Cost-effective N:1 Firewall Array via subnet-level load balancing by SDN/OpenFlow switches

Christian I. Quispe, Cesar A. Santivanez
Advanced Networks Research Lab (GIRA)
Engineering Department
Pontificia Universidad Católica del Perú (PUCP), Lima, Perú
Email: {cquispe,csantivanez}@pucp.pe

***Abstract*— Enterprise networks' firewalls are rarely set up in a shared-protection (N:1) configuration, preferring instead the more costly active-stand by (1+1) configuration. This is due to, in part, to the high cost associated with software-based per-flow load balancing at high traffic loads. In this work, we propose the use of SDN/OpenFlow switches as a low-cost hardware-based alternative for subnet-level load balancing, for use in a N:1 firewall array. Our design guarantees that the N:1 system exhibits the same performance of a 1+1 system but at a much lower cost. In particular, the design provides lossless firewall handover of subnets in response to traffic bursts. A prototype of the system has been implemented in Python over the base of the Floodlight OpenFlow Controller and tested in our SDN Test-Bed using the ISPDSL II traffic traces, evaluating the behavior during traffic peaks. Initial results show that the system successfully migrate subnet traffic without packet losses, call blockings, or TCAM exhaustion.**

Keywords*— *Application Delivery Controller (ADC), Firewall Load Balancer, Firewall Array N:1 , HandOff Process, ExoGENI TestBed.

I. INTRODUCTION

To defend against potentially damaging cyber-attacks, Enterprise Networks employ expensive firewalls typically under the "1+1" High Availability (HA) configuration. In this scheme, one firewall (active) is handling the traffic while the other (stand by) receives a copy of the traffic, keeps track of the sessions' state, and is ready to take over in case of the active firewall failure.

In the 1+1 configuration, each firewall is dimensioned to support the current and future traffic of the network, with a projection of at least five years. That is, assuming a (typical) compound annual growth rate (CAGR) of traffic 35%, the company acquires two firewalls each with capacity up to 5 times the current traffic need. This is a costly proposition, even more taking into account that the cost of firewalls typically grows faster than the traffic they support (for example, having to acquire high-end devices instead of a medium-end one).

On the other hand, the N:1 model where N active devices share a single back up device (shared protection) results on a significant reduction in capital cost (CaPex) with respect to the 1+1 model. Under the N:1 model, firewalls are acquired based on actual demand (new devices are added as needed, reducing financing costs) and the size and cost of the backup firewall is significantly reduced. Furthermore, postponing a device's adquisition will likely benefit from the natural reduction of the cost of the equipment over time (for the same processing capacity). In order to achieve these gains, an N:1 firewall array requires load balancers (two for high availability) that distributes the traffic into the different firewalls in the array.

Typical comercial load balancers (Application Delivery Controller, or ADC) for an Enterprise-level network (with several Gbps of traffic and hundreds of thousands of active sessions at a given time) are quite expensive (ten of thousands of dollars), because their balancing strategies operate at the granularity of the order of a session (quintuple: IPsrc, IPdst, Protocol, TCP/UDP PORTsrc, TCP/UDP PORTdst). Other load balancers (like the one proposed in [1]) use dedicated hardware that is expensive and inflexible.

This work proposes the use of inexpensive SDN/OpenFlow switches as subnet-level load balancer for an N:1 firewall array. The OpenFlow switches group sessions in subnets (of varying granularity) and assign each subnet's traffic to a given firewall. Since the number of subnets is significantly lower tan the number of sessions, this approach does not exceed the limit of possible entries in the switch TCAM memory. Since the lower granularity of operation results on a loss of statistical multiplexing and higher overprovisioning per firewall, a novel firewall handover technique is implemented that allows migrating the subnet traffic (in response to traffic bursts) without interrupting ongoing sessions (that cannot change firewall).

Under our approach, mission critical flows are provided dedicated (1+1) protection, while normal traffic is provisioned with shared protection (N:1). Under normal operation (traffic burst, or suspension of a computer for predictive maintenance / software upgrade), no session will be interrupted. During the unexpected failure of a device, critical traffic will not be interrupted (as in the 1 + 1 model), while normal traffic will be migrated immediately as soon as the failure is detected, resulting in a possible reset of active sessions.

