EnforSDN: Network Policies Enforcement with SDN

Yaniv Ben-Itzhak*, Katherine Barabash*, Rami Cohen†, Anna Levin*, Eran Raichstein*
*IBM Research Lab, Haifa, Israel
{yanivb, kathy, lanna, eranra}@il.ibm.com
†Empow Networks
ramic@empownetworks.com

***Abstract*—Network services, such as security, load-balancing, and monitoring, are an indisputable part of modern networking infrastructure and are traditionally realized as specialized appliances or middleboxes. Middleboxes complicate the management, the deployment, and the operations of the entire network. Moreover, they induce network performance issues and scalability limitations by requiring huge amounts of traffic to be, often sub-optimally redirected, and sometimes redundantly processed. Recent trends of server virtualization and Network Function Virtualization (NFV) exacerbate these scalability and performance issues. In this paper, we present *EnforSDN* - a new management approach that exploits SDN principles to decouple the *policy resolution* layer from the *policy enforcement* layer in network service appliances. Our approach improves the enforcement management, network utilization and communication latency, without compromising the policy and the functionality of the network. Using emulated SDN-based data center environment, we demonstrate higher throughput and lower latency achieved with *EnforSDN*, as compared to a baseline SDN network. In addition, we show that *EnforSDN* reduces the overall network appliances load, as well as the forwarding tables size.**

Keywords*—*Software-Defined Networks, Network Function Virtualization, Middleboxes

I. Introduction

Modern computer networks are expected to provide not only fast and efficient connectivity between communicating nodes, but also an increasingly large and complex set of enhanced network services, such as security, load balancing, monitoring, acceleration, and many more. Traditionally, these enhanced services are delivered over the network by dedicated appliances, also called middleboxes. For example, common security appliances are firewalls (FW), intrusion detection and prevention systems (IDS and IPS); another example is load balancing which performed by Application Delivery Controllers (ADC). In most cases, every packet of a communication flow[1] has to pass through the middlebox in order to get service. Management schemes, required to selectively push packets through middleboxes, are usually complex, cause suboptimal routes, and often break otherwise well architected routing systems. Moreover, such schemes are mostly static, not scalable, cause operational headaches, and overload network links surrounding the middleboxes [1], [2], [3].

Recent IT trends exacerbate these problems. First, ever growing demand for increased throughput and predictably low communication latency require distributing the load between redundant instances of the same appliance. Second, server virtualization and scale-out consolidation call for sharing the same IT infrastructure, including the network services infrastructure, by multiple independent tenants. Third, modern workloads are often virtualized and are highly mobile and dynamic, further requiring the infrastructure, and the network services, to be flexible and adaptable.

In order to address these and additional challenges, vendors began virtualizing network service appliances and the Network Function Virtualization (NFV) concept was conceived. While NFV, especially in combination with SDN, has great potential of simplifying the management and reducing the acquisition and the operation costs of large scale service-rich infrastructures, it does not question the basic management premise of pushing all the serviced data through the service appliances.

In this paper, we challenge the aforementioned management premise, basing on the observation that network service appliance consists of three distinct logical layers, each with its responsibilities and concerns. The first layer is responsible for policy *configuration* based on a high level policy description. The second layer is responsible for policy *resolution* using concrete policy rules derived from the policy *configuration* and applied to the communication flows. The third layer is responsible for policy *enforcement* using low level data plane instructions applied to each and every packet of the flow.

Figure 1 presents the distributed approach which is based on SDN concept of separation of the control plane from the data plane. In this approach, the control decisions (i.e., policy *configuration*) are made by a logically centralized entity based on global knowledge and delivered as simplistic per-flow instructions into the forwarding tables of programmable switches, which execute the policy *resolution* and *enforcement*.

However, as bandwidth requirements increase, this solution poses additional challenges, e.g. in state synchronization and management. In particular, these challenges are manifested in virtualized environments, where the virtual appliances and application components instances are dynamically created, migrated, and terminated.

In the sequel, we exemplify some of the performance challenges imposed by the middleboxes deployment using the distributed approach (depicted by Figure 1).
Latency: According to [2], introducing a new service appliance into the network adds latency to the traffic propagated over the network. The sources of middlebox-induced latency are: the added processing time at the service appliance, the additional network propagation delay on the path towards the middlebox, and the queuing delay in the more loaded forwarding devices.

