Poster: On the Safety and Efficiency of Virtual Firewall Elasticity Control[*]

Hongda Li[†#], Juan Deng[†#], Hongxin Hu[†], Kuang-Ching Wang[†], Gail-Joon Ahn[‡], Ziming Zhao[‡] and Wonkyu Han[‡]

[†]Clemson University [‡]Arizona State University
{hongdal, jdeng, hongxih, kwang}@clemson.edu, {gahn, zzhao30, iamhwk}@asu.edu

ABSTRACT

Firewalls have been typically used to enforce network access control. Network Functions Virtualization (NFV) envisions to implement firewall function as software instance (a.k.a virtual firewall). Virtual firewall provides great flexibility and elasticity, which are necessary to protect virtualized environments. In this poster, we propose an innovative virtual firewall controller, `VFW Controller`, which enables safe, efficient and cost-effective virtual firewall elasticity control. In addition, we implement the core components of `VFW Controller` on top of NFV and SDN environments. Our experimental results demonstrate that `VFW Controller` is efficient to provide safe elasticity control of virtual firewalls.

KEYWORDS

Virtual Firewall; Network Functions Visualization; Software-Define Networking; Elasticity Control

ACM Reference format:
Hongda Li[†#], Juan Deng[†#], Hongxin Hu[†], Kuang-Ching Wang[†], Gail-Joon Ahn[‡], Ziming Zhao[‡] and Wonkyu Han[‡] [†]Clemson University [‡]Arizona State University {hongdal, jdeng, hongxih, kwang}@clemson.edu, {gahn, zzhao30, iamhwk}@asu.edu . 2017. Poster: On the Safety and Efficiency of Virtual Firewall Elasticity Control. In *Proceedings of SACMAT'17, Indianapolis, IN, USA, June 21-23, 2017,* 3 pages.
https://doi.org/http://dx.doi.org/10.1145/3078861.3084166

1 INTRODUCTION

Firewalls have been widely used to enforce network access control. Traditional hardware-based firewalls are often placed at fixed network entry points and have a constant capacity with respect to the maximum amount of traffic they can handle per time unit. However, today's prevailing virtualized environments have fluid network perimeters and significantly variation of network traffic [1] [4]. Therefore, it is difficult to deploy hardware-based firewalls to protect virtualized environments. Two emerging network paradigms, Network Functions virtualization (NFV) and Software-Defined Networking (SDN) facilitate a new type of firewalls, *virtual firewall (VFW)*, which feature flexibility and elasticity and are well suited to protect virtualized environments. NFV implements firewall function as software instance that can be created or destroyed quickly to handle traffic volume variations, while SDN, recognized as complementary technology to NFV, seamlessly provides dynamic traffic steering support toward flexible, on-demand placement of virtual firewalls. Major commercial virtualized environments (e.g., VMware vCloud, Amazon AWS, VCE Vblock) have recently started to embrace virtual firewalls.

However, to fully take advantage of VFW benefits, our study reveals that there are great challenges to enable VFW elastic scaling. When VFW is overloaded, new instances are quickly created. Selective firewall rules and states are migrated to new instances and corresponding flow rules in SDN switches are updated to redistribute traffic. When multiple VFW instances are underloaded, some instances are destroyed, all firewall rules and states on them are migrated to remaining instances, and flow rules require update properly. The scaling of VFW must be *safe, efficient* and *optimal*. A *safe* scaling does not cause legal traffic to be dropped or illegal traffic to be allowed. An *efficient* scaling ensures that the latency overhead caused by scaling is bounded. An *optimal* scaling consumes minimum compute and network resources.

We identify four key challenges to achieve *safe*, *efficient* and *optimal* VFW scaling. First, semantic consistency of security policies must be preserved after scaling. Preserving semantic consistency of security policies after rounds of splits and mergences is non-trivial because firewall rules are often logically entangled with each other. Second, network flow rules in SDN switches must be correctly updated to redistribute traffic to corresponding instances. Third, a *safe* scaling also requires handling in-flight traffic during migration. Existing works [2] [3] buffer the in-flight traffic. However, in practice, buffer size is not unlimited and migration of different firewall rules incurs different amount of in-flight traffic. Therefore, care must be taken while selecting firewall rules to migrate to avoid buffer overflow. In addition, resources for VFW provision are neither unlimited nor free, thus it is important to optimize the resource usage during VFW scaling.

[*]A conference paper version of this poster appears in Proceedings of the 24th Network and Distributed System Security Symposium (NDSS), 2017.