This article is organized as follows: Section II describes the related previous work. Section III describes the design of our load balancer. Section IV discusses our prototype implementation in Python on top of the Floodlight OpenFlow Controller [12]. Section V presents initial results of the system

978-1-5386-8372-9/18/$31.00 ©2018 IEEE

evaluated in the high fidelity ExoGENI network emulator [10]. Finally, Section VI presents our conclusions.

II. RELATED WORKS

Scipass [2] uses an SDN application that communicates with the Bro IDS array to detect large data transfers (elephant flows). Once these flows have been identified and determined to be harmless, the controller insert bypass rules in the SDN Switch to prevent these flows from overloading the firewall. With this methodology Scipass provides security to the campus network of Indiana University (which supports a transmission rate of 100 Gbps) with non-expensive equipment (firewalls) avoiding sending high-volume elephant flows to them.

Besides [2], several works have proposed the use of the SDN for load balancing. Govindarajan et al. [3] introduce a Particle Swarm Optimization (PSO)-based load balancing techniques to distribute customer applications in a cloud scenario trying to minimize the average response time. Adriano et al. [4] compare the performance of SDN-based load balancers using three strategies: Random, Round-Robin and CPU-Usage-based. They conclude that SDN-based load balancing is an effective, viable and inexpensive solution compared to traditional ones. Kaur et al. [5] propose an SDN load balancer that uses flow statistics to guide the load balancer strategy. The load balancer implementation is through a POX controller and an OpenFlow switch. They show that that their balancer has a shorter response time than the ones based on the Round Robin strategy. The above works distribute the load coming from the requests of the clients to a set of web servers, operating at a granularity of a session, not a subnet, and therefore they do not face the challenge of avoiding migrating an active session in response to traffic bursts. The latter is critical in second-generation firewalls that keep track of the state of connections. Migration of an active session from one second-generation firewall to another will likely result on the session being blocked.

Another important challenge, especially in large Enterprise networks, is avoid exceeding the capacity limit of the TCAM memories used by the switches to store the OpenFlow rules (OpenFlow Entries) [11]. i.e., the P-3297 switch of the Pica8 brand only has 8K entries in its TCAM memory. The use of TCAMs allows the quick search of rules that correspond to the header of a packet. If the capacity of the TCAMs is exhausted and the rest of the OpenFlow entries have to be stored in conventional memory, the ability of the switch to operate at full line rate would be severely limited. A session-based SDN-load balancer may require one OpenFlow Entry per active session, severely limiting its scalability. Our proposed subnet-based load balancer avoids these problems making an efficient use of TCAM memory. The problem regarding the limit on the number of flows entries due to a small TCAM memory has been studied in [6, 7]. [8] Proposes an effective management of the switch memory in Openflow networks from a centralized logic.

III. LOAD BALANCER DESIGN

To ground the discussion, an to showcase the type of scenario where we expect our proposal to be cost-effective, we will consider our school (PUCP) campus network: Internet access rate of 2.5Gbps, approximately 10K wired and 12K wireless users, and at average 80K simultaneous active outgoing sessions (i.e., accesing Internet). However, it should be noted that our design is able to support networks and traffic intensities one or two order of magnitude higher.

There are two scenarios where the load balancer has to intervene: (i) to migrate the entire traffic from a device to another, as in response to a failure, and (ii) to migrate just a subnet traffic, in response to traffic bursts. In the first case, session interruption/restart may be unavoidable. In the second, a careful Firewall HandOver (FHO) procedure can avoid traffic interruptions. For mission-critical traffic, traffic interrumption due to a device failure is a problema to be solved, taking into consideration:

- Not all network traffic is “mission critical”. Most traffic can be reset immediately after a failure (i.e., “refresh” in the browser). For most normal traffic, the goal is to recover connectivity as soon as possible, instead of waiting hours.
- Not all failures are unforeseen. Many times a device is taken off line for software update or maintenance. In these cases, it is posible to migrate the traffic before turning off the device. In fact, firewall device failures are a rare occurrence.
- Mission critical traffic (which is easy to identify, because the company knows what it is and we assume it is a small fraction - in volume and in number of sessions - of the total traffic) can be assigned to 2 firewalls simultaneously.

