/security engineers to be able to reproduce the results and conduct the experiments, as well as to deploy and implement an elastic cloud-based firewall service.
- Finally, and in general, our model, formulas, experiments, and guidelines presented in this article can be used and applied to implement and deploy other types of similarly-behaving rule-based services, systems, and applications to be deployed on the cloud. Such services may include intrusion detection systems, spam email filters, anti-virus appliances, etc.

The rest of the article is organized as follows. Section II summarizes work related to cloud-based firewalls. Section III presents our analytical model capturing the inner-working and behavior of cloud-based firewalls. In particular, we derive closed-form formulas to estimate the SLA response time incurred at both LB and vFWs, and devise an algorithm for the computation of these formulas. Section IV verifies and validates our model using simulation and an experimental testbed deployed within an Amazon Web Services environment. We also provide numerical examples and show how the model can be used to achieve elasticity of cloud firewalls that satisfy a given SLA response time. Finally, Section V concludes our article.

II. Related Work

To the best of our knowledge, there has been limited studies of elastic cloud-based firewall services in the literature. In [5], [6], a firewall framework of a cluster of firewalls was proposed to protect cloud-hosted applications and services. The authors used queuing theory to model and capture the behavior of the firewalls in order to study performance. In [7], a cloud-based firewall service is proposed to outsource firewall functionality from an enterprise local network to the cloud platform. The authors proposed a framework to preserve privacy and evaluate different algorithms to study the service performance issues.

In [8], [9], cloud-based firewalling was advocated as a future trend in which a traditional physical firewall can be outsourced to the cloud. In [10], a hybrid cloud-based firewalling service was proposed. The hybrid firewalling service is composed of a physical server infrastructure part and a cloud-based part. As the network traffic increases beyond a certain level, the traffic is routed to the cloud firewalls to be handled at-scale. Such an approach was proven to be potentially effective for mitigating DDoS attacks [11], [12] by making a decision to create or not to create virtual firewalls based on traffic demand.

In all of this prior work, the scalability and elasticity issues of a cloud-based firewall service for critical services such as VPC were not addressed. Also, all of this prior work did not consider the role of a load balancer and its critical impact on the performance. In general, these existing models do not capture the inner-working details and dynamics of the elastic services–which makes it difficult to derive proper guidelines for handling the tradeoffs inherent to cloud-based firewall elasticity design. As we noted, the LB can play a key role in creating, monitoring, managing, and orchestrating vFW instances in the cloud. All of this may result in a significant processing overhead at the LB, especially when the incoming workload is high. Hence, any analytical model should account for the role of LB in order to accurately model behavior and performance. Accordingly, this article develops an analytical model used to determine the efficient number of needed vFWs to meet a given SLA response time taking into account many key design parameters including, the LB as well as vFW processing capacity, workload, and depth of firewall rulebase.

III. Analytical Model

In this section, we develop an analytical model to capture the service behavior within a vFW, and then we derive formulas to estimate the response time. Typically, for a PC-based or virtual firewall, and as shown in Figure 4, incoming packets are queued into a Rx DMA ring and then go into three stages of service. In *Stage 1*, packet pre-processing takes place whereby the packet is removed from the queue, header fields are checked for errors, and packet is prepared for delivery to upper layers. Firewall rulebase interrogation takes place at *Stage 2* in which rule conditions are checked sequentially one by one until a match occurs. *Stage 3* executes the rule in which an action to drop, log, or pass the packet takes places. In our model, a new packet only gets forwarded to *Stage 1* after the previous packet has left *Stage 3* completely, i.e., it has departed from the entire queuing system. We also assume that the execution of the three stages is mutually exclusive. More specifically, if the CPU is executing one of the stages, the other two stages are halted. This is realistic considering vFWs are all based on an x86 architecture with a virtual CPU executing one task at a time.

The behavior of firewall processing of incoming packets can be modeled as a *finite* queueing system size with three stages of service. As shown in Figure 5, an incoming network packet gets first queued in a buffer of size K-1 and then gets served sequentially in three stages with each stage having a different mean service rate, i.e., μ_1, μ_2, μ_3. We assume that incoming packets follow a Poisson arrival λ. Also, the stage service times are independent with exponential distribution.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2640297, IEEE Transactions on Network and Service Management

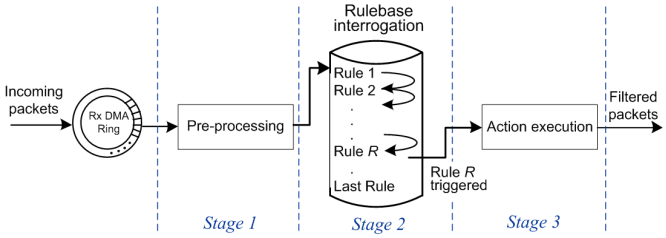

Figure 4 – Processing sequence of packets within a single vFirewall

The service discipline of packets is FCFS (First Come First Served). At Stage 2, the mean processing rate μ_2 depends on the service time to interrogate a rule and how many rules need to be interrogated before a match occurs, i.e., a rule is triggered. On average, $1/\mu_2$ can be expressed as follows

$$1/\mu_2 = L \cdot T_R,$$

where L is the average number of rules to be interrogated for an incoming traffic, and T_R is the average interrogation time per rule.

