ACLFLOW: An NFV/SDN Security Framework for Provisioning and Managing Access Control Lists

Leopoldo A. F. Mauricio*‡ Marcelo G. Rubinstein†, and Otto Carlos M. B. Duarte*
*Grupo de Teleinformática e Automação - Universidade Federal do Rio de Janeiro - COPPE/UFRJ, Brazil
†Universidade do Estado do Rio de Janeiro - FEN/DETEL/PEL, Brazil
‡Globo.com, Brazil
Emails: leopoldo@corp.globo.com, rubi@uerj.br, {leopoldo,otto}@gta.ufrj.br

***Abstract*—Router Access Control Lists (ACLs) are a traditional way to selectively filter traffic on cloud computing. However, the number of rules required may be large, whereas the storage capacity of router Ternary Content Addressable Memories (TCAMs) is scarce and expensive. This paper proposes a Network Functions Virtualization (NFV)/Software-Defined Networking (SDN) security framework, named ACLFLOW, which: (i) translates regular ACLs (source/destination IP, source/destination port, and protocol) into OpenFlow filtering rules; (ii) creates and manages large OpenFlow ACLs on distributed software switches, which act as security virtual network functions (named OpenFlow VNF-ACLs), to address the TCAM storage capacity problem; (iii) implements a proposed algorithm to dynamically prioritize the most popular rule to accelerate switching operations; and (iv) orchestrates and accelerates the deployment of NFV/SDN environments into production clouds. We have implemented a framework prototype into the Open Platform for NFV (OPNFV) and evaluated its performance using different tools and scenarios. Results show that the OpenFlow VNF-ACL improves maximum throughput by up to 90%, its HTTP request rates are up to 50% better, and it reduces Round Trip Time (RTT) by 70% when its performance is compared with a stateless Iptables running in virtual machines. Moreover, the proposed algorithm dynamically improves the HTTP request rate of flows with the highest traffic volume by 15% and reduces RTT by 25% when compared with ACLFLOW without prioritization.**

I. Introduction

The implementation of cost-effective security policies to control the data flow between different networks is a major challenge in cloud computing [1]. These policies are generally Access Control Lists (ACL) processed in line speed by router Ternary Content Addressable Memories (TCAMs). However, TCAMs limit the number of filters to a few thousand rules, between 2-4k, and are up to 400 times more expensive than RAMs, with a power consumption up to 100 times higher [2].

Network Functions Virtualization (NFV), standardized by the European Telecommunications Standards Institute (ETSI), is a new technology that aims to make networks more agile and flexible and to reduce equipment and operating costs [3]. NFV proposes the deployment of Virtual Network Functions (VNFs) as software in general purpose hardware to be an alternative to using specialized network and security devices. With NFV, the number of physical devices decreases as well as the costs with heat dissipation, electricity consumption, and maintenance. Thus, it is possible to reduce both CAPEX and OPEX [4].

This work has been sponsored by CNPq, CAPES, Globo and FAPERJ.

Software-Defined Networking (SDN) separates the network control function of the forwarding function. Its main protocol is OpenFlow [5], [6]. Through it, the control plane inserts flow rules into the data plane to control its behavior. Therefore, by combining NFV with SDN, multiple types of network functions can be created in a data plane, such as a load-balancer, which makes decisions based on destination and source IP addresses [7]; a traffic manager, when the rules specify which paths the data flow should follow [8], [9]; a security firewall, which enforces filtering rules [10], [11], etc.

In this work, we argue that each cloud computing physical server can host an ACL security virtual network function (named VNF-ACL) to control its Virtual Machines (VMs) traffic, as an alternative to using router TCAMs or specialized security middleboxes. Moreover, we can orchestrate these VNFs as a distributed data plane of an SDN [11].

We propose a cost-effective security framework, named ACLFLOW, which translates regular ACLs (source/destination IP, source/destination port, and protocol) into OpenFlow filtering rules implemented in software switches running within virtual network functions (OF VNF-ACLs) to address the TCAM limited size problem. As VNF implementation generally reduces performance [3], [10], [11], our framework includes an algorithm that dynamically prioritizes the rule with the highest matching traffic volume, to accelerating switching operations. Furthermore, ACLFLOW implements a mechanism that orchestrates and accelerates the implementation of OF VNF-ACLs in production clouds.

We have implemented a framework prototype into the Open Platform for NFV (OPNFV) and evaluated its performance using different tools and scenarios. We have compared the performance of the OF VNF-ACL and Iptables running in virtual machines (Iptables VNF) [10]. Results show that VNF-ACL has up to 90% higher throughput, HTTP request rates are improved by 50%, and RTT (Round Trip Time) is reduced by 70% when compared to a stateless Iptables VNF. Moreover, ACLFLOW algorithm performance is evaluated using real OpenFlow access control list traces from a production environment. Results show that the prioritization algorithm improves OpenFlow VNF-ACL HTTP request rate by 15%

and RTT is reduced by about 25%.