Various configurations in firewall fixes are possible so as not to interrupt critical traffic:

- N active firewalls plus 1 extra firewall that serves as stand-by (backup, 1+1) for critical sessions - and to which traffic can be migrated from non-critical sessions in case some other equipment fails (leaving critical sessions unprotected, assuming at most one failure simultaneously).
- N active firewalls, 1 firewall for shared protection N: 1 for non-critical sessions, and n firewalls dedicated to protecting critical traffic in stand-by mode (backup, 1+1).

Each enterprise can decide the configuration that best suits their needs based on the ratio of critical traffic vs normal traffic, and the number of failures it wants to support. Due to the space limit, in this paper we will only focus on the handling of normal traffic during traffic bursts, as it is the most complex due to the FHO process (subnet traffic migration from one firewall to another without interrupting active sessions).

Figure 1 shows a simplified description of the proposed system, where only one pair of OpenFlow Switches (OFS) and one OpenFlow Controller (OFC) is shown, for clarity. An actual High Availability deployment will consist of two pairs of OFSes and two instances of the OFC. The core idea is to adapt an OFS (i.e., Pica8 P-3290) to fulfill the role of load balancer. The switch receives rules sent by the OFC through

the (c) interface via the OpenFlow protocol for the creation, update or elimination of Flow Entries (FE). These rules are responsible for the load balancing in the firewalls bank N:1.

Considering a network with F flows divided among M subnets, carrying on average L sessions each (in the case of the PUCP network, F = 80K, M = 300, and and L = 266), initially the total number of FEs in the OFS will be M. Later, during the FHO process up to L FE will be added (see Section III.C). Thus, our system scalability limit is that M+L is less that the size of the OFS TCAM. For an 8K-entries TCAM, under the best conditions the system could support up to 16 million active sesions.

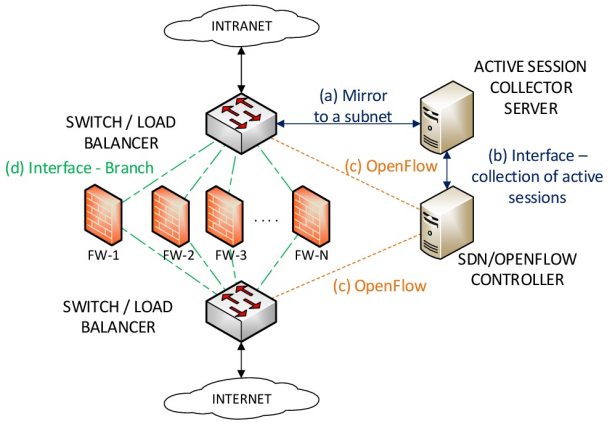

Fig. 1. Design of the load balancer and its elements.

The OFC inserts (interface c) M load balancing rules (one FE each) in the OFS, indicating a mapping between a subnet and a switch output port (interface d in Figure 1). Each of these ports constitutes a "branch" and is assigned to one firewall. The OFC makes its assignment trying to balance the long-term traffic average over the N active firewalls.

Since during normal network operation the traffic fluctuates and different branches experience traffic bursts, the OFC regularly monitors the traffic volumen (bps and pps) in each branch in order to take corrective action in case one of the branches experiences an overload. If the OFC considers that an overload is inminent, it will choose one (or more) subnets for Firewall HandOver (FHO), migrating that subnet traffic to another firewall.

In order to successfully perform load migration without session interruption in response to traffic peaks, an Active Session Collector Server (ASCS) is included in the design. Once a subnet has been targetted for FHO, the ASCS is instructed to monitor and record all the active sessions belonging to that subnet, by means of the interface a in Figure 1. The ASCS communicates to the OFC (interface b) the list of active sessions in the subnet, so that the OFC can avoid interrupting them. That is, when the OFC migrates a subnet's traffic to a different firewall, it only migrates *new* sessions, leaving existing ones untouched.