[#]The first two authors contribute equally to the paper.

ACM ISBN 978-1-4503-4702-0/17/06…$15.00
https://doi.org/http://dx.doi.org/10.1145/3078861.3084166

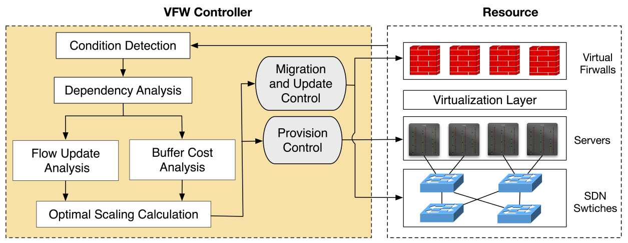


Figure 1: `VFW Controller` components and workflow.

In this poster, we propose a novel virtual firewall controller, `VFW Controller` to address above challenges and achieve *safe*, *efficient* and *optimal* VFW scaling. `VFW Controller` applies packet space analysis to identify intra-dependencies of firewall rules and group-based migration strategy to guarantee semantic consistency. To correctly update flow rules, inter-dependencies between firewall rules and flow rules are identified. To avoid buffer overflow, we model migration process and predict the amount of in-flight traffic during migration. Finally, `VFW Controller` adopts a three-step heuristic approach and integer linear programming to achieve optimal scaling. We implement `VFW Controller` on a real NFV and SDN platform and our experimental results show that our `VFW Controller` provides efficient VFW scaling control.

2 VFW CONTROLLER

The components and workflow of `VFW Controller` are shown in Figure 1. `VFW Controller` monitors each VFW instance and detects traffic overload and underload conditions. Once a condition is detected, `VFW Controller` first performs *Dependency Analysis*, *Flow Update Analysis* and *Buffer Cost Analysis*. Then the results are utilized by *Optimal Scaling Calculation*. Finally, *Provision Control* and *Migration and Update Control* interact with the compute and network resources and execute VFW scaling.

2.1 Dependency Analysis and Semantic Consistency

Definition 1 (Packet space). *Packet space of a rule r, denoted as $PS(r)$, is defined as a 5-dimensional hyperspace with dimensions being protocol, source IP, source port, destination IP, destination port.*

Definition 2 (Direct dependency). *Two rules r_i and r_j in a rule set $\mathbb{R}$ are directly dependent iff $PS(r_i) \cap PS(r_j) \neq \emptyset$, where $PS(r_i)$ is the packet space defined by r_i, and $PS(r_j)$ is the packet space defined by r_j.*

Definition 3 (Indirect dependency). *Two rules r_i and r_j in a rule set $\mathbb{R}$ are indirectly dependent iff $PS(r_i) \cap PS(r_j) = \emptyset$ and there exists a subset $R \subseteq \mathbb{R} \backslash \{r_i, r_j\}$ such that $PS(r_i) \cap PS(R) \neq \emptyset$ and $PS(r_j) \cap PS(R) \neq \emptyset$.*

Rules that have *direct* or *indirect dependencies* are put into one *group*. We identify the relation between a firewall rule group V and a flow rule group F is one of the following:

- Independency *iff* $PS(V) \cap PS(F) = \emptyset$. We denote independency as $V \overset{ind}{\rule{1em}{0.4pt}} F$.
- Congruence *iff* $PS(V) = PS(F)$. We denote congruence as $V \overset{con}{\rule{1em}{0.4pt}} F$.
- Superspace *iff* $PS(V) \supset PS(F)$. We denote superspace as $V \overset{sup}{\rule{1em}{0.4pt}} F$.
- Subspace *iff* $PS(V) \subset PS(F)$. We denote subspace as $V \overset{sub}{\rule{1em}{0.4pt}} F$.
- Intersection *iff* $PS(V) \cap PS(F) \subset PS(V)$ and $PS(V) \cap PS(F) \subset PS(F)$. We denote intersection as $V \overset{int}{\rule{1em}{0.4pt}} F$.