[1] In this paper we use 'flow' and 'session' interchangeably, to denote the communication flow/session between the interconnected endpoints.

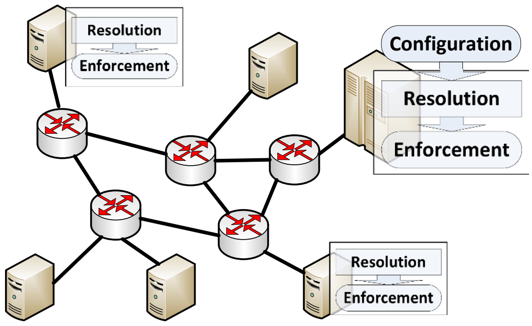


Fig. 1. Distributed approach: multiple service instance share common configuration rules

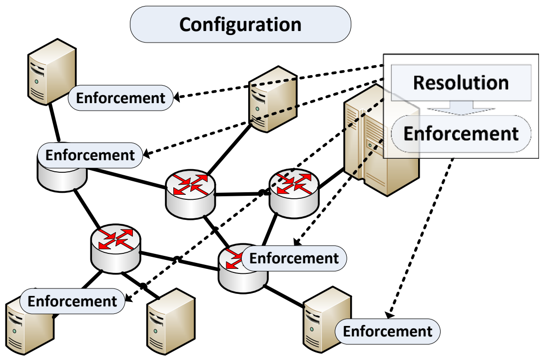


Fig. 2. *EnforSDN* approach: policy *resolution* is decoupled from the policy *enforcement*

Appliance Overload: Network appliances overload is also a well-known problem [3]. According to [4], roughly 16% of enterprise data center administrators cite overload as the most common cause of middlebox failures. The overloaded appliance becomes a network bottleneck, introducing unnecessary delay, packet loss, and even threatening network connectivity. As reported in [2], over 40% of connectivity loss issues in service providers networks were caused by middleboxes.

Network Overload: In many cases middleboxes placement is dictated by the operational, compliance, or business structure requirements. Thus, forwarding traffic towards middleboxes induces unnecessary network overload and bandwidth waste. The waste of bandwidth is greater when traffic is further redirected by middleboxes along service chains. The resulting non-optimal routing paths between service chain participants overload the network, increase the latency of the flows, and complicate the routing in the network. The problem becomes even more noticeable when middleboxes are moved to the cloud, as suggested in [4], due to redirection bandwidth cost.

In this paper, we present *EnforSDN* which decouples the policy *resolution* layer of network service appliances from the policy *enforcement* layer, centralizing the former, and greatly simplifying the latter. Figure 2 outlines this decoupling principle whereby the policy *enforcement* layer of the network service is separated from its policy *resolution* layer. As a result, a small set of policy *resolution* instances control and manage a potentially large number of distributed, simple and programmable policy *enforcement* instances, by SDN.

We further observe that in SDN-based networks, the role of policy *resolution* instances can remain with the network appliances, while the role of policy *enforcement* can be trusted to the programmable SDN forwarding devices, e.g. software or hardware SDN switches managed by, for example, OpenFlow protocol. Applicable to a significant subset of network appliances , our approach enables to dynamically and remotely control the flow tables of network forwarding devices and to dynamically steer the data forwarding paths such that data either passes through the appliance or is sent directly to the destination as prescribed by the policy decision.

EnforSDN targets the aforementioned drawbacks of the distributed approach. In particular, *EnforSDN* benefits are:
1) Reduction of the communication *latency* due to redundant processing elimination and more efficient routing paths.
2) Reduction of the *appliances load*.
3) Reduction of the *network load* over the links surrounding the appliances.

The rest of the paper is structured as follows. In section II, we describe the *EnforSDN* architecture, focusing on firewalls, which are well known to be the most widespread network appliances in enterprise networks [3], [4], [5], and usually serving as a first element in a service appliances chain, redirecting flows to a set of other appliances (DPI, ADC, etc). Furthermore, firewall operations are straightforward and well known; therefore, making it a good tool for presenting *EnforSDN* approach. In section III, we evaluate *EnforSDN* and demonstrate its benefits, which result in improved network performance, scalability, and utilization. In Section IV, we discuss advanced concepts and limitation of the solution. In Section V, we survey the related work, and in Section VI we conclude the paper, as well as outline some future work directions.