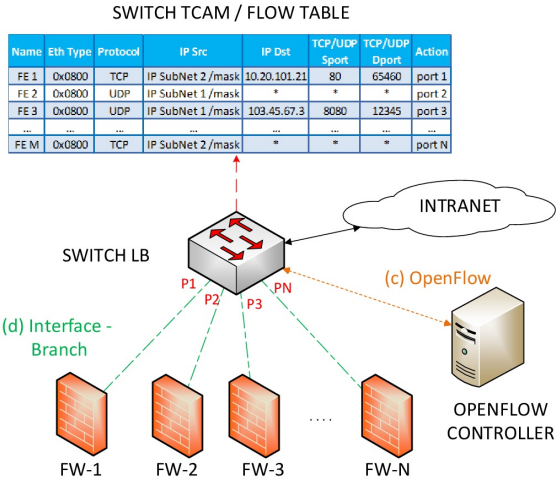

Fig. 2. Architecture of the load balancing module.

The remaining of this section describes the different modules employed in our solution more in detail.

A. *Load Balancing and Traffic Monitoring Module*

The load balancing module operates through the creation of flow entries according to pre-established subnets based on long term traffic averages. For example in Figure 2, let's suppose that there are M subnets in the intranet. For each subnet a flow entry is pushed into the switch specifying its prefix, mask and action (all the packages of SubNet 1 will be redirected to Firewall 1, from SubNet 2 to Firewall 2 and so on). Since the number of subnet in a enterprise network (M = 300 in the PUCP network) is less than the number of entries in the TCAM memory of the OFS switch (8000), there is plenty of room left for adding rules associated to the FHO process discussed below.

Since the OpenFlow specification mandates that each OFS keeps a counter with the number of hits (header matches) of each FE, the OFC can query the switch for this information every T seconds (T = 2 in our implementation) to determine the traffic volume (packets per second) over each branch/firewall. Similarly, the OFS keeps a counter of the number of the number of bytes associated (being matched) by a FE, and this way the OFC can also determine the traffic volume (bits per second) over each branch.

B. *Active Sessions Collector Server (ASCS) module*

The ASCS module is responsable to determine the active sessions of a subnet targeted for FHO, in order to prevent these sessions from being interrupted during the subnet's FHO. Therefore, this module is very time sensitive, as latencies in processing session information may result on session interruption. For example, if the time between the ASCS determination of the active sessions and the OFC instructing the OFS of migrating the subnet traffic is too high (say, milliseconds) then there is a distinct possibility of a session arriving in that interval, being directed towards the old branch only to be migrated a few milliseconds later to the new branch.

Furthermore, care should be taken to avoid the ASCS process to overload the system. A naive implementation that monitors all the traffic will deplete the processing resources. Instead, the system should divert/process as little traffic as posible.

Figure 3 shows a diagram of our ASCS process, which starts when the OFC decides that a subnet S, going through branch 1 is target for a FHO. The process consists of the following steps:

- The OFC instructs the OFS to mirror all the traffic of subnet S (and only subnet S) toward the ASCS module. This is done by inserting a new rule (rule M+1) with higher priority that the rest indicating that the subnet S traffic should be sent through two ports: 1 (existing branch) and X (connecting to the ASCS module). Once this entry is properly loaded in the OFS, the original FE (subnet S -> port 1) can be safely removed (make-before-break).
- For each packet arriving to the ASCS module, it determines if it corresponds to a new session (defined by the quintuple: src_ip, dst_ip, protocol, src_port, dst_port), and if so it records the session information and instructs the OFC to introduce an FE with highest priority indicating that this session should be output through port 1, overwritting the subnet-generic FE (both the original one if still there, and the traffic mirroring one). Thus, shortly after the firts packet arrives, the session traffic stops flowing through the ASCS module. The ASCS module records the time it sends this notification to the OFC.
- On the other hand, if a packet arriving at the ASCS belong to an already observed session, the ASCS either discard the packet or send a new notification to the OFC if the time between the original notification and the new arrival is greater that T3 seconds. That is, the ASCS avoids sending a notification for each arriving packets, imposing the hold off timer of T3 seconds for these notifications.