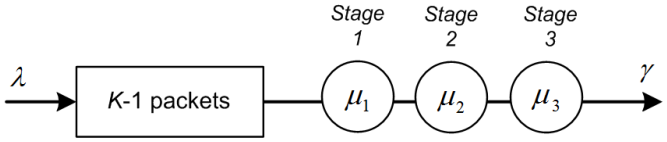

Figure 5 – A model to capture vFirewall processing of a packet

In our analysis, we assume the packet arrivals are Poisson with fixed packet sizes, and the service times are all exponentially distributed. For some other traffic types, the size of network packets are not fixed, and arrival rates do not always follow a Poisson process, but in many cases are classified to be bursty [13]–[15]. Also, the service times are not necessarily exponential. However, for specific types of network traffic, assuming Poisson arrivals can be sufficient [16]. Moreover, it was demonstrated in [17] that analytical solutions based on these assumptions do indeed offer adequate approximation and close results to real experimental results and measurements. An analytical solution becomes infeasible to solve, and in fact, intractable when assuming network packets with variable sizes and incoming rates that follow a non-Poisson process, or with general service times. Moreover, the assumptions, we undertake in this article, have been widely adopted in the literature, and do, in fact, provide acceptable approximation of real-world systems as reported in [5], [6], [18], [20], [21]. More importantly, we show in Section 4.2 that our analytical results are valid and in good agreement with those measurements taken from real-world experiments conducted in the popular AWS public cloud environment. It is worth noting that there is also some work reported in the literature, as in [19], where the authors presented a performance model and analysis of a cloud datacenter using a Poisson arrival but with a generally distributed service times. As demonstrated in [19], such models do not provide closed-form solutions and formulas that can be easily used by a cloud controller node in a cloud environment, and can in fact be computationally expensive to implement. The authors in [19] have used a software package Maple 13 to solve numerically the balance equations, as closed-form solution was not attainable.

Our analytical model is built on the principles of the embedded Markov chain with a finite state space. The model captures the behavior of the vFirewall processing with a state space $S = \{(k,n), 0 \le k \le K, 0 \le n \le 3\}$, where k denotes the number of packets in the entire system, and n denotes the service stage number being performed by the CPU. The queuing system has a buffer size of K-1. In other words, state (0,0) represents the special case when the system is empty or idle, i.e. the state of system idleness. States (k,n) represent the states where the CPU is busy executing service of stage n with k packets in the system. The state rate transition diagram is shown in Figure 6.

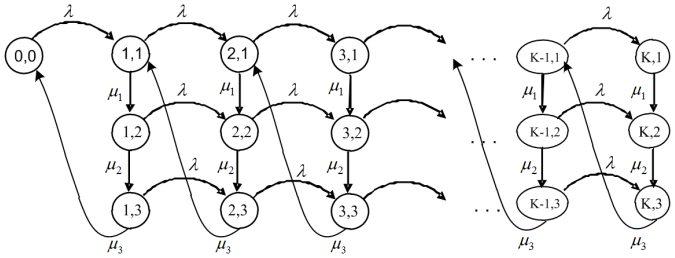

Figure 6 – State transition rate diagram with three stages of service

We start first by expressing the steady-state balance equations for each state (k,n). If we have $p_{k,n}$ denote the steady-state probabilities at state (k,n).

At state (0,0):

$$0 = -\lambda p_{0,0} + \mu_3 p_{1,3} \quad (1)$$

At state (1,3):

$$0 = -(\lambda + \mu_3) p_{1,3} + \mu_2 p_{1,2} \quad (2)$$

At state (1,2):

$$0 = -(\lambda + \mu_2) p_{1,2} + \mu_1 p_{1,1} \quad (3)$$

At state (1,1):

$$0 = -(\lambda + \mu_1) p_{1,1} + \lambda p_{0,0} + \mu_3 p_{2,3} \quad (4)$$

At state (k,3):

$$0 = -(\lambda + \mu_3) p_{k,3} + \lambda p_{k-1,3} + \mu_2 p_{k,2} (2 \le k \le K-1) \quad (5)$$

At state (k,2):

$$0 = -(\lambda + \mu_2) p_{k,2} + \lambda p_{k-1,2} + \mu_1 p_{k,1} (2 \le k \le K-1) \quad (6)$$

At state (k,1):

$$0 = -(\lambda + \mu_1) p_{k,1} + \lambda p_{k-1,1} + \mu_3 p_{k+1,3} (2 \le k \le K-1) \quad (7)$$

At state (K,3):

$$0 = -\mu_3 P_{K,3} + \lambda p_{K-1,3} + \mu_2 p_{K,2} \quad (8)$$

At state (K,2):

$$0 = -\mu_2 P_{K,2} + \lambda p_{K-1,2} + \mu_1 p_{K,1} \quad (9)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2640297, IEEE Transactions on Network and Service Management

At state (*K*,1):

$$0 = -\mu_1 p_{K,1} + \lambda p_{K-1,1} \tag{10}$$

Therefore $p_{k,n}$ can be recursively written in terms of $p_{0,0}$ as follows:

From Equation (1)

$$p_{1,3} = \frac{\lambda}{\mu_3} p_{0,0} \tag{11}$$

From Equation (2)

$$p_{1,2} = \left(\frac{\lambda + \mu_3}{\mu_2}\right) p_{1,3} \tag{12}$$

From Equation (3)

$$p_{1,1} = \left(\frac{\lambda + \mu_2}{\mu_1}\right) p_{1,2} \tag{13}$$

From Equation (4)

$$p_{2,3} = \left(\frac{\lambda + \mu_1}{\mu_3}\right) p_{1,1} - \left(\frac{\lambda}{\mu_3}\right) p_{0,0} \tag{14}$$

From Equation (5)

$$p_{k,2} = \left(\frac{\lambda + \mu_3}{\mu_2}\right) p_{k,3} - \left(\frac{\lambda}{\mu_2}\right) p_{k-1,3} (2 \leq k \leq K-1) \tag{15}$$

From Equation (6)

$$p_{k,1} = \left(\frac{\lambda + \mu_2}{\mu_1}\right) p_{k,2} - \left(\frac{\lambda}{\mu_1}\right) p_{k-1,2} (2 \leq k \leq K-1) \tag{16}$$

From Equation (7)

$$p_{k+1,3} = \left(\frac{\lambda + \mu_1}{\mu_3}\right) p_{k,1} - \left(\frac{\lambda}{\mu_3}\right) p_{k-1,1} (2 \leq k \leq K-1) \tag{17}$$

From Equation (8)

$$p_{K,2} = \left(\frac{\mu_3}{\mu_2}\right) p_{K,3} - \left(\frac{\lambda}{\mu_2}\right) p_{K-1,3} \tag{18}$$

From Equation (9)

$$p_{K,1} = \left(\frac{\mu_2}{\mu_1}\right) p_{K,2} - \left(\frac{\lambda}{\mu_1}\right) p_{K-1,2} \tag{19}$$

And from Equation (10)

$$p_{K,1} = \left(\frac{\lambda}{\mu_1}\right) p_{K-1,1}. \tag{20}$$

Please note that $p_{K,1}$ can be derived from either Equation (19) or (20). Both of these equations are numerically equivalent. Now, $p_{0,0}$ can be obtained using the normalization condition in the following form:

$$p_0 = p_{0,0} = \frac{1}{1 + \sum_{k=1}^{K} \sum_{n=1}^{3} \frac{p_{k,n}}{p_{0,0}}}. \tag{21}$$

All state probabilities $\{p_{k,n}; 1 \leq k \leq K, 1 \leq n \leq 3\}$ can be computed recursively using Equations (11-19), as shown in Algorithm 1, which can be converted easily to a MATLAB or another similar package script. As shown, the algorithm first computes the loop invariants (C_1 to C_8) in Line 05, and then uses the Equations (11-19) to determine all state probabilities.