II. *EnforSDN* Architecture

EnforSDN is a novel approach for integrating network service appliances, e.g. FWs, IPSs, IDSs, into SDN environments. Typically, Software Defined Network (SDN) consists of programmable forwarding devices, the controller, and the network appliances, for providing advanced connectivity services. Network appliances are usually responsible to process both the policy *resolution* and *enforcement* decisions of their ingress flows (i.e., distributed approach), inducing the limitations and shortcomings, as described in section I.

With *EnforSDN*, the policy *resolution* and the policy *enforcement* responsibilities are decoupled, reducing both the management overhead and performance degradation caused by monolithic service appliances. While, similarly to traditional middleboxes, *EnforSDN*-enabled network appliance implements both the policy *resolution* and *enforcement* steps, it can choose to delegate some of *enforcement* processing to programmable forwarding devices across the network.

To enable this kind of delegation, we exploit the SDN's communication channel between the controller and the forwarding devices. As depicted in Figure 3, an *EnforSDN* enabled appliance consists of one or more policy *resolution* instances, and each such instance is connected to the SDN controller by an *EnforSDN manager* that can be deployed as SDN application on top of the controller. Upon request from the *EnforSDN manager*, SDN controller can delegate *enforcement* decisions of different policy *resolution* instances, to the forwarding devices in the network.

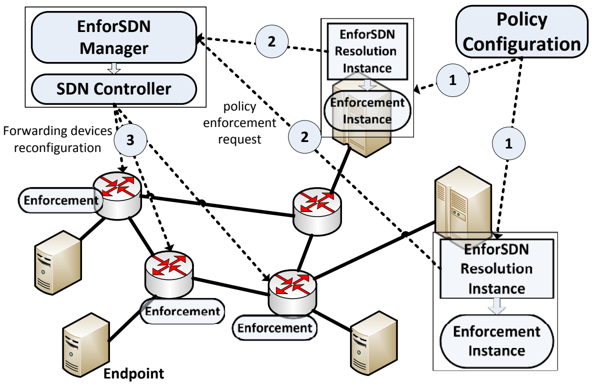


Fig. 3. The *EnforSDN* control plane flow

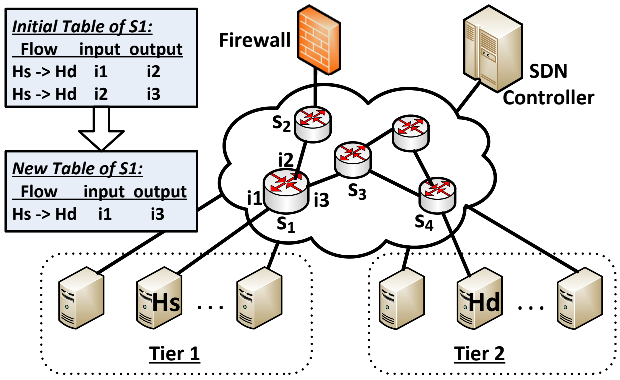


Fig. 4. *EnforSDN* end-to-end example

Let's describe the control plane flow for *EnforSDN* architecture. Based on the policy *configuration* (step 1 in Figure 3), upon sending a flow from one endpoint to another, the flow must be passed through an ordered set of *resolution* instances of different appliances. In this case, for each network appliance, an external manager is responsible to determine which policy *resolution* instance should handle the flow. Note that, the policy *resolution* instance is collocated with the policy *enforcement* instance at the network appliances. The SDN controller configures the physical network infrastructure, e.g. with OpenFlow, such that the flow is routed through the appropriate *resolution* instance of the appliance. As mentioned above, in order to enable the decoupling of the policy *resolution* from the policy *enforcement*, each policy *resolution* instance is connected to the SDN controller, through the *EnforSDN manager*. Through this communication channel, the appliance can notify the *EnforSDN manager* about its decision, requesting the *EnforSDN manager* to take care of enforcing the decision; the decision can be, for example, blocking, modifying, logging, steering, redirecting, or rate limiting the flow.