After a short while, only the first packets of newly arriving sessions will reach the ASCS, which therefore will be able to process them in a timely fashion. Also, note that L new FE have been added to the OFS.

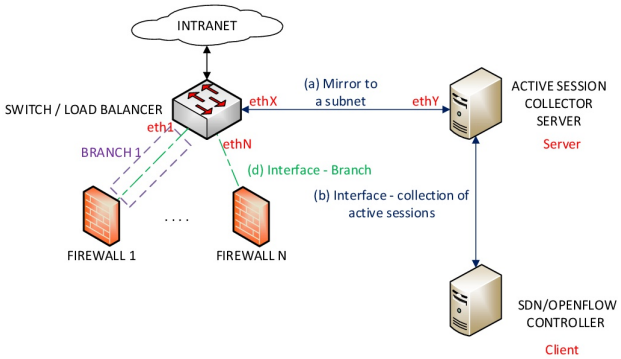

Fig. 3. Active session collector service operation.

C. Firewall HandOver Process or Load Migration

Internet traffic fluctuates due to the presence of burst traffic destabilizing the performance of a firewall. The firewalls of the N: 1 protection model exceeds their processing capacity at peak traffic times if the load distribution by the balancer is not homogeneous. To avoid this phenomenon and use all available resources of the firewall bank N: 1, the Firewall Hand Over (FHO) process migrates subnet traffic in response to temporary traffic bursts. A simplified pseudocode of the FHO process is shown in Figure 4.

```
Input: Total_Traffic, Subnet_Traffic.
Insert_Default_Subnet_In_Each_Branch().
Each T1 seconds
    branch_available = Find_Branch_Available().
    For each branch of Branchs do
        branch_avarage[n] = ρ*branch_average[n-1] + (1- ρ)*branch_traffic.
        If (branch_avarage[n] > umbral_traffic) during T2 seconds then
            subnet_x = Find_SubNet_In_Branch().
            Insert_FlowsEntries_Active_Sessions(subnet_x, branch).
            Move subnet_x to branch_available.
        End
    End
End
```

Fig. 4. Firewall HandOver Process simplified pseudo-code.

As shown in Figure 4, every $T_1 = 1$ seconds the OFC queries the OFSes to determine the traffic load in each branch. To smooth over short-term fluctuations and keep only medium-term tendencies, a first order filter is applied (by using a forgetting window with $\rho = 0.8$).

If the OFC estimates that an overload over a branch in the next T_2 seconds is about to happen, it initiates the FHO process by selecting a subnet traversing over that branch to be migrated. Currently, the overload is predicted by comparing the smoothed traffic against a threshold below the branch capacity. More complex prediction techniques (e.g., second order Kalman filters) are left for future optimizations. The subnet chosen for FHO is the subnet owth lower traffic intensity. If this subnet has too little traffic intensity, a second subnet is also chosen, and so forth. It should be noted that if during the next T_2 seconds the traffic on the congested branch decreases below the threshold, the FHO process is aborted.

After T_2 seconds, if the traffic in the congested branch has not been reduced, then OFC modifies the target subnet OFS rule, replacing its action from "send toward original port AND mirroring port" with the action "send toward new port" (again, a make-before-break approach is employed). Note that this action will only affect new traffic, as the existing traffic will be handled by the higher priority session-specific rules introduced during the ASCS monitoring stage.

One important design parameter is T_2: a large value means waiting too long to alleviate the congestion and it may result in packet losses or having to introduce a much lower threshold ("umbral" in Figure 4) reducing the branch utilization/efficiency. A small value, on the other hand, risks missing active sessions that were quiet (i.e., idle) during the T_2-

seconds monitoring interval. Thus, extra care was taken in selecting the value of T_2, as explained in the next section.