Definition 4 (Inter-dependency). *A firewall rule group V and a flow rule group F are inter-dependent if $PS(V) \cap PS(F) \neq \emptyset$.*

Group-Based Migration Strategy: *To guarantee semantic consistency, firewall rules in a group are migrated to the same destination virtual firewall instance in the same order as they are in the source virtual firewall instance. The destination instance can only start to process traffic matching rules in a group until all the rules and flow states associated with the group are ready on the destination instance.*

2.2 Flow Update Analysis

We identify two types of operations for flow update: CHANGE and INSERT. Let $V_i \in \mathbb{V} = \{V_1, ..., V_m\}$ be the firewall rule group to be migrated from a source VFW to a destination VFW instance and $\mathbb{F} = \{F_1, ..., F_n\}$ be the set of flow groups on the SDN switch (SW). To find the updates on $\mathbb{F}$, `VFW Controller` iterates through $\mathbb{F} = \{F_1, ..., F_n\}$ sequentially, and compares the inter-dependency relation between V_i and each $F_j \in \mathbb{F}$ to determine the updates.

- If $V_i \overset{ind}{\rule{1em}{0.4pt}} F_j$, no update is required.
- If $V_i \overset{con}{\rule{1em}{0.4pt}} F_j$ or $V_i \overset{sup}{\rule{1em}{0.4pt}} F_j$, only CHANGE operation is required. For every flow rule $f \in F_j$, the `VFW Controller` changes its forwarding action to redirect corresponding traffic to the new instance.
- If $V_i \overset{sub}{\rule{1em}{0.4pt}} F_j$ or $V_i \overset{int}{\rule{1em}{0.4pt}} F_j$, `VFW Controller` compares each pair of $PS(v)$ and $PS(f)$, where $v \in V_i$ and $f \in F_j$.
  1. If $PS(v) \cap PS(f) = \emptyset$, f needs no update.
  2. If $PS(v) \supseteq PS(f)$, CHANGE is performed to f to redirect the traffic defined by f to the new instance.
  3. If $PS(v) \subset PS(f)$, INSERT operation is performed. A new flow rule f' is inserted right before f to redirect the traffic matching v to the new instance.
  4. If $PS(v) \cap PS(f) \subset PS(v)$ and $PS(v) \cap PS(f) \subset PS(f)$, INSERT operation is performed. A new flow rule f' is inserted right before f. Each field of f' is the same as f, except that the protocol, source IP, source port, destination IP, destination port fields of f' are the intersection of the respective fields of v and f, and the forwarding action of f' forwards the traffic to the new instance.

2.3 Buffer Cost Analysis

We first model the process of migration and then predict the amount of in-flight traffic based on our model. Figure 2 shows the work flow of migration control. Then we define the parameters as follows:

- $t0$: is the time that VFW$_1$ starts to transfer the firewall rules and flow states specified in *fspace*;
- $t1$: is the time that SW finishes the update;
- $d1$: is the transmission delay between SW and VFW$_1$;
- $d2$: is the transmission delay between SW and VFW$_2$;
- $d3$: the transmission delay between VFW$_1$ and VFW$_2$;
- $b1$: is the average time that VFW$_1$ spends processing a packet;
- $b2$: is the average time that VFW$_2$ spends processing a packet.

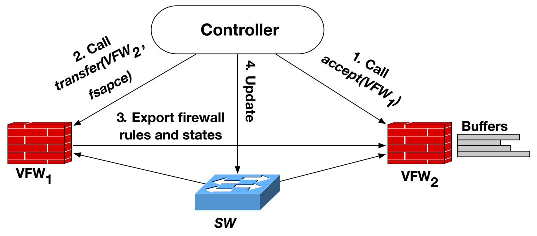

Controller
2. Call transfer(VFW 2, fsapce)
4. Update
1. Call accept(VFW 1)
3. Export firewall rules and states
Buffers
VFW 1
VFW 2
SW

Figure 2: Workflow of migration control.

At ***SW***, the traffic matching V_i that arrives before $t1$ is sent to VFW$_1$, and after $t1$ it is sent to VFW$_2$. Let Γ be a set comprising the matching traffic that arrives during $(t0 - d1, t1)$.

At **VFW**$_1$, the firewall rules and flow states defined in *fspace* are sent to VFW$_2$. Traffic in Γ starts to arrive after $t0$, and the last packet in Γ arrives before $t1 + d1$. VFW$_1$ processes all the traffic in Γ. VFW$_1$ finishes processing Γ before $t1 + d1 + b1$.

At **VFW**$_2$, traffic directly sent from *SW* starts to arrive after $t1 + d2$. The last packet in Γ arrives at VFW$_2$ before $t1 + d1 + b1 + d3$ and it is processed before $t1 + d1 + b1 + d3 + b2$. Therefore, traffic that is directly sent from *SW* and arrives at VFW$_2$ during $(t1 + d2, t1 + d1 + b1 + d3 + b2)$ is buffered.