The rest of the paper is organized as follows. Section II presents related work. In Section III, we briefly review some features of the OpenFlow protocol and of the Open vSwitch (OVS) used by the prioritization algorithm. Section IV describes ACLFLOW security framework. Section V discusses the proposed ACLFLOW prioritization algorithm. The experimental performance evaluation is presented in Section VI. Section VII presents conclusions and future work.

II. Related Work

The efficient and low-cost processing of policies is a subject that has required the attention of the research community because the rules are usually stored in TCAMs that are expensive, power-hungry and have a scarce rule space [12]. To address these problems, some solutions propose the optimization of the TCAM rule space, while others implement filtering rules in software switches that can store large amounts of policies [13], [14].

Kanizo et al. [15] implement a framework for decomposing large flow tables into small ones to distribute the rules among various heterogeneous switches with tables of limited size. Using a positioning algorithm, Kang et al. [7] optimize the rule space of multiple switches. An algorithm prioritizes the implementation of the filters related to the affected paths of all switches involved when changes in the network topology occur. Katta et al. [2] propose a hybrid hardware-software switch that provides large, low-cost rule tables using a cache algorithm that places the most important rules in a TCAM and redirects the cache missed rules to software switches.

Minlan et al. [16] propose a different kind of switch to reduce traffic between the data plane and control plane, improving network performance. All rules necessary for network operation are entered preemptively and proactively on these switches that are known as "authoritative". The regular switches act reactively, because of their shortage of rule space, sending the first packets of a new flow to the authoritative switches, to ask what to do about them. The authoritative switches forward these packets and install flow rules into the regular switches so that they directly deliver next packets.

The aforementioned solutions do not reduce OPEX/CAPEX costs since they still heavily rely on TCAMs processing. Besides that, traffic diverted is handled by ordinary software switches that do not accelerate switching operations for improving network performance.

Maqbool et al. [13] develop an application called T-Flex that implements virtualized TCAMs on ToR (Top-of-Rack) switches, using their onboard CPUs and virtual disk memories. T-Flex increases the TCAM ToR size, by performing the same approach that an Operating System (OS) does when it extends the amount of memory available to applications by using the virtual memory disk. Whenever an input packet does not match a rule stored in the TCAM switch, it is redirected to the switch CPU by a default lowest priority rule that redirects all "missed TCAM incoming packets" to T-Flex, which performs a lookup operation to find out a matching rule. After that, a new rule is inserted into the TCAM to accelerate subsequent packet processing. That solution still uses the expensive TCAMs and is not flexible because it is tied to the life cycles of network devices from different vendors.

Deng et al. [11] propose a framework for positioning and managing firewall virtual network functions, based on NFV and SDN, called VNGuard. VNGuard also defines a high-level language that simplifies policy management because users do not need to know low-level information from virtual networks to create filtering rules. The VNGuard components have been implemented in ClickOS [17]. However, there is a significant reduction in VNF network performance when the number of rules increases.

Kourtis et al. [18] implement a DPI (Deep Packet Inspection) virtual network function using SR-IOV, and DPDK features to enhance the VNF performance. SR-IOV provides efficient allocation of low-level network resources because it enables the direct access and control of the system hardware devices. The Intel DPDK libraries make it easy to deploy network-intensive applications by implementing a packet-processing engine that uses polling mode instead of the standard interrupt mode of the Linux network stack. All packet classification and forwarding actions happen in the userspace, with no copying from the kernel, for efficiently consuming CPU cycles. However, with this solution, all applications running on the operating system need to be changed to allow communication with the DPDK library and specialized NICs must be used. For this reason, the implementation cost increases and it could demand hardware changes on all existing physical servers to be fully implemented in a production cloud.

In [10], we have shown that small firewall virtual network functions with only 4 GB of RAM and two virtual CPUs can store and process more filtering rules than commercial top-of-rack router TCAMs, reducing costs and increasing flexibility. Nevertheless, as in [11], network performance decreases significantly when the number of filtering rules increases. One solution to this problem mentioned in [10] is to increase the number of firewall virtual network functions.

This paper extends [10] by proposing the NFV/SDN ACLFLOW security framework that implements filtering rules in OpenFlow VNF-ACLs with greater filter storage capacity than that found in the ToR router TCAMs. Our solution implements an algorithm that dynamically prioritizes the most popular rule to accelerate switching operations. The implementation of the ACLFLOW prioritization algorithm neither requires changes in applications running on the same operating system nor specialized hardware. Besides, there is no significant reduction in network performance when the number of rules increases.

III. OpenFlow Protocol and OVS Features

Software switches are widely used by virtualization tools to interconnect virtual machine network interfaces and forward their packets. They can store large amounts of rules, and some already reach rates of up to 40 Gb/s when running on machines

with only four cores [2]. OVS is an example of a software switch, widely used in cloud computing implementations.