Based on the request (step 2 in Figure 3), the *EnforSDN manager* (using the SDN controller) configures one or more switches (step 3 in Figure 3) to enforce the policy. This may include blocking, dropping, logging or modifying the flow by one of the switches along the flow's route path (e.g. by adding ACL rules), sending the flow directly to the destination (e.g. by configuring the flow tables of the switches), steering the flow, or a combination of more than one actions (e.g. modifying and steering). Such a configuration, which may be time bounded, ensures that while the policy is enforced, the flow doesn't load the network appliance instances and then it is routed in more efficient way.

Note that it is up to the appliance, or the service, to decide which policy to enforce locally and which policy should be remotely enforced, based on a predefined criteria or based on the flow plan capabilities. Moreover, in order to handle packets that have already been sent, before the new configuration takes place in the network, the *enforcement* instance of the network appliance keeps enforcing the policy.

The exact API between the policy *resolution* instance and the *EnforSDN manager*, depends on the appliance functionality and the infrastructure capabilities. Information exchanged in *EnforSDN* API must identify the flow and describe the action to be done with associated packets, but must not involve information regarding the network topology and its current state, which should be handled by the *EnforSDN manager* and the SDN controller. Thus, upon receiving a policy *enforcement* request, the *EnforSDN manager* is responsible to retrieve the necessary network related information, to configure the network, and to track the configuration steps in order to be able to shift to former states. This can be done either by direct interaction with the SDN controller and/or the SDN switches, or by using other SDN applications such as topology discovery, etc. Also, each policy *enforcement* request may contain an expiration time-out whereby the policy *resolution* can request to re-inspect the flow after a period of time.

Firewalls are well known to be the most widespread network appliances in enterprise networks [3], [4], [5]. Hence, in the following, we use firewall as a network appliance use-case, to demonstrate and evaluate *EnforSDN*.

End-to-End Example:

Let us demonstrate by example the way that *EnforSDN* works. In the network fragment presented in Figure 4, there are two tiers of communicating hosts. The switching fabric includes, switches S_1 to S_5. The SDN controller configures the switches according to the appropriate network policies (*configuration* phase). The figure shows a fragment of deployed multi-tier application, with network policy configuration which forwards traffic between Tier 1 and Tier 2 through the Firewall. Consequently, when the sender host H_s from Tier 1 sends the first packet towards the destination host H_d in Tier 2, the SDN controller configures forwarding devices with corresponding rules. For example, the initial table of the switch S_1 presented in Figure 4 is configured as follows:
Traffic from H_s to H_d goes towards Firewall via S_2, i.e. input on interface **i1** is forwarded to the interface **i2**.
Traffic from Firewall to H_d goes towards S_3, i.e. input on interface **i2** is forwarded to the interface **i3**.

Without *EnforSDN*, this same configuration remains active for all the packets of the flow, introducing additional *latency* to the flow, as well as *network overload* and *appliance overload*, as mentioned in Section I. However, with *EnforSDN*, the configuration changes as soon as the Firewall approves the flow, based on the first packet (*resolution* phase). The Firewall sends notification to the SDN controller, that the flow is legitimate and can be sent directly to the destination. The SDN controller in turn recalculates configuration of the forwarding devices and sends them new rules (*enforcement* phase). The new forwarding table of the switch S_1 presented in Figure 4 is configured in a following way:

Traffic from H_s to H_d goes towards S_3, i.e. input on interface **i1** is forwarded to the interface **i3**.

Our approach reconfigures the forwarding tables redirecting the flow to bypass the Firewall and go over the optimal path. Hence, redirecting the flow reduces the *latency* and *network overload* due to shorter and better routing path. In the case that the Firewall decides to drop the flow, *EnforSDN* would enforce the Firewall decision by dropping the flow's packets at the nearest forwarding device to H_s (i.e., S_1), which in turn reduces the *network overload*. Furthermore, the Firewall needs to inspect only the first packets of the flows and report to the SDN controller its decision, thus reducing significantly the *appliance overload* presented by this flow. Lastly, the size of the forwarding table is reduced , as we will discuss later in Section III.

III. Evaluation and Results

In order to evaluate *EnforSDN*, we have chosen a scenario similar to the depicted end-to-end example in section II, where stateless firewall service is required to inspect and filter traffic flows between the tiers of typical multi-tier workloads. To that end, we emulate an OpenFlow-based network with Mininet [6] running over an IBM System x3850 X5 server equipped with 3TB of RAM and eight Xeon-E7@2.13GHz CPUs (with eight cores each).