Algorithm 1 Computing steady-state probabilities

1: **Input:** $\lambda, \mu_1, \mu_2, \mu_3, K$
2: **Output**: p_0, Matrix $P[1..K, 1..3]$
3: $p_0 = 1$
4: $P[i, j] = 0$ for i=1 to K and for j=1 to 3
5: $C_1 = \lambda/\mu_3; C_2 = (\lambda + \mu_3)/\mu_2; C_3 = (\lambda + \mu_2)/\mu_1; C_4 = (\lambda + \mu_1)/\mu_3; C_5 = \lambda/\mu_2; C_6 = \lambda/\mu_1; C_7 = \mu_3/\mu_2; C_8 = \mu_2/\mu_1$
6: $P[1, 3] = C_1$
7: $P[1, 2] = C_2 \times P[1, 3]$
8: $P[1, 1] = C_3 \times P[1, 2]$
9: $P[2, 3] = C_4 \times P[1, 1] - C_1$
10: **for** i=2 to $K - 1$ **do**
11: $P[i, 2] = C_2 \times P[i, 3] - C_5 \times P[i - 1, 3]$
12: $P[i, 1] = C_3 \times P[i, 2] - C_6 \times P[i - 1, 3]$
13: $P[i + 1, 3] = C_4 \times P[i, 1] - C_1 \times P[i - 1, 1]$
14: **end for**
15: $P[K, 2] = C_7 \times P[K, 3] - C_5 \times P[K - 1, 3]$
16: $P[K, 1] = C_8 \times P[K, 2] - C_6 \times P[K - 1, 2]$
17: $p_0 = 1/(1 + sum(\text{P}))$
18: $P = p_0 \times P$
19: return p_0 and P

Now, we show how to derive formulas for important performance metrics of the system. First, the metric for the mean system throughput γ is fundamentally the departure rate, i.e., the rate at which packets leave Stage 3, that is

$$\gamma = \mu \sum_{k=1}^{K} p_{k,3}. \tag{22}$$

The mean system throughput γ can equivalently be expressed as

$$\gamma = (1 - p_0)/\bar{X},$$

where p_0 is given by Equations (21), and $\bar{X}$ is the mean service time. $\bar{X}$ is actually the sum of the mean service time for the three stages, and hence, $\bar{X}$ can be written as

$$\bar{X} = \sum_{n=1}^{3} 1/\mu_n. \tag{23}$$

The departure rate γ can also be expressed as the effective arrival rate λ' which is $\lambda(1 - P_{loss})$. Therefore,

$$\gamma = (1 - p_0)/\bar{X} = \lambda(1 - P_{loss}), \tag{24}$$

where P_{loss} is the loss (or blocking) probability. P_{loss} can be expressed from (24) as

$$P_{loss} = p_K = 1 - \frac{1 - p_0}{\rho} = \frac{p_0 + \rho - 1}{\rho}, \tag{25}$$

where $\rho = \lambda\bar{X}$ is referred to as the offered load, and also known as the traffic intensity. We can also express P_{loss} as the probability of being in states (*K*,1), (*K*,2) or (*K*,3), that is

$$P_{loss} = \sum_{n=1}^{3} p_{K,n}. \tag{26}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2640297, IEEE Transactions on Network and Service Management

Both of Equations (25) and (26) are equivalent.

The mean number of packets that can be found in the entire system can be written as

$$E[K] = \sum_{k=1}^{K} \sum_{n=1}^{3} k p_{k,n}. \quad (27)$$

The mean number of packets in the queue can be expressed as

$$E[K_q] = \sum_{k=1}^{K} \sum_{n=1}^{3} (k-1) p_{k,n} = E[K] - (1 - p_0). \quad (28)$$

Note that the term $(1 - p_0)$ is the mean number of packets in service.

And finally, the mean time spent in a single vFW T_{vFW} can be written, using Little's formula, as

$$T_{vFW} = \frac{E[K]}{\gamma} = \frac{1}{\gamma} \sum_{k=1}^{K} \sum_{n=1}^{3} k p_{k,n}. \quad (29)$$

We can now estimate the overall response time $T_{CloudFW}$ for a cloud-based firewall system considering multiple *M* vFirewalls with a load balancer (LB) as shown in Figure 2. The LB system can be modeled as a simple finite queuing system *M/M/1/K* with a service rate of μ_{LB}. Basically, $T_{CloudFW}$ is composed of the service time incurred at the LB and the vFW, that is:

$$T_{CloudFW} = T_{vFW} + T_{LB}, \quad (30)$$

where T_{LB} formula is given in [22], and T_{vFW} is given in Equation (29) but with substituting λ with λ/M.

IV. Results and Discussion

In this section, we validate our analytical model and show how it can be used to provide elasticity and efficient design for cloud-based network firewalls. Validation is done using simulation, as well as real experiments conducted in an AWS cloud environment. The section also provides a practical example to show how elasticity of cloud firewalls can be accomplished according to the mean incoming workload and the mean depth of triggered rules of a firewall rulebase.