OVS packet classification evaluates the first packets of a flow in the userspace (OVS pipeline), through queries executed in several hash tables (Tables 0 to *n* in Figure 1). The number of OVS hash tables varies according to the number of flow rule types. This happens because the OVS packet classification algorithm creates unique hash tables, automatically, in userspace, based on the different match field combinations of the OpenFlow rules it stores. For example, let P = {p1,...,pn} be the OVS flow rule types. When all OVS flow rules evaluate only the packet destination header, there is only one hash table in userspace, represented by:

$$(p1 \in \text{Table 1} \mid \text{p1} = [dst\_ip, *])$$

On the other hand, the OVS classifier automatically creates two hash tables in userspace when some flow rules evaluate only the IP destination address, while others deal with the source IP, destination IP, and TCP destination port fields, such as:

$$(\text{p1} \in \text{Table 1; p2} \in \text{Table 2} \mid \text{p1} = [dst\_ip, *] \text{ and p2} = [src\_ip, dst\_ip, dst\_port, *])$$

OVS pipeline makes only one query by each hash table (an O(1) process) to find a match. Therefore, when an OpenFlow firewall has 20000 ACLs created from the five tuples source IP, destination IP, source port, destination port, and protocol, only up to 32 different hash tables may exist in userpace to be queried by OVS packet classifier, no matter how many rules exist in the OpenFlow firewall. In this case, queries will be performed in linear time O(n), where n is the number of hash tables.

OpenFlow rules could be prioritized. When this happens, the OVS packet classifier reads the rule priorities in descending order, from highest to lowest, and caches the forwarding decision in the kernel data-path that speeds up the forwarding of subsequent packets [2], [14]. However, cached rules leave kernel space whenever their cache times expire, or every time the SDN controller changes rules in the data plane. When this happens, packets which do not match a rule in the fast-path kernel are sent back to the slower userspace processing [6].

Every time the matching rule does not have the highest priority, the sequential searching between hash tables goes on until the last table. However, when the matching rule already has the highest priority, the linear searching between hash tables is stopped, and the OVS immediately forwards the packet, without query other hash tables of the pipeline. Consequently, the number of queries performed in userspace is smaller.

OpenFlow reactive or proactive approaches could be used in an SDN implementation [5], [6]. In reactive mode, the controller installs flow rules in the data plane according to the request made by the OpenFlow switches. Whenever switches receive a new flow, they send an instruction request message *OFF-packet-in* to SDN controller, that inserts rules into the data plane to allow or block traffic. Although all subsequent

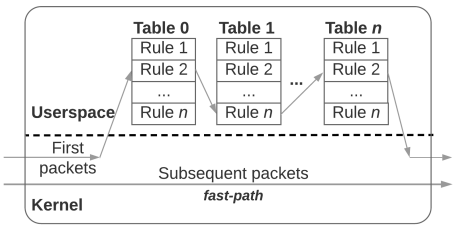


Figure 1. Open vSwitch architecture. First packets of a flow are evaluated in userspace and cached decisions are evaluated in kernel [14].

packets processed in the data plane have no controller intervention, this type of approach inserts delays, and the controller and switch buffers may become overloaded when the number of new flows is large [19].

All filtering rules are previously installed in the data plane when a proactive SDN is used to provide security. Thus, when the first packets of a new flow reach the OpenFlow switch, those that do not match the previously installed rules are immediately blocked, and the matching packets are forwarded without setup delays [20].

OpenFlow secure or standalone approaches could be configured in an SDN switch. When OVS runs in standalone mode, all rules created by SDN controller are erased, and OVS change its operation to act as the L2 switch if the communication with the controller is lost. Otherwise, when the OpenFlow secure mode is configured OVS persist all rules previously created in the data plane if communication with its SDN controller is interrupted. In this case, OVS still act as an OpenFlow switch and prior recorded access control rules remain active.

ACLFLOW builds OpenFlow VNF-ACLs that use OVS to create filtering rules, which are configured to act in the proactive mode. Thus, all rules are installed in VNF-ACL in advance. Besides, ACLFLOW implements an algorithm to dynamically prioritizes rules in userspace to accelerate the OpenFlow pipeline. The OpenFlow secure mode is configured in ACLFLOW VNF-ACLs to persist filtering rules if its communication with the SDN controller fails, preventing inappropriate traffic blocking and security failures. However, no new rule can be inserted into OpenFlow VNF-ACLs while communication with the controller is not active.

IV. The Proposed ACLFLOW Security Framework

ACLFLOW is an NFV/SDN security framework that creates distributed OpenFlow VNF-ACLs as an alternative to using router TCAMs or specialized security middleboxes to control traffic from Virtual Machines (VMs) in a cloud computing environment. Each physical cloud server of the ACLFLOW framework has an OpenFlow VNF-ACL that belongs to a security domain (Figure 2). Besides, a security domain also has a set of SDN controllers to manage multiple VNFs that can be grouped in the same rack or distributed across different pods or datacenters. All VNFs that belong to the same security domain have the same filtering rules enabling the secure migration of virtual machines without the risk of inappropriate traffic