Mininet is a network emulator which creates SDN networks of virtual hosts, switches, controllers, and links for any given topology. Mininet networks run real Linux kernel and network stack, which provides correct system behavior and performance. Due to its benefits, Mininet becomes increasingly common in both academic and industry for preliminary evaluation and testbed purposes.

We have extended Mininet with modules for parametrized creation of multi-level data-center topologies, for traffic generation, and for firewall emulation, as well as with the *EnforSDN* extensions to the controller and the firewall. For the kernel-level software switch we use Open vSwitch 2.0.90 [7], an open-source multilayer virtual switch that supports OpenFlow protocol. All the Python code created for the evaluation can be obtained from [8].

The rest of this section describes the details of our implementation in §III-A and the three experimental settings in §III-B, followed by the presentation and analysis of the *EnforSDN* impact on overall network performance in §III-C, on load experienced by the firewall in §III-D, and on forwarding table sizes in §III-E.

A. Solution Components

Data-Center Network Topology: We emulate multi level switched network topology typical to modern Data Center Networks (DCNs). Our Mininet extension module allows creating three-level Fat-Tree topologies [9] for various sizes and employs D-mod-K routing algorithm [10], [11], [12], the most common routing for fat-tree-based DCNs[2]. In our tests, due to Mininet limitation, we scale-down the links' bandwidth, such that, each host is connected through a 1Mbit/sec link to an edge switch, and switches are connected through a 10Mbit/sec link, maintaining equal bisection bandwidth, as in typical data center deployments.

[2]For the sake of brevity, we focus on Fat-Tree topology which is the most common topology for data-centers.

Data-Center Workload: In each emulated Data Center application, the application components reside in two tiers, modeling the web-servers tier and the databases tier, such that each host from the first tier communicates with a host from the second tier. According to the data-center traffic measurements presented in [13], we emulate a real data center traffic with random distribution of sessions between the query traffic (2KB to 32KB in size), the short messages (100KB to 1MB), and the long flows (1MB to 100MB), as well as between the UDP and the TCP sessions. To emulate the firewall policy configuration, we assign each communication session with a random security rule of *process*, *accept* or *deny* that must be applied to the session's traffic. In the following section (III-B), we describe the experimental settings used for our evaluation, and the way each security rule is handled in each setting.

***EnforSDN*-enabled Firewall:** In order to implement the firewall, we use iptables and Netfilter queue [14]. For the policy *configuration* step, the firewall is configured out of band according to the sessions' assigned security rules. Upon configuration, each security rule is attached to a different Nfqueue and processed according to the firewall's configured operation mode. For the sake of *EnforSDN* evaluation, the firewall can be configured in two different operation modes: the regular uncooperative firewall mode and the *EnforSDN* mode where firewall cooperates with the SDN controller as described in Section II.

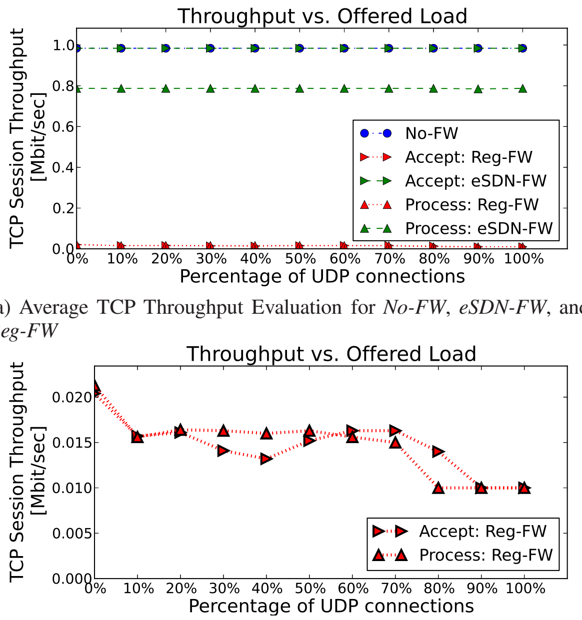
***EnforSDN* Controller:** We have implemented an extended SDN controller which supports *EnforSDN*. The extended controller is capable of deploying both the initial OpenFlow rules whereby all flows are being routed through the firewall, and additional OpenFlow rules computed based on dynamic decisions made by the *EnforSDN* enabled firewall.