A. Validation through Simulation

To validate our analytical model, we report and compare numerical results obtained from analysis and simulation. The analytical curves were obtained by MATLAB implementation of the equations derived from the analytical models. The simulation results were obtained using a discrete-event simulation written in C. Details on how to develop a DES simulator in C can be found in [23]. There are a number of publically and commercially available network simulation tools. Some of these simulators are designed specifically for cloud environments (e.g., CloudSim, iCanCloud, EMUSIM, MDCSim), and some are generic in natures (e.g., OPNET, NS, OMNeT, J-Sim, JMT). All of these available simulators did not have the capabilities to capture accurately the internal behavior and dynamics of the firewall-particularly, the processing of packets in three stages of services in a mutually exclusive manner and with the middle stage interrogating rules sequentially. Our simulation code was verified carefully and checked to give correct output for pseudorandom number generator, and also to give correct results for known queueing cases as that for a service of single stage (i.e. M/M/1/K). Moreover, the simulation followed carefully the principles and recommendations outlined in [23]. To validate our analytical model, we considered the same assumptions for our analysis model as that for simulation. We then compared the analytical results with the simulation results. As depicted in Figures 7 and 8, red circles represent the simulation results, and the solid blue curves represent the analysis results. Clearly, both figures show that the results obtained from both analysis and simulation are in good agreement-thus, implying the correctness of the analytical model. For our numerical examples, we have chosen K=300 packets, $1/\mu_1 = 5.3\mu s, 1/\mu_3 = 200\mu s, T_R = 0.1\mu s$ and $1/\mu_{LB} = 100\mu s$. These values for the processing time are realistic and roughly twice as much as the experimental measurements reported in [17] for a high-end quad core physical machine. In the cloud, we approximately doubled the processing time to account for the overhead introduced by the virtualization of the underlying network and compute infrastructure.

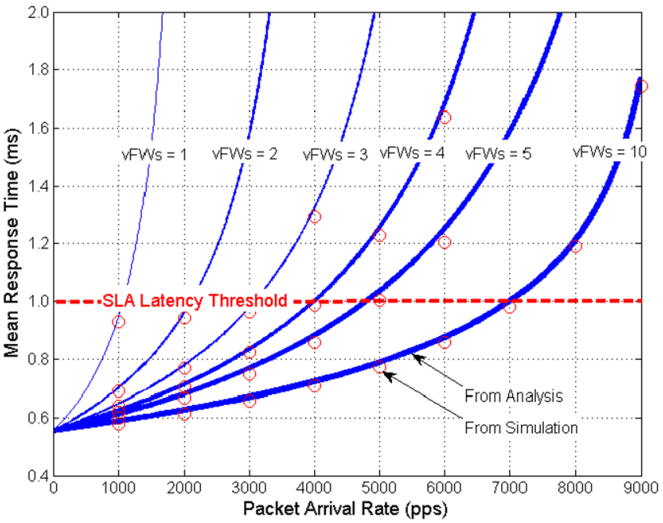

Figure 7 – Impact of vFWs and incoming rate on response time

Figure 7 illustrates the impact of incoming traffic λ on the $T_{CloudFW}$ which is the mean response time incurred at both the load balancer and vFWs. Incoming traffic λ is expressed in packets per seconds (pps). In the figure, we fixed the average number of interrogated rules L to 2500. The figure illustrates how the response time is affected by both incoming traffic rate as well as the number of allocated vFWs. The figure shows the response time curves increase under high traffic load, and also are highly impacted by the number of vFWs used. More importantly, the figure illustrates how to determine the minimal number of vFWs required to meet a given mean response time so that the end-to-end SLA latency of a hosted application

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2640297, IEEE Transactions on Network and Service Management

or service can be achieved. For Example, the SLA latency threshold for a firewall service, shown in the red horizontal dashed line, is 1.0 ms and given an incoming arrival rate of 4000 pps, the figure shows that a total of 5 vFWs is required to keep the mean SLA firewall latency under 1.0 ms. Another important observation to be made is that the number of vFWs does not need to change as the workload changes within a certain range. For example, a single vFW would suffice for a workload ranging from 0 to 1000 pps. Also, two vFWs would suffice for a workload ranging from 1000 to 2000 pps, and three vFWs would suffice for a workload ranging from 2000 to 3000 pps.

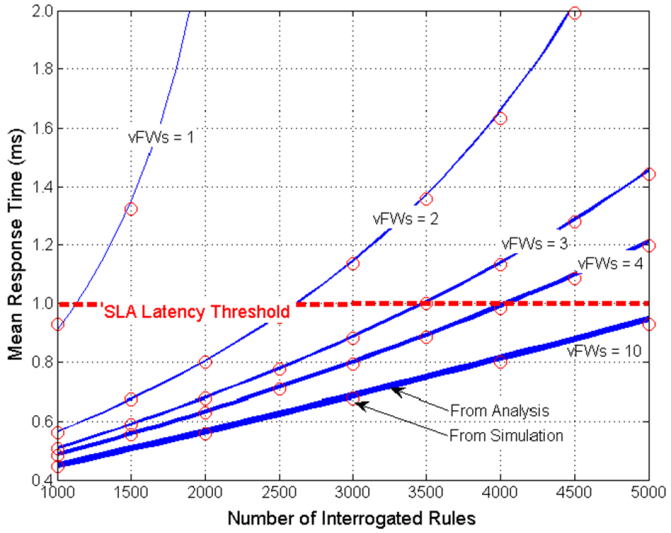

Figure 8 – Impact of vFWs and the number of interrogated rules on response time

The impact of L, which is the average number of rules to be interrogated for an incoming traffic, on the mean response time is shown in Figure 8. For all performance curves shown in the figure, we fixed the incoming traffic rate to 2000 pps. The shows how the response time increases as the average number of interrogated rules increases. This is expected since the deeper the interrogation of the firewall rulebase, the more processing and interrogation the vFW has to undertake. The figure also shows that as the interrogation and processing gets distributed among multiple vFWs, the response time is reduced significantly. The figure also shows how we can determine the minimal number of vFWs required to satisfy a given SLA latency. At a rate of 2000 pps, only three vFWs are needed to satisfy an SLA latency of 1.0 ms. It can be seen from the figure that there is no change required in the number of vFWs if the average number of interrogated rules lies between 2500 to 3500. Similarly, we can see that only four vFWs are needed if the average number of interrogated rules lies between 3500 to 4000.