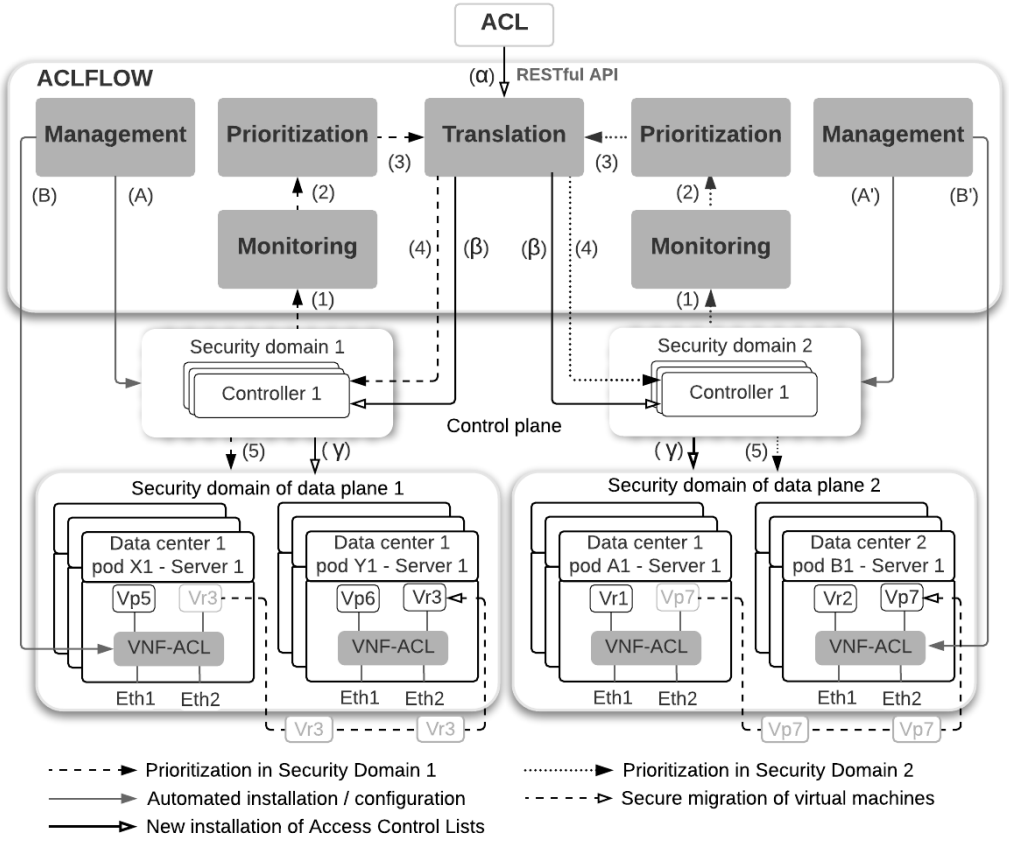


Figure 2. ACLFLOW security framework. A VNF-ACL is installed on each physical server in the cloud to provide performance and enable secure migration of virtual machines between servers.

blocking or security breaches. Thus, when a virtual machine migrates from one cloud computing physical server to another, in the same security domain, the filtering rules required for its operation are already replicated in the remote OpenFlow VNF-ACL. ACLFLOW translation, management, monitoring, and prioritization modules are illustrated in Figure 2.

ACLFLOW translation module is a python API that we have developed to transform regular ACLs, which evaluate the five-tuple source/destination IP header, source/destination port, and protocol, into OpenFlow filtering rules (12 tuples) [5], [6]. This module hides of network administrators the details and complexities of the distribution of OpenFlow rules among various VNF-ACLs.

Regular ACLs inserted as a JSON document into the RESTful interface of the ACLFLOW translator (Figure 2 - (α)) are sent to SDN controller(s) (Figure 2 - (β)), which insert OpenFlow filtering rules into the VNF-ACLs (Figure 2 - (γ)). Since only five tuples vary, the ACLFLOW translator inserts wildcards into the remaining OpenFlow header tuples when translating the regular ACLs, and each OpenFlow filtering rule receives a unique identifier. In this way, it is possible to obtain usage statistics and manage each access rule installed on any VNF-ACLs of the ACLFLOW security framework.

ACLFLOW management module provides an API and a web interface that we have developed as python applications [1]. The management automates the installation of SDN controllers (A and A' of Figure 2) and the configuration of OpenFlow VNF-ACLs (B and B' of Figure 2). Thereby, cloud administrators can quickly create an NFV/SDN environment with OpenFlow VNF-ACLs in a production environment.

The REST API or web interface input data of the ACLFLOW management are the IP address of the machine that hosts the Open vSwitch, the IP of the virtual machine where the management will install OpenDaylight (ODL), the security environment identifier where both ODL and OVS will be configured, and the IP addresses of the essential network services (servers related to DHCP, DNS, NTP, etc.) which must always be allowed for the operation of the ACLFLOW framework. Using these informations, the management module configures OpenFlow VNF-ACLs in a proactive mode so that reactive mode delays, which can create security problems when the number of new flows is large, are avoided [5], [6].

Finally, the management module configures the secure OpenFlow mode in VNF-ACLs to persist filtering rules previously created in the data plane if communication with its SDN controller fail.