IV. IMPLEMENTATION OF THE LOAD BALANCER

We implemented a prototype of our subnet-level load balancer and Firewall HandOver in Python, on top of the FloodLight controller, and communication with it through its REST API. The prototype of the Active Session Collector Server (ASCS) is done using the BaseHTTPServer library from Python. We use the BaseHTTPRequestHandler class that handles HTTP requests (GET / POST) that arrive from the client to the server. The capture of packets by the interface (ethY), according to Figure 3 is done with the sniff tool of the Python Scapy library. Constantly monitoring the packets coming from the subnets to be migrated.

Our prototype has been implemented on top of an emulated network in the ExoGeni rack [10] of our group (GIRA-PUCP, Lima, Peru). This network is shown in Figure 5. It consists of 1 VM with 4 cores running ovs-switch (version 2.5.0) - Linux kernel version 3.13 as the SDN / OpenFlow switch, 1 VM running the Floodlight controller, 1 VM as HTTP server for the ASCS (Linux kernel version 3.13), and several VMs with Linux kernel version 3.13 with the conntrack tool libraries and iptables as firewalls. IP traffic is injected by 4-cores VM with 12GB RAM, Linux kernel version 3.13, and Python version 2.7.12 plus the Scapy library.

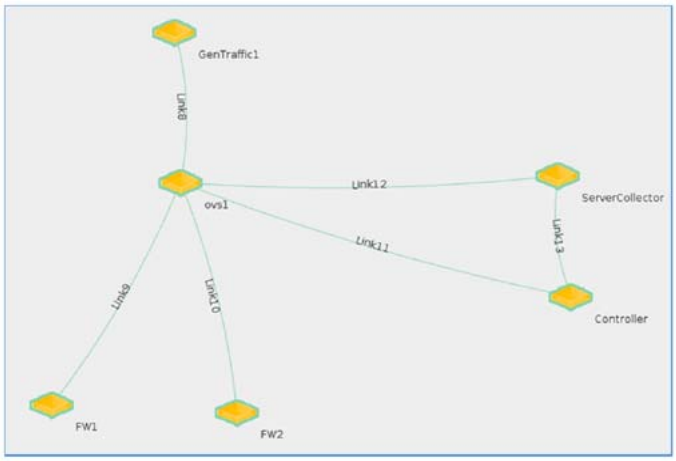

Fig. 5. Test topology in the ExoGeni Test-Bed.

As a previous step to determine the proper value of T_2, we analize traffic traces to determine the statistics of interpacket times (namely, "gap intervals") in active sessions. We used publicly available ISPDSL-II traces [9], that although dated with respect to traffic intensities it is likely still valid with respect to intrasession packet timing dynamics. In total, 14 traces were processed, from 20100106-030946-0.dsl to 20100106-093000-0.dsl, which together represent 6 and a half hours of traffic. From the 484, 700,000 packets there are 14, 509,555 sessions where the most frequent traffic is due to TCP and UDP flows (see Table 1). For this reason, the present work focuses on the analysis of these two protocols for the development of load balancing policies.

TABLE I. TRACE STATISTICS IN SIX HOURS

Statistics\Protocol	ICMP	TCP	UDP	GRE	ESP
% packets	0.31	77.5	21.9	0.2	-
% traffic volume (bytes)	0.05	89.4	10.3	0.17	-
% flows	2.5	60.4	36.9	-	-

We ordered the sessions by traffic intensities and for the top thousand we look into their intrasession "gap intervals". Table II shows the results for those sessions with gaps greater than 4 seconds. It should be noted that although some TCP sessions indeed present gaps of several minutes, the packets after the gap are just TCP control traffic to close the session. That is, there is no useful data traffic after the gap, and consequently, there will be no harm in migrating/interrupting those sessions. Those cases are marked in red in Table II. Thus, we can see from Table II than only 6 out of more than 14 million sessions present a gap over 15 seconds (i.e., chance of a session having a gap greater than 15 seconds is less than 10^{-6}). Furthermore, these are long-lived sessions, and the chance that the gap coincides with the time of Firewall HandOver is small (in the order of 10^{-3} for a $T_2 = 15$). Thus, we chose 15sec. as the period for the collection of active sessions since this value induces a probability of interrupting a session in the order of 10^{-9}, well below the probability of blocking due to congestion or other network problems. It should be noted that this value is conservative, as typical usage for session timeout in Openflow entries is in the order of 5 sec.