B. Experimental Validation

To validate further our analytical model, we compare our analytical results against measurements collected from an experimental testbed deployed in an AWS cloud environment shown in Figure 9. Our experiments are comprised of different sizes of EC2 VMs with an ELB (Elastic Load Balancer) that auto-scales its processing capacity according to incoming workload and traffic [29]. We selected EC2 VMs that are optimized for networking and processing power capacity that would meet our requirements. For that, we specifically used two large size EC2 Linux instances for generating and receiving network traffic using the open-source D-ITG 2.8.1 traffic generator [31], [32]. D-ITG has two major components. ITGSend for sending traffic, and ITGRecv for receiving it. For our vFWs, we used small size EC2 Linux instances with Netfilter firewall installed. Small size Linux firewall was sufficient to process thousands of rules with acceptable delay. The ELB was configured to distribute equally (in a round robin manner) incoming network packets to all running vFW instances. The aforementioned cloud instances were all hosted in the same AWS VPC [33] to provide logical isolation from other tenants VMs. For all of these instances, we used Ubuntu Linux Server 13.10 as the base operating system.

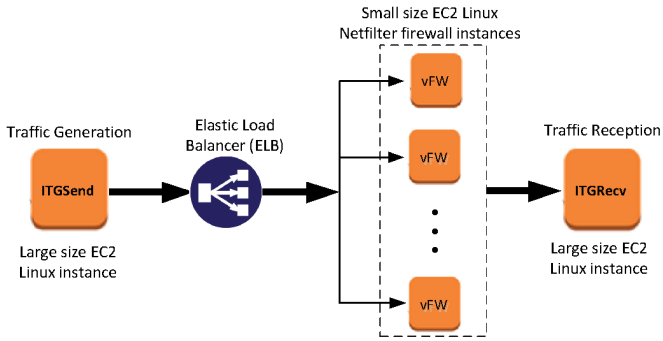

Figure 9 – Experimental setup using Amazon AWS Cloud

As shown in Figure 9, D-ITG traffic generator had to be configured to have ITGSend EC2 instance generate a single unidirectional flow to ITGRecv instance [31] , [32] where packets are received, and statistics are collected. NTP (Network Timing Protocol) service was used for time synchronization between the two instances required for precise computation of statistics. This notably ensures accurate measurements for the one-way packet delay calculation. For our measurements, the CPU utilization was measured by the *sar* Linux utility, whereas the throughput and one-way response time were measured by D-ITG. It is worth noting that we configured ITG-Send to send the smallest packet size of 64 bytes for UDP flows in order to generate high traffic rates. More importantly, we used the popular Puppet automation software [34] to configure, coordinate, and execute the various commands and tasks for D-ITG, NTP, and *sar*. Puppet ran on a different small size Linux instance in our testbed setup.

We followed the guidelines in the experiment described in [35] to set up a Linux *Netfilter* firewall. We constructed a ruleset of 5,000 rules using *iptables* commands. We also added specific rules for D-ITG traffic being generated from ITGSend instance. For this type of D-ITG traffic, we set up *Netfilter* to accept, log and pass D-ITG traffic so that it can be received and recorded by ITGRecv. Other traffic types would be dropped. In the dummy rules, we added matching conditions for source

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2640297, IEEE Transactions on Network and Service Management

MAC addresses to add noticeable processing overhead for each rule. According to [30], the matching of MAC address conditional was found to be computationally more expensive when compared to other header fields.

To measure the average service times for $1/\mu_1, 1/\mu_3$ and T_R, we instrumented the Linux code at different positions to measure the difference in time using `rdtscl` macro which uses the machine instruction `rdtsc` (read time stamp counter) to return the number of CPU cycles accumulated since system bootup. For example, for measuring the mean kernels pre-processing time $1/\mu_1$ of Stage 1, we instrumented the Linux code with `rdtsc` at the start of function *tg3_interrupt()* found in *tg3.c* file (the start of receiving a packet in the device driver) until the entry point of delivery to *Netfilter* processing (specifically, in *ip_local deliver_finish()* found in *ip_input.c* file). For T_R, we instrumented the Linux code with `rdtsc` at the start and finish points of *Netfilter* processing, specifically in the Linux kernel function *ip_local_deliver_finish()* found in *ip_input.c* file as discussed in [31]. As for $1/\mu_3$, we measured the difference in time between *ip_local_deliver_finish()* and the point before sending the packet to the NIC in *tg3_tx()* in *tg3.c* file. Since this instrumentation requires the modification of Linux kernel and *tg3* driver, we had to create a customized Amazon Machine Image (AMI) for Linux with instrumentation embedded. This required to first create a VMware Linux image with these instrumentations embedded into the Linux kernel, and then export it to Amazons AMI image [36] in order to perform the real measurements using Amazon small size EC2 Linux instances.

To perform the measurements, we had ITGSend instance send traffic directly (i.e., without passing through the ELB) to the vFW Linux instance which will forward it further to ITGRecv instance. We had ITGSend send a low constant UDP traffic rate λ of 1000 pps for a duration of one full second, targeting rule number 5000, with a *log* and *pass* actions. We took 1000 reads of the difference between the start and finish timestamps. All of these differences were then added up in memory into a single variable, and then the mean value was obtained by dividing this total value by 1000. This gives us the mean values for $1/\mu_1$ and $1/\mu_3$ as $12\mu s$ and $342\mu s$, respectively. We note that $1/\mu_3$ is relatively large when compared with $1/\mu_1$. This is due to the extra overhead involved in logging and then passing the packets. The mean value $1/\mu_2$ of interrogating 5000 rules was 700, or 0.14 μs per rule, i.e., T_R = 0.14 μs. Finally, we set the default buffer size *K* to 512 packets, which is the same value defined for Rx DMA Ring in header file definition */net/drivers/tg3.h*

For measuring T_{LB}, which is the mean delay attributed to the ELB, we first had ITGSend send 1000 UDP packets at a low constant rate directly (without passing through the ELB) to ITGRecv. Second, we repeated the same but with ELB placed in the middle between the ITGSend instance and ITGRecv instance. We then computed the differences between the two averages to determine the true delay T_{LB} attributed to ELB. T_{LB} was approximately 121 μs. Lastly, all generated traffic flows by ITGSend were designed to target rule number 3500, i.e., *L* = 3500.