B. Experimental Settings

We configure the solution components described above to run three different experimental settings:

Reg-FW: This setting is used to emulate operations of the regular uncooperative firewall deployed in the SDN-based network. Here, SDN controller is used in its regular mode for deploying OpenFlow rules forcing all the traffic through the firewall where both the policy *resolution* and the policy *enforcement* are taking place. Sessions which are configured with *deny* rule are discarded at the firewall, while sessions with *accept* (*process*) rules are forwarded (processed and forwarded) by the firewall towards their destination.

eSDN-FW: This setting is used to emulate operations of the *EnforSDN*-enabled firewall deployed in the *EnforSDN* environment. Here, both the SDN controller and the firewall are used in the *EnforSDN* mode. At flow initiation, *EnforSDN* controller deploys OpenFlow rules forcing all the traffic through the firewall. In addition, each time *EnforSDN* firewall makes policy *resolution* decision for a flow, it communicates the decision to the *EnforSDN* controller which in turn computes new OpenFlow rules and deploys them in forwarding devices where the policy *enforcement* step will now take place for the rest of the flow's packets.



(a) Average TCP Throughput Evaluation for *No-FW*, *eSDN-FW*, and *Reg-FW*

(b) Zoom in to the *Reg-FW* results in (a)

Fig. 5. Average TCP Throughput Comparison for Fat-Tree Data-Center with 32 hosts

For flows with *deny* decision, the *EnforSDN* controller sets rules to drop the packets into the forwarding device nearest to the flow's source host. For flows with the *accept* decision, the *EnforSDN* controller sets rules to directly route the flow's packets to their next destination, bypassing the firewall. As flows with the *process* decision must be processed by the firewall, no action is required on behalf of the *EnforSDN* controller and there is no need to communicate the decision to it in the first place.

No-FW: This setting is used in order to obtain the upper performance bound, for the sake of comparison. Here, no firewall is deployed; the sessions' security rules are enforced, such that *deny* packets are dropped nearest to the flow's source host, while *accept* and *process* sessions are routed directly to their next destination. Therefore, all the firewall-induced performance penalties are avoided (both in terms of computation and network).

C. Throughput and Latency Evaluation

We evaluate network performance by measuring the throughput and round-trip-time (RTT) experienced by communication sessions of different types, namely by sessions with the assigned policy configuration of *allow*, *process*, and *deny*, for each one of the three experimental settings described in section III-B. To represent realistic data center network situation, we emulate a given offered network load of the real data-center traffic. To that end, we define a set percentage of UDP sessions out of the overall data-center network traffic (composed of the query, the short, and long flows as described in section III-A). As opposed to TCP sessions, UDP sessions are network-load insensitive, therefore, higher percentage of UDP sessions results in data-center network traffic with higher offered network load. For any given offered load, varying from 0 to 100 percents in 10 percents increments, we measure the average throughput and RTT of *accept* and *process* TCP sessions for *Reg-FW*, *eSDN-FW*, and *No-FW* settings.

Figure 5 presents the average achieved throughput of TCP sessions in the data-center, as reported by *iperf*, configured with either *accept* or *process* security rules. Our emulation demonstrates that by re-routing *accept* sessions directly towards their destinations, *eSDN-FW* achieves the maximum possible throughput available to hosts (~1Mbit/sec, the link bandwidth between the host and its edge switch), also achieved by the ideal case represented by the *No-FW* results.