TABLE II. TCP FLOWS MORE VOLUMINOUS VS MAXIMUM TIME BETWEEN SESSION PACKET

Dense TCP Flows (decendent) / Inter-ArrivalTime (seconds)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Gap - 1	131.77	85.44	35.76	33.97	33.61	25.70	24.44	24.15	14.43	12.56	9.62	9.41	8.94	7.64	7.26	5.64	4.47
Gap - 2	10.56	16.20	14.45	23.20	12.95	12.87	8.10	14.05	7.45	6.33	4.81	5.66	5.18		7.08	5.57	
Gap - 3	8.65	12.72	14.15	5.98	9.88	7.92		8.65	5.64						5.53	4.84	
Gap - 4	5.29	6.15	7.82		6.64	6.41		7.02							4.42	4.18	
Gap - 5			5.93					4.37									

V. TESTS AND RESULTS

In this section we present results obtained using the first 7M packets (out of 42M) from the ISPDSL-II trace 20100106-083000-0.dsl [9]. We classify the traffic into two subnets: Subnet 1 - 0.0.0.0/1 and Subnet 2 – 128.0.0.0/1. The traffic generated has an average intensity of roughly 110 Mbps, with temporary peaks reaching up to 120Mbps. Subnet 1 has the greater traffic intensity.

We modeled a scenario where the total capacity of the branch carrying Subnet 1 and Subnet 2 traffic is 115Mbps. We therefore set the threshold for FHO in 105 Mbps. We then test that the FHO process is effective in migrating traffic (Subnet 2 in this case) to a second branch, preventing overload without packet losses or session interruptions.

Figure 6 (top) shows in blue the traffic being carried over branch 1 while using our load balancer. Also, the curve in red shows the original traffic that would be carried over branch 1 if our system was not present. It can be seen that the original system would result in overload arund time 160sec. Our system, on the other hand, avoid congestion by migrating traffic to another branch (shown in Figure 6 bottom). To help visualize the process, the black dashed curve in Figure 6 top shows the output of the smoot filter and the time when this output hits the threshold (105Mbps) around 140sec. At this time, the ASCS starts collecting information about the active

sessions (2758 in total) and inserting session-specific OpenFlow Entries in the switch. It can be seen that 15 seconds later the system starts redirecting new session traffic toward the second branch (Figure 6 bottom).

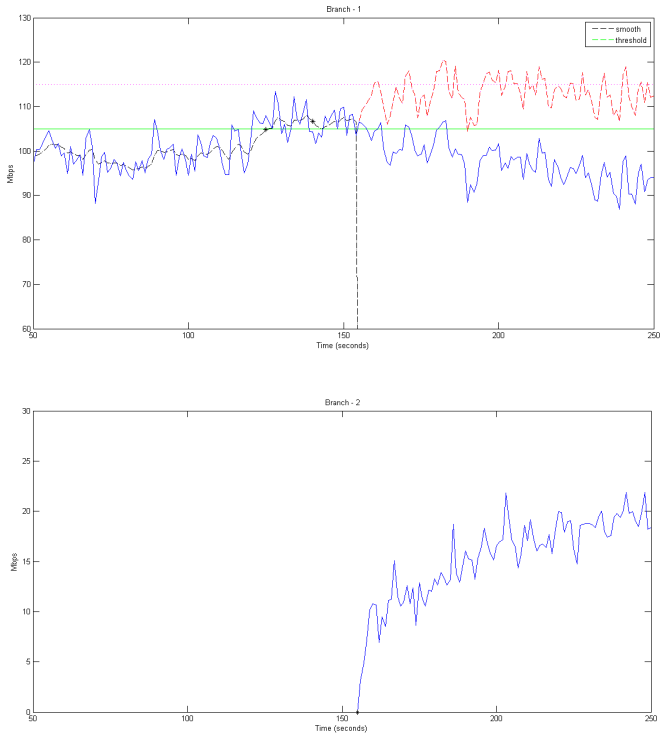

Fig. 6. Firewall HandOver procces for Subnet 2. Top: traffic though branch 1. Bottom: traffic through branch 2 (starting after FHO process).