Table I provides comparative results from real-world experiments and analysis, for the three key performance metrics of response time, throughput, and CPU utilization. For the experimental results, we record the minimum, maximum, and average values of five runs, with each run having traffic flows being generated for a duration of 15 minutes. Adding more experimental runs than five would yield little difference, and we found that five runs are adequate and yield acceptable results. All of these runs were carried out approximately at midnight GST time in an AWS zone located in Ireland. We believe this time might corresponds to the lightest workload in the Amazon cluster. We opted to run the traffic flows for 15 minutes to offset the variability and fluctuation that occur within the cloud environment as a result of network and workload activities induced by co-tenant machines or cloud-hosted services and apps.

In our experiment, we measured the performance with different incoming traffic rates and with different numbers of vFWs. A few observations can be made from Table 1. First, the average experimental measurements, in general, are in line with those of the analysis results-which further validates our analytical model. Second, the response times obtained from experimental measurement, for the most part, are slightly bigger (by approximately 1.6 ms) than those in the analysis results. This is due to the fact that the average response time in the analysis does not consider the additional processing delays encountered by ITGSend or ITGRecv, since both of these programs run as applications in the user space (not in the kernel). Third, the experimental results for throughput are slightly smaller than their analysis counterparts. This can be attributed to the processing capacity of the various cloud instances, and the ability of ITGSend to accurately send the specified traffic rate. Finally, in general, the experimental measurements have large standard deviations examining the difference between the min and max values. The reason for such large deviations can be attributed to the considerable fluctuation and variability encountered in the cloud environment, including: the overheads introduced by the underlying virtualization technology; the activities and workload generated by co-tenant instances and cloud-hosted applications and services co-existing on the same cloud infrastructure. If complete isolation in a VPC environment with the cloud-based firewall is required to obtain fully predictable network delay and performance, the more expensive option of running the testbed instances on single-tenant dedicated hardware within AWS can be used.

C. Achieving Proper Elasticity

In this section, we give a numerical example to illustrate how to achieve elasticity for cloud-based firewalls. Figure 10 illustrates how the analytical model can be used to determine the minimal number of vFWs required to satisfy a given SLA latency, taking into account the fluctuation of incoming workload and the depth of triggered rules. For our example, we selected to perform auto-scaling every minute, i.e., the adjustment period to be 1 minute. At the end of this period, the mean workload λ, and the mean depth of triggered firewall rules L are calculated and used as input to Algorithm 1 and Equation (30) in order to determine the minimal number of vFWs required to satisfy an SLA latency of 1 ms.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2640297, IEEE Transactions on Network and Service Management

TABLE I – Comparative results from analysis and experiments

	Response Time (ms)				Throughput (pps)				CPU Utilization (%)			
	Analysis	Experiment			Analysis	Experiment			Analysis	Experiment		
	Avg	Avg	Min	Max	Avg	Avg	Min	Max	Avg	Avg	Min	Max
1 vFW at a rate of 500 pps	1.62	2.12	1.01	6.22	500	495	461	500	42	44	38	47
2 vFWs at a rate of 1200 pps	1.85	3.44	2.35	6.81	1200	1182	1110	1200	51	53	44	59
3 vFWs at a rate of 2500 pps	3.02	4.58	2.89	9.21	2500	2479	2463	2500	70	74	64	81
4 vFWs at a rate of 2500 pps	1.96	3.61	2.11	677	2500	2482	2468	2500	53	58	49	68
4 vFWs at a rate of 4000 pps	5.64	7.12	4.27	10.89	4000	3901	3821	4000	84	86	74	91
6 vFWs at a rate of 6000 pps	5.85	7.42	4.11	11.02	6000	5867	5701	6000	84	87	71	94
10 vFWs at a rate of 10,000 pps	4.83	6.36	3.19	12.92	10,000	9879	9711	10,000	84	89	77	98

Two important observations can be made from Figure 10. First, the example shows that the latency specified in the SLA is always satisfied with a mean response time less than the required 1 ms. In some cases, the response time is close to 1 ms, and in some other cases (at adjustment periods of 0, 2, 6, and 8 minutes), the mean response time is between 0.5 and 0.8 ms. This all depends on the processing capacity of each vFW in relationship to the processing need presented in terms of workload and rule interrogation overhead. Second, the figure shows that required VMs to satisfy a response latency of 1 ms is clearly impacted by both incoming workload as well as depth of triggered rules. For example, at 3 minute adjustment period, the required number of vFWs increased from 1 to 3 to meet the increase of the processing overhead attributed to the increase of the depth of triggered rules from 2000 to 4000. At this period, there was no change in the mean workload from the previous adjustment period. Also, at 4 minute adjustment period, the required number of vFWs increased from 3 vFWs to 10 in response to the noticeable sharp increase of workload from 1000 pps to 4000 pps. Similarly, there was no change at this adjustment period for the mean depth of triggered firewall rules. Conversely, at adjustment periods of 2 and 6 minutes, the number of required vFWs is decreased when the average workload or depth of triggered rules decreases. At adjustment period of 2 minutes, a decrease in the workload results in reducing the required number of vFWs from 3 to 1. At adjustment period of 6 minutes, a decrease in the depth of triggered rules results in reducing the required number of vFWs from 5 to 3. It is also observed, when both of the workload and the depth increase, as is the case at adjustment period of 7 minutes, the required number of vFWs sharply increases.

From the figure, it can also be noted that the largest required number of vFWs needed to satisfy an SLA latency of 1 ms happens at adjustment period of 4 minutes, with a relatively high mean workload of 4000 pps and mean rule depth of 4000. Also, the least required minimal needed number of vFWs occurs at adjustment period of 2 minutes, with a relatively light mean workload of 1000 and a mean rule depth of 2000. As the figure shows, an increase or decrease in either workload or rule depth does impact the required number of needed vFWs, as depicted for other adjustment periods.