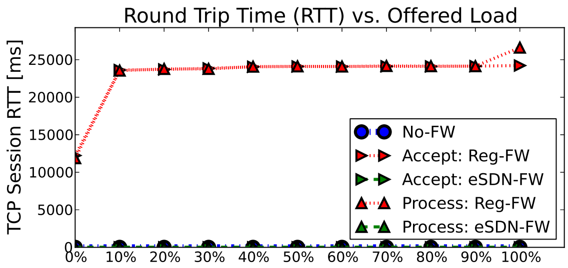
It is well-known that many-to-one communication patterns result in major network performance degradation due to TCP throughput collapse (and increased RTT), also known as Incast [15]. In particular, Fat-Tree topology provides a single route from a core switch (top-level) towards any given host, exacerbating the Incast challenge. As opposed to *Reg-FW*, which pushes all the sessions towards the firewall, in a case of *eSDN-FW* only the *process* sessions are forwarded towards the firewall. Therefore, *EnforSDN* benefits the system in two aspects: (1) reducing the firewall load (see section III-D for more details), which in-turn (2) reduces the load of the links towards the firewall, thus easing the Incast problem. All the benefits combined enable *EnforSDN* to achieve 400%-500% throughput improvement, as compared to *Reg-FW*. Furthermore, as can be seen in Figure 5(a), the *eSDN-FW* shows stable good results independently of the offered load of the data-center traffic. Figure 5(b) demonstrates that the throughput achieved by *Reg-FW* decreases as the offered load increases, due to amplified Incast problem caused by higher load over the links towards the firewall. Moreover, *process* sessions experience relatively small reduction (up to 20%) in the *eSDN-FW* setting as compared to the ideal case of the *No-FW* setting. Overall, *EnforSDN* offers consistently better performance, performance isolation, and stability in terms of the total throughput allowed by a network of a specified size.

Figure 6 presents the average round-trip-times (RTT), reported by *ping*[3], experienced by sessions configured with either *accept* or *process* security rules. It can be seen that *Reg-FW* results in very high RTT values, increasing with the offered load. The RTT is influenced by the amount of hops the packet goes through on it way to the destination, the queuing delay imposed at each hop (including the firewall, which contributes most of the RTT for the *Reg-FW* case), the processing delay imposed by the firewall, and the Incast problem. On the other hand, *No-FW* and *eSDN-FW* demonstrate consistently low RTT values as can be seen in Figure 6(b). The RTT values of *accept* sessions under *eSDN-FW* increase at the presence of UDP sessions, and stay stable thereafter as the offered load increases. The *process* sessions under *eSDN-FW* experience a bit higher RTT values[4]. We attribute this to the manifestation of the Incast problem, which is much milder here than in *Reg-FW* case, because less sessions are directed towards the firewall.

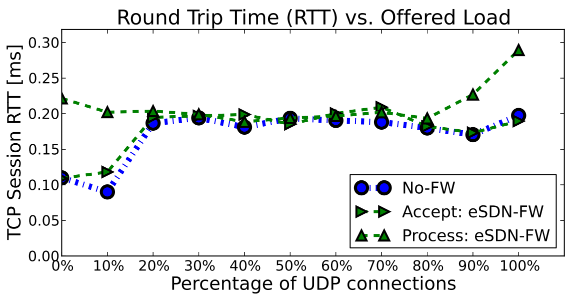
In summary, *EnforSDN* offers better overall network performance in terms of both throughput and RTT, as compared to *Reg-FW*, through reducing the network links' load on the path towards the firewall and relaxing the Incast problem.

[3]We measure RTT by frequent pings along the TCP sessions.

[4]Notice that the results in this case are partly blurred, since the pings (ICMP packets) are transmitted by the firewall without incurring overhead.



(a) RTT experienced by *accept* and *process* sessions in *No-FW*,*eSDN-FW*, and *Reg-FW* settings



(b) Zoom in to the *No-FW* and *eSDN-FW* RTT results

Fig. 6. Round-Trip-Time(RTT) Comparison for Fat-Tree Data-Center with 32 hosts

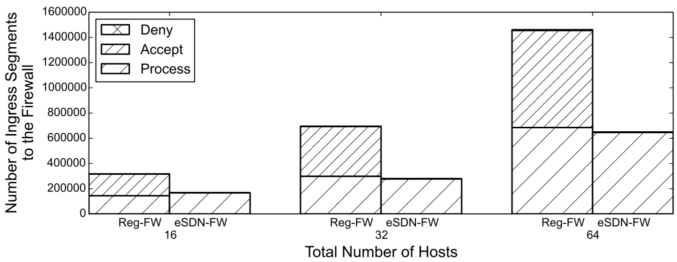


Fig. 7. Number of *process*, *accept* and *deny* segments hitting the firewall, for varying Fat-Tree sizes (Table I details the number of *deny* segments)

D. Firewall Load

We evaluate the load experienced by the firewall by counting the number of segments[5] ingressing the firewall, as part of either the policy *resolution* or the policy *enforcement* steps. Obviously, as the number of ingress segments