VI. CONCLUSIONS

A novel subnet-level load balancer design based on an SDN / OpenFlow switch is presented. The design guarantees the permanence of the active sessions after the traffic migration (Firewall handover process) in response to traffic bursts. The proposed balancer can be used to implement a N:1 Firewalls array to significantly reduce the Capital Cost (CapEx). We present our prototype implementation in the ExoGeni Test-Bed. Initial test results for rates in the order of 110 Mbps. Testing at higher rates has been limited by intense traffic fluctuations introduced by the Traffic Generator based on the Python Scapy library, which is unable to deal with high rates or time intervals in the order of microseconds. It is estimated that in real scenarios (transmission rate higher than 1 Gbps) the performance of the balancer will improve, as fluctuations will decrease. Testing of 1Gbps traffic intensities over our SDN testbed composed of Pica 8 switches will be our next task.

Future works include: analysis of the impact of choosing the more or less voluminous subnet in the system, the implementation of a "firewall bypass" module that migrate elephant sessions deemed safe such as streaming video (similar to the work done in SciPass[2]), and the testing of the performance of the load balancer in a CyberSecurity Test-Bed with more traffic traces and commercial firewalls.

REFERENCES

[1] S. Kaur, J. Singh, K. Kumar, and N. S. Ghumman, “Round-Robin Based Load Balancing in Software Defined Networking,” *2nd International Conference on Computing for Sustainable Global Development*, 2015.

[2] Balas, E., & Ragusa, A. (2014). SciPass: a 100Gbps capable secure Science DMZ using OpenFlow and Bro. *In Supercomputing 2014 conference* (SC14).

[3] K. Govindarajan, V. Kumar, “An Intelligent Load Balancer for Software Defined Networking (SDN) based Cloud Infrastructure” *2nd Intern. Conf. on Electrical, Computer and Comm. Tech.* (ICECCT), 2017.

[4] Walber José Adriano Silva, Djamel Fawzi Hadj Sadok, "Control inbound traffic: Evolving the control plane routing system with Software Defined Networking", *High Performance Switching and Routing (HPSR) 2017 IEEE 18th International Conference on*, pp. 1-6, 2017, ISSN 2325-5609.

[5] K. Kaur, S. Kaur and V. Gupta, "Flow statistics based load balancing in OpenFlow," *2016 International Conference on Advances in Computing, Communications and Informatics* (ICACCI), Jaipur, 2016, pp. 378-381.

[6] N. Kang and M. Ghobadi and J. Reumann and A. Shraer and J. Rexford, “Efficient traffic splitting on commodity switches,” In *Proc. 11th ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT 2015.

[7] Arpit Gupta, Robert MacDavid, Rudiger Birkner, Marco Canini, Nick Feamster, Jennifer Rexford, Laurent Vanbever, “An Industrial-Scale Software Defined Internet Exchange Point”, *13th USENIX Symposium on Networked Systems Design and Implementation* (NSDI’16), 2016.

[8] K Newmeyer, A Cubiero, M Sanchez, “Ciberespacio, Ciberseguridad y Ciberguerra”, *II Simposio internacional de Seguridad y Defensa*: Perú 2015., 76-95.

[9] WAND, «WAND Network Research Group,» January 2010. [Online]. Available: https://wand.net.nz/wits/ispdsl/2/.

[10] I. Baldine et. al., “ExoGENI: A Multi-Domain Infrastructure-as-a-Service Testbed,” In *Testbeds and Research Infrastructures*, pp 93-117, Springer, Berlin, Heidelberg (2012).

[11] T. Xu et. al., “Mitigating the Table-Overflow attack in Software-Defined Networking,” *IEEE Trans. On Network and Service Management*, Vol. 14, No. 4, Dec. 2017.

[12] V Harkal, A Deshmukh, “Software Defined Networking with Floodlight Controller,” *International Journal of Computer Appications*, 2016., 23-28.