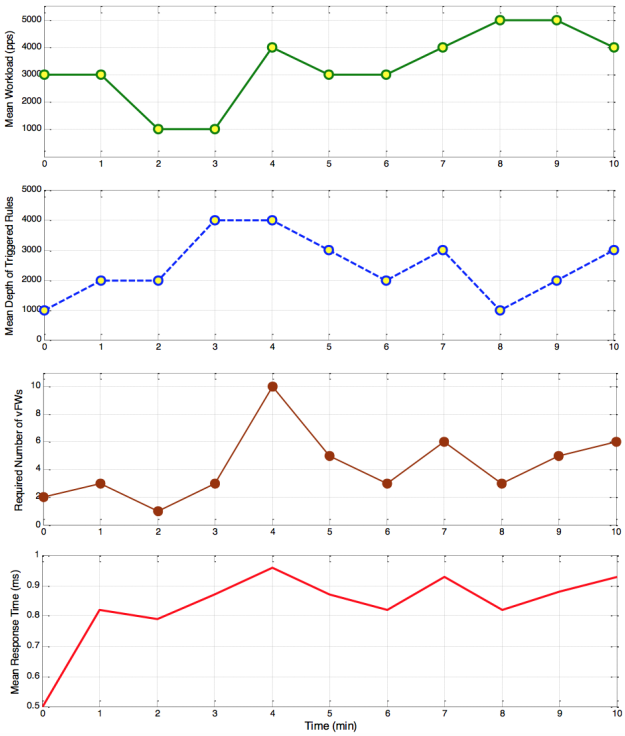

Figure 10 – Impact of incoming workload and depth of interrogated rules on required vFWs

It is worth noting that the adjustment period can be a key design factor for achieving proper elasticity for cloud-based network firewalls. In reality, the elasticity or adjustment period for determining the required vFWs has a minimum bound which is governed by a VM provisioning time. That is, this period has to be at least larger than the provisioning time for a vFW instance. Provisioning (or spin-up time) includes the time to allocate, configure, launch, reboot, and connect the vFW with the LB. According to measurements performed by [24]-[25], the provisioning time of one VM takes in the order of 30 seconds to 100 seconds. For our numerical example, we chose a realistic adjustment period of 1 minute, which is in line with these reported measurements, and also in line with

our experimental findings when using AWS cloud–whereby the provisioning time for a Linux-based small instance vFW takes on average less than 50 seconds.

The adjustment period can be chosen to be larger than this minimal period in order to reduce the amount of elasticity overhead attributed to computing the average workload and that of the depth of triggered rules. However, selecting large adjustment period can result in SLA latency violation or improper provisioning (i.e. over or under provisioning) as workload or rule depth may have sharp increase or decrease during large adjustment periods. Hence, we argue that the adjustment period should be slightly larger (i.e., a few or ten seconds more) than the minimum provisioning time. Even though, SLA violation can still happen when the system is subjected to an abrupt and sharp increase of incoming workload or depth for rule interrogation. If these abrupt conditions are very frequent, a possible solution is to slightly overprovision vFWs (by one or two instances) beyond the required minimal vFWs.

The computation of the average workload as well as the provisioning or de-provisioning of vFWs can be implemented as a specific agent within the LB or at the edge router of the VPC. However, the computation of the average depth of triggered rules has to be computed locally at each running vFW through agents and this individual average must be relayed back through the deployed agents to the Elasticity Orchestrator where the average of the received averages is computed. Since the number of firewall rules and the number of received packets by each firewall are the same for all individual vFWs (as workload gets evenly distributed), the overall average depth L for the triggered rules of the individual averages L_j of N vFWs can be calculated as follows

$$L = \frac{\sum_{j=1}^{N} L_j}{N} \tag{31}$$

Both of the average workload and the average of rule depth are moving averages that are calculated over the adjustment period. The average workload can be computed in multiple ways. In [26] - [28], authors show how to estimate the mean workload λ using moving average techniques.

V. Conclusion

In this article, we showed how elasticity can best be achieved for cloud-based firewalls. To do so, we developed an analytical model useful for the efficient design of cloud-based firewalls, which are essential in VPC services. Given the offered means for workload, depth of targeted rules, and the processing capacity of each firewall and load balancer instances, the model can estimate accurately the minimal number of VMs needed to meet a specific SLA criterion such as response time, delay or throughput. We validated our analytical model by comparing analytical results with results obtained through discrete-event simulation and real-world measurements taken from an experimental testbed deployed in Amazon AWS cloud environment. Although, an obvious degree of fluctuation was exhibited in the experimental measurements, the overall mean recorded measurements were in good agreement with results obtained from analysis. The fluctuation in measurements was attributed to the overhead from virtualizing and sharing of the various cloud physical infrastructure elements of compute, storage, and network resources, and also attributed to the workload generated from the various activities of other co-located cloud-hosted services and applications. As a final remark, we believe that using our derived analytical formulas, algorithm, and guidelines presented in this artcile, an elastic cloud-based firewall system can be designed, implemented, and deployed efficiently in any cloud environment with VPC configuration.

Acknowledgments

The authors thank the anonymous reviewers for their valuable comments, which helped us to considerably improve the content, quality, and presentation of this article. The experimental work in this paper was supported by a generous research award under the Amazon Web Services Education Program [37].

References

[1] F. Al-Haidari, M. Sqalli and K. Salah, "Enhanced EDoS-shield for mitigating EDoS attacks originating from spoofed IP addresses", *In the proceedings of the 11th IEEE Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2012)*, 2012

[2] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," *in the proceedings of the 16th ACM conference on Computer and communications Security*, 2009

[3] E. Al Awadhi, K. Salah and T. Martin, T., "Assessing the security of the cloud environment," In the proceedings of the 7th IEEE GCC Conference and Exhibition (GCC 2013), November 2013, pp. 251-256.

[4] Salah, K., "Analytical Model to Achieve Elasticity for Cloud-based Firewalls," Proceedings of the 40th IEEE Conference on Local Computer Networks, (IEEE LCN 2015), Clearwater Beach, Florida, October 26-29, 2015.

[5] M. Liu, W. Dou, S. Yu and Z. Zhang, "A clusterized firewall framework for cloud computing, " In the proceedings of the IEEE International Conference on Communications (ICC 2014), pp. 3788-3793/

[6] S. Yu, R. Doss, W. Zhou, and S. Guo, "A general cloud firewall framework with dynamic resource allocation," in the proceedings of the IEEE International Conference on Communications (ICC 2013), pp. 1941-1945.