ACLFLOW monitoring module collects the unique identifier of the filter rule (*rule.id*), the idle time of the rule (*rule.idle*), and the number of packets that match the rule

[1]available at https://bit.ly/2LDnbQA

(*rule.pk.count*), from the controllers API ((1) in Figure 2), and sends them to the prioritization module ((2) in Figure 2) at each time interval *T*. From this information, the **ACLFLOW prioritization module** performs the algorithm described in Section V to accelerate packet classification and performance of OpenFlow VNF-ACL. The prioritization module informs the *rule.id* and the new priority (*regular.priority* or *highest.priority*) for the translation module ((3) in Figure 2). Using the *rule.id* information, the translation module can update the filtering rule identified through a message that it sends to each SDN controller of the security domain ((4) in Figure 2). Then, SDN controllers update the priority of the identified rule on all VNF-ACLs in that security domain ((5) in Figure 2).

V. The Proposed ACLFLOW Prioritization Algorithm

The ACLFLOW security algorithm (algorithm 1) dynamically prioritizes the rule with the highest volume of traffic (the most popular rule) to increase the network performance of OVS, by reducing unnecessary searches across multiple hash tables in userspace to perform fast switching operations. Four different priority levels are implemented to accelerate the packet classification in userspace: dynamic, low, minimum, and static. The dynamic priority (*Pri.dynamic*) is configured for the rule identified as having the highest amount of traffic in each round of the algorithm Its value is equal to (*maximum priority*).

ACLFLOW sets up the low priority (*Pri.low*) on the regular filtering rules, like those created by tenants and cloud administrators to protect their applications. The default drop rule has the minimum priority ($Pri.drop = minimum\ priority$) and ACLFLOW sets up the static priority on the initial OpenFlow VNF-ACL filtering rules ($Pri.static = low\ priority + 1$). The initial rules are those necessary for the operation of the ACLFLOW framework. These rules allow established TCP connections, initial connection TCP packets (SYN packets) and DHCP, DNS, NTP, etc., that are related to essential network services traffic.

ACLFLOW security algorithm also implements two different thresholds: an Upper Threshold (UT) and a Lower Threshold (LT). UT is used to identify which rules have sufficient amount of traffic; consequently these rules are candidate to be prioritized. LT is used to find out which rules should be given low priority because they are associated with low traffic volumes. Therefore, filtering rules with amount of traffic higher than UT can be prioritized and those who no longer have the highest traffic volumes can go back to the low priority.

ACLFLOW prioritization module executes the Algorithm 1 at each time interval *T*. In lines 6–13 and 14-24 of Algorithm 1 each OpenFlow VNF-ACL filtering rule is analyzed. The initial rules priorities are never changed. Thus, the algorithm does not process their statistics (lines 7–8). Otherwise, the algorithm calculates and registers the difference between the current number of matching packets for each rule and previous one (line 10), and also calculates the number of rules processed (line 12) at each time interval *T*.

When the time without new matching for a rule is less than or equal to the time between two executions of the algorithm (*T*), it evaluates if the variation in the number of packets is greater than or equal to UT (line 15 of Algorithm 1). If so, the number of packets that have matched the rule since the last analysis is considered significant and this could be the filtering rule with the highest amount of traffic. Then, its rule ID (line 16) and current number of matching packets since the last algorithm execution are recorded (line 17). The identified most popular rule priority is dynamically sets up to the dynamic priority (line 25) in all OpenFlow VNF-ACLs through an update message sent to OpenFlow controller(s) (line 27). Thereby, the ACLFLOW prioritization algorithm dynamically speeds up the packet classification in userspace to improve the OpenFlow VNF-ACLs performance.

Otherwise, if the number of packets that have matched the rule since the last analysis is less than the lower threshold (line 18 of Algorithm 1), the traffic variation is considered small. Then, if the filtering rule analyzed still has a $Pri.dynamic$ priority, the algorithm records its rule ID in a list (line 19) of rules that have their priority reset to the default value, also through a message sent to an OpenFlow controller(s) (line 28).

Finally, the algorithm calculates the portion of inactive filtering rules since its last execution (lines 20,21, and 29) and the percentages of rules that never recorded packet traffic (lines 22,23, and 30). With this information, cloud administrators can remove long-term inactive rules, as well as identify who is no longer in use. They also can decrease the Upper Threshold to increase the number of rules prioritized in userspace. On the other hand, by reducing the Lower Threshold, administrators reduce the number of candidate rules to be prioritized.

VI. Evaluation and Results

This work extends [10] by proposing ACLFLOW VNF-ACL. The Iptables security virtual function presented in [10] is a stateless packet filter. Iptables classifies packets by performing linear, ordered and sequential searches on the rule set [21]. For this reason, its performance varies according to the position of the rule that matches the traffic. For this reason, we evaluated the performance of Iptables "first-match", Iptables "middle-match" and Iptables "last-match" and compared their performance with that of OpenFlow VNF-ACL.