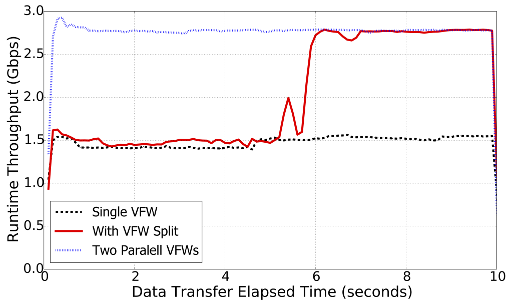
Suppose there are k_i flows matching V_i and the rate of flow j is λ_j. Then we estimate the buffer cost of V_i as

$$\beta_i = (\sum_{j=1}^{k_i} \lambda_j) \times \{(t1 + d1 + b1 + d3 + b2) - (t1 + d2)\}$$

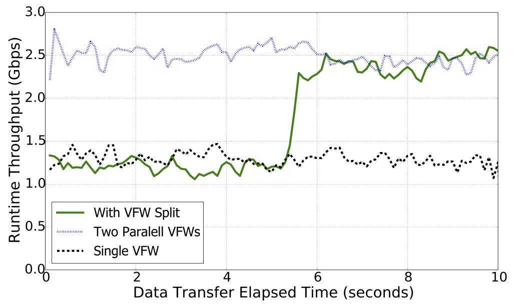
3 IMPLEMENTATION AND EVALUATION

We have implemented a prototype of `VFW Controller` on top of ClickOS, a Xen-based NFV platform. We used Floodlight and Open vSwitch to construct the SDN environments. We also implemented a stateful virtual firewall based on Click. In addition, we implemented 7 new Click elements to support our virtual firewall. Key functions of `VFW Controller` have been realized as individual modules. In particular, we have implemented a *Dependency Analysis* module based on Header Space Library (Hassel), a *Flow Updatea Analysis* module to find the correct flow updates, a *Buffer Cost Analysis* module to calculate buffer costs, and an *Optimal Scaling Calculation* module that realize the approaches for optimal scaling.

In the experiments, we setup three physical machines: *client*, *server* and *firewall*. The *client* sent traffic to the *server* via the *firewall* machine. Our `VFW Controller` was installed on *firewall* along with several VFW instances. We tested both TCP and UDP flows to see how quickly the VFW instances can scale when they were overloaded. Figure 3 shows that without scaling (black dashed), the throughput of *firewall* is around 1.3Gbps; with two parallel VFW instances (blue dashed), the throughput of *firewall* is around 2.8Gbps for UDP (Figure 3(a)) and 2.5Gbps for TCP (Figure 3(b)). We tested our `VFW Controller` by letting it control the VFW instance to scale when the instance was overloaded. We installed 400 firewall rules in a single VFW instance and 200 firewall rules in each of the paralleled VFW instances. For the VFW instance under test, we installed 400 firewall rules. It turns out that our `VFW Controller` can efficiently guide the overloaded VFW instance to scale out, in



(a) Split with UDP flow overload.



(b) Split with TCP flow overload.

Figure 3: `VFW Controller` for VFW elasticity.

less than 1 second in our experiments. After scaling, the throughput of *firewall* is equal to the throughput of two parallel VFW instances. In addition, this scaling preserved the TCP connection and UDP packet order.

4 CONCLUSIONS

In this poster, we have proposed `VFW Controller`, a virtual firewall controller, which enables *safe*, *efficient* and *optimal* virtual firewall scaling. Also, We have implemented the core components of `VFW Controller` on top of ClickOS. Our experiments showed that our `VFW Controller` is able to provide efficient and safe elasticity control of virtual firewalls.

ACKNOWLEDGMENTS

This work was partially supported by grants from National Science Foundation (NSF-ACI-1642143, NSF-ACI-1642031, NSF-IIS-1527421, and NSF-CNS-1537924).

REFERENCES

[1] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 267–280.
[2] A. Gember-Jacobson and A. Akella. 2015. Improving the safety, scalability, and efficiency of network function state transfers. In *ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*.
[3] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. 2014. OpenNF: Enabling innovation in network function control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 163–174.
[4] T. Benson and A. Anand and A. Akella and M. Zhang. 2009. Understanding data center traffic characteristics. In *Proceeding of SIGCOMM Workshop on Research on Enterprise Networking*. Barcelona, Spain.