[7] A. Khakpour and A. Liu, "First step toward cloud-based firewalling," in the proceedings of the 31st Symposium on Reliable Distributed Systems (SRDS 2012), pp. 41-50.

[8] S. Gold, "The future of the firewall." Journal of Network Security, 2011, No. 2, Elsevier, pp. 13-15.

[9] K. Salah, J. Alcaraz Calero, S. Zeadally, S. Almulla, and M. Alzaabi, "Using Cloud Computing to Implement a Security Overlay Network," IEEE Security and Privacy, Vol. 11, No. 1, January 2013, pp. 44-53.

[10] F. Guenane, H. Boujezza, H., M. Nogueira, and G. Pujolle, "An architecture to manage performance and reliability on hybrid cloud-based firewalling," Proceedings of the IEEE Network Operations and Management Symposium (NOMS), May 2014, pp. 1-5.

[11] T. Booth and K. Andersson. "Network Security of Internet Services: Eliminate DDoS Reflection Amplification Attacks." Journal of Internet Services and Information Security (JISIS), Vol. 5, No. 3, 2015, pp. 58-79.

[12] F. Guenane, B. Jaafar, M. Nogucira, and G. Pujolle, "Autonomous architecture for managing firewalling cloud-based service," Proceedings of the 5th International Conference on the Network of the Future (NOF14), Paris, France, pp. 76-80.

[13] W. Leland, M. Taqqu, W. Willinger, D. Wilson, "On the self-similar nature of Ethernet traffic", IEEE/ACM Transaction on Networking, vol. 2, no. 1, February 1994, pp. 1-15

[14] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," IEEE/ACM Transactions on Networking, vol. 3, no. 3, June 1995, pp. 226-244

[15] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson, "Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level," In the proceedings of ACM SIGCOMM 1995, Cambridge, Massachusetts, August 1995, pp. 100-113.

[16] M. Karam and F. Tobagi, "Analysis of delay and delay jitter of voice traffic in the Internet," Computer Networks Magazine, vol. 40, no. 6, December 2002, pp. 711-726.

[17] K. Salah, K. Elbadawi, and R. Boutaba, "Performance modeling and analysis of network firewalls," IEEE Transactions on Network and Service Management, 2012, vol. 9, no. 1, pp. 12-21.

[18] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, A. Tantawi, "An Analytical Model for Multi-tier Internet Services and its Applications," Proceedings of the 2005 ACM SIGMETRICS International Conference, Vol. 33, Alberta, Canada, pp. 291-302.

[19] H. Khazaei, J. Misic, and V. Misic, "Performance Analysis of Cloud Computing Centers Using M/G/m/m+r Queueing Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 23, No. 5, May 2012.

[20] S. Kikuchi and Y. Matsumoto, "Performance Modeling of Concurrent Live Migration Operations in Cloud Computing Systems using PRISM Probabilistic Model Checker," Proceedings of the 4th IEEE International Conference on Cloud Computing, 2011, Melbourne, Australia, pp. 49-56.

[21] M. Firdhous, O. Ghazali, and S. Hassan, "Modeling of Cloud System using Erlang Formulas," Proceedings of the 2011 7th Asia-Pacific Conference on Communications (APCC), Saba, Malaysia, October, 2011, pp. 411-416.

[22] L. Kleinrock, "Queueing Systems, Vol. 1: Theory", John Wiley & Sons, 1975

[23] A. Law and W. Kelton, Simulation Modeling and Analysis, McGraw-Hill, 2nd Edition, 1991.

[24] H. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. Rumble, E. Lara, M. Brudno, M. Satyanarayanan, "SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing," Proceedings of the 4th ACM European Conference on Computer Systems, EuroSys09, Nuremberg, Germany, March 2009, pp. 1-12.

[25] M, Mao and M. Humphrey, "A Performance Study on the MV Startup Time in the Cloud," Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD2012), June 2012, pp. 423-430.

[26] A. Cockcroft, "Utilization is Virtually Useless as a Metric," Proceedings of CMG 2006 Conference, December 2006.

[27] K. Salah, "Implementation and experimental evaluation of a simple packet rate estimator." AEU-International Journal of Electronics and Communications 63.11 (2009): 977-985.

[28] K. Salah and F. Haidari. "Performance evaluation and comparison of four network packet rate estimators." AEU-International Journal of Electronics and Communications 64.11 (2010): 1015-1023.

[29] Amazon Web Services, "Elastic Load Balancing," Available at https://aws.amazon.com/elasticloadbalancing/

[30] A. J. Melara, "Performance Analysis of the Linux Firewall in a Host," Master Thesis, California Polytechnic State University, June 2002.

[31] A. Botta, A. Dainotti, A. Pescap, "A tool for the generation of realistic network workload for emerging networking scenarios", Computer Networks (Elsevier), 2012, Volume 56, Issue 15, pp 3531-3547.

[32] Distributed Internet Traffic Generator, 2014. Available at http://traffic.comics.unina.it/software/ITG/

[33] Amazon Web Services, "Amazon Virtual Private Cloud Route Tables," Available at http://aws.amazon.com/documentation/vpc/

[34] Puppet Open Source, 2014. Available at http://puppetlabs.com/puppet/puppet-open-source

[35] Salah, K., Sattar, K., Sqalli, M., and Alshaer, E., "A Potential Low-Rate DoS Attack against Network Firewalls," International Journal of Security and Communication Networks, John Wiley, Vol. 4, No. 2, pp. 109-238, February 2011.

[36] Amazon Web Services, "Importing a VM into Amazon EC2 as an Instance," Available at http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ UsingVirtualMachinesinAmazonEC2.html

[37] Amazon Inc, "Amazon AWS Education Grants," 2014. Available at http://aws.amazon.com/education

[38] S. Rajagopalan, D. Williams, H. Jamjoom, A. Wareld, "Split/merge: System support for elastic execution in virtual middleboxes", *In Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013.

[39] J. Hwang, K. Ramakrishnan, T. Wood, "NetVM: high performance and flexible networking using virtualization on commodity platforms", *IEEE Transactions on Network and Service Management, 12(1)*, 2015.

[40] A. Gember, A. Krishnamurthy, S. John, R. Grandl, X. Gao, A. Anand, V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud", *Technical Report*, 2015.