The Open vSwitch, version 2.5.1, runs inside the OpenFlow VNF-ACL, configured to act in the proactive mode. Thus, all filtering rules are installed in VNF-ACL in advance. Only up to 32 hash tables are created inside OpenFlow VNF-ACLs by the five header tuples: source IP, destination IP, source port, destination port, and protocol [5], [6], [14]. In addition, no matter how many rules exist in each OpenFlow VNF-ACL hash table, the OVS packet classifier performs only one query (an O(1) lookup) in each hash table. Therefore, the increase in the number of rules does not significantly change the number of queries performed in the OpenFlow VNF-ACL, which is always less than 32.

Both security virtual functions run on Ubuntu 16.04 virtual machines. We use *Fuel* installation tool to install the Open

Algorithm 1: PRIORITIZATION ALGORITHM

input :
UT: Upper Threshold;
LT: Lower Threshold;
S: Initial rule unique identifiers;
R: OpenFlow VNF-ACL flow rules;
$rule.id$: Unique identifier of each rule;
$Pri.low$: Priority value of regular rules;
$Pri.static$: Priority value of initial rules;
$Pri.drop$: Priority value of the default deny rule;
T: Time between two executions of the algorithm;
$rule.idle$: Time without new matching for the rule;
$rule.pk$: Last number of packets that match the rule;
$rule.pk.count$: Number of packets that match the rule;
$Pri.dynamic$: Priority value of dynamically prioritized rules.

begin
  $IdleRules \longleftarrow 0$
  $regular.rule.ids[\ ] \longleftarrow 0$
  $MisusedRules, CountRules \longleftarrow 0$
  $highest.rule.id, highest.rule.\delta \longleftarrow 0$
  for $rule \in R$ **do**
    if $rule.id \in S$ **then**
      CONTINUE;
    else
      $rule.\delta \longleftarrow (rule.pk.count$ - $rule.pk)$;
      $rule.pk \longleftarrow (rule.pk.count)$;
      $CountRules \longleftarrow CountRules + 1$;
  end
  for $rule \in R$ **do**
    if $rule.idle \leq T$ **AND** $rule.\delta \geq UT$ **AND** $rule.\delta > highest.rule.\delta$ **then**
      $highest.rule.id \longleftarrow rule.id$;
      $highest.rule.\delta \longleftarrow rule.\delta$;
    if $rule.\delta \leq LT$ **AND** $rule.priority == Pri.dynamic$ **then**
      $regular.rule.ids[\ ] \longleftarrow rule.id$;
    if $rule.\delta == 0$ **then**
      $IdleRules \longleftarrow IdleRules + 1$;
    if $rule.pk.count == 0$ **then**
      $MisusedRules \longleftarrow MisusedRules + 1$;
  end
  $highest.priority \longleftarrow Pri.dynamic$;
  $regular.priority \longleftarrow Pri.low$;
  $UpgradePriority(highest.rule.id, highest.priority)$;
  $DowngradePriority(regular.rule.ids[\ ], regular.priority)$;
  $IdleRules \longleftarrow (IdleRules/CountRules)$;
  $RulesNeverUsed \longleftarrow (MisusedRules/CountRules)$;
end

Platform for NFV (OPNFV) [22] on four physical servers. Each node is an Intel (R) Core i7-4770 @ 3.40 GHz with eight cores, 32 GB of RAM and three 1 Gb/s Ethernet interfaces running a Ubuntu 16.04 operational system. The *Fuel* machine is a CentOS server that has the same hardware characteristics. Three physical servers of the OPNFV cloud are computing nodes [23] that host the three virtual machines used in performance analysis (Figure 3). These VMs have 4 GB of RAM and 2 VCPUs. The fourth OPNFV server is the OpenStack controller node [23].

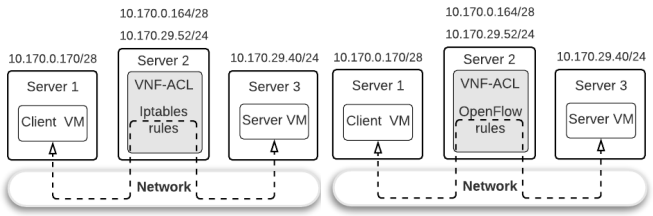
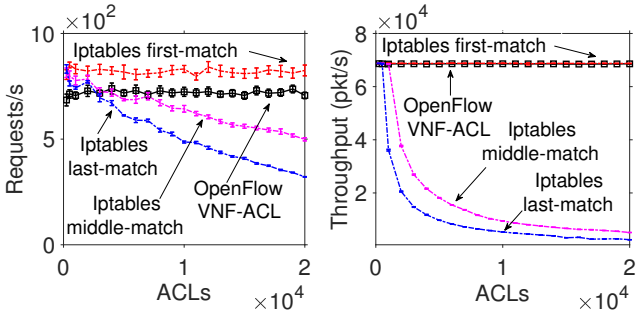


Figure 3. Test scenario with virtual network functions that implement ACLs on commercial off-the-shelf servers with (a) Iptables rules and with (b) OpenFlow rules (OpenFlow VNF-ACL).

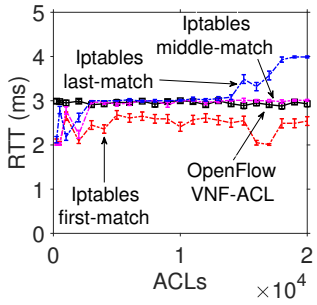
Communication between a client (IP 10.170.0.170) and a server (IP 10.170.29.40) traverses the VNF-ACLs shown in Figure 3. To measure the network performance we use httperf, Iperf, and Netcat tools. Httperf is used to measure the impact of VNF-ACLs on the maximum rate of HTTP requests that the client generates and reaches the server. In another experiment, UDP packets with 1472 bytes are sent from the client to the server by the Iperf tool to measure the maximum throughput. In a third experiment, we measure RTT using Netcat. Evaluations are averages of 15 rounds with 95% confidence intervals.

Iptables performance is compared to OpenFlow VNF-ACL without prioritization rules (OpenFlow VNF-ACL). The maximum HTTP request rate (Figure 4(a)), the maximum throughput (Figure 4(b)) and the Round Trip Time (RTT) (Figure 4(c)) are evaluated to identify the best solution when the number of rules varies from 250 to 20000. Results show better HTTP request rate (about 800 requests/s), higher throughput (about 68k pkt/s), and lower RTT (less than 3 ms) when the matching rule is in the first position of Iptables policies (Iptables first-match). In this case, the number of rules does not affect its performance.

RTT increases and the rate of HTTP requests and the maximum throughput decrease when the matching filtering rule is the middle position of the rule set (Iptables middle-match). The worst results happen when traffic is forwarded only when the last Iptables filtering rule is read (Iptables last-match). The increasing number of filtering rules causes a reduction on the throughput of more than 90%, the reduction of the HTTP request rate is approximately 60%, and the RTT increases by about 70%. That is the worst case for Iptables VNF.



(a) Maximum HTTP requests rate. (b) Maximum throughput.



(c) Round-trip-time.

Figure 4. Performance by increasing the number of rules. Evaluated metrics: a) maximum HTTP request rate; b) maximum throughput and c) RTT.

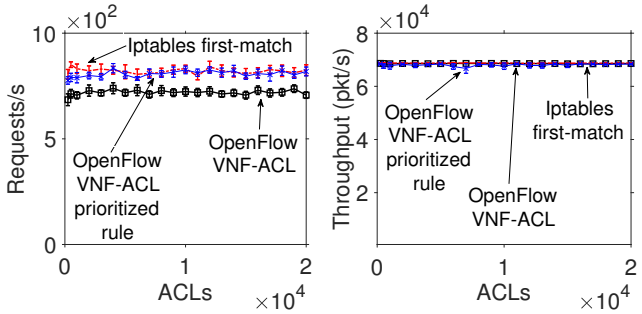
When comparing OpenFlow VNF-ACL without the prioritization engine to Iptables first-match, results show similar maximum throughput values, a slightly higher RTT for ACLFLOW, and a maximum HTTP request rate a little bit lower than the best case of Iptables. It is important to highlight that OpenFlow VNF-ACL performance is not affected by the number of rules.

To evaluate the efficiency of the ACLFLOW prioritization algorithm, we have deployed 22 OpenFlow VNF-ACL on 22 physical servers of the production datacenter used. These servers host more than 145 virtual machines, about 80 distinct applications, and 145 virtual networks. All OpenFlow VNF-ACLs are within the same security domain. Therefore, all VNF-ACLs have the same set of production filtering rules.

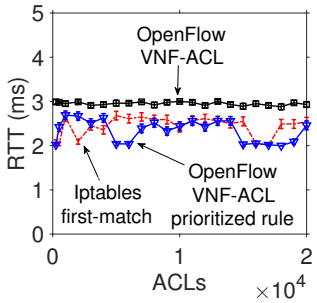
However, each OpenFlow VNF-ACL has a unique set of "matching statistics" because physical servers host distinct virtual machines with different traffic patterns. We have found 2860 filtering rules in each OpenFlow VNF-ACL, and we have used 22 different "traces", each for 15 times to obtain the averages with 95% confidence intervals, because 22 different number of rules have been used in our tests (250, 500, 1000, 2000, ..., 19000, 20000).

The prioritization algorithm time interval T of ACLFLOW is set up to 5 min and the cache entries in the OpenFlow VNF-ACL kernel space is set up to 60 s. The Upper Threshold is set up to prioritize filtering rules with packet variance greater than 2000. The Lower Threshold is configured to reset the priority of the rule with (*Pri.dynamic*) if packet variation less than 500. The priority of the default deny rule is 300, the non-privileged rules receive priority equals to 30000, and the most popular rules receive priority equal to 65001 (*maximum priority* == 65001).

When the evaluation starts the filtering rule that matches the test traffic generated by httperf, Iperf, and netcat is not prioritized immediately because another filtering rule in the production "trace" still records more traffic. After a few seconds, test traffic makes the matching rule the most popular one because of the highest volume of traffic.



(a) Maximum HTTP requests rate. (b) Maximum throughput.



(c) Round trip time.

Figure 5. Comparing the iptables first-match performance with OpenFlow VNF-ACL with and without dynamically prioritized rules. Evaluated metrics: a) maximum HTTP request rate; b) maximum throughput and c) RTT.

First-match Iptables is compared with OpenFlow VNF-ACL with and without prioritization of the matching rule. Figures 5(a), 5(b) and 5(c) show that the ACLFLOW prioritization module presents similar performance when compared with Iptables first-match. Moreover, it improves the HTTP request rate by about 15% and reduces RTT by about 25% when compared to ACLFLOW without prioritization. Performance is enhanced because the algorithm engine speeds packet classification into userspace to quickly return the rule with more traffic to the OpenFlow VNF-ACL kernel, where packets are rapidly forwarded.

VII. Conclusions and Future Work

This paper proposes the ACLFLOW security framework, which creates and manages security virtual network functions in software switches. The framework implements a management module that simplifies and automates the installation of OpenFlow VNF-ACLs into production clouds. Its translation

module transforms access control lists into OpenFlow filtering rules. Moreover, the ACLFLOW prioritization algorithm dynamically accelerates the packet classification of the highest-volume traffic, which is quickly sent to the kernel of the switch for performance improvement. In ACLFLOW, VNF-ACLs that have the same set of rules can be implemented in different physical servers in the same security domain to enable secure migration of virtual machines between those servers.

This paper also compares ACLFLOW VNF-ACL with the Iptables VNF performance. ACLFLOW VNF-ACL provides maximum throughput up to 90% higher, HTTP request rates up to 50% better, and RTT up to 70% less than that of a stateless Iptables VNF. Results also show that the prioritization algorithm dynamically improves the rate of HTTP requests for the flow with the highest traffic volume by about 15% and reduces RTT by about 25%. As a future work, we want to extend the management module to optimize the consumption of computing resources by deploying distributed controllers in the ACLFLOW security framework.

REFERENCES

[1] H. Takabi, J. B. D. Joshi, and G. J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security Privacy*, vol. 8, no. 6, pp. 24–31, Nov 2010.

[2] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 175–180. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620734

[3] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, First quarter 2016.

[4] Y. D. Lin, P. C. Lin, C. H. Yeh, Y. C. Wang, and Y. C. Lai, "An extended SDN architecture for network function virtualization with a case study on intrusion prevention," *IEEE Network*, vol. 29, no. 3, pp. 48–53, May 2015.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[6] N. Shelly, E. J. Jackson, T. Koponen, N. McKeown, and J. Rajahalme, "Flow caching for high entropy packet fields," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. –, Aug. 2014.

[7] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: ACM, 2013, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2535372.2535373

[8] L. Xie, Z. Zhao, Y. Zhou, G. Wang, Q. Ying, and H. Zhang, "An adaptive scheme for data forwarding in software defined network," in *2014 Sixth International Conference on Wireless Communications and Signal Processing (WCSP)*, Oct 2014, pp. 1–5.

[9] W. Braun and M. Menth, "Wildcard compression of inter-domain routing tables for openflow-based software-defined networking," in *2014 Third European Workshop on Software Defined Networks*, Sept 2014, pp. 25–30.

[10] L. A. F. Mauricio, M. G. Rubinstein, and O. C. M. B. Duarte, "Proposing and evaluating the performance of a firewall implemented as a virtualized network function," in *2016 7th International Conference on the Network of the Future (NOF)*, Nov 2016, pp. 1–3.

[11] J. Deng, H. Hu, H. Li, Z. Pan, K. Wang, G. Ahn, J. Bi, and Y. Park, "VNGuard: An NFV/SDN combination framework for provisioning and managing virtual firewalls," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network, NFV-SDN 2015*, 2015, pp. 107–114.

[12] C. Vivek and S. P. Rajan, "Z-tcam: An efficient memory architecture based TCAM," *Asian Journal of Information Technology*, vol. 15, no. 3, pp. 448–454, 2016.

[13] Q. Maqbool, S. Ayub, J. Zulfiqar, and A. Shafi, "Virtual TCAM for data center switches," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov 2015, pp. 61–66.

[14] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[15] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 545–549.

[16] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, Aug. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043164.1851224

[17] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, 2014, pp. 459–473. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/martins

[18] M. A. Kourtis, G. Xilouris, V. Riccobene, M. J. McGrath, G. Petralia, H. Koumaras, G. Gardikis, and F. Liberal, "Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov 2015, pp. 74–78.

[19] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, Oct 2013, pp. 18–25.

[20] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228283.2228297

[21] Iptables, https://wiki.archlinux.org/index.php/iptables, April 2018.

[22] OPNFV, "Open platform for NFV," https://www.opnfv.org/, April 2018.

[23] F. Lucrezia, G. Marchetto, F. Risso, and V. Vercellone, "Introducing network-aware scheduling capabilities in openstack," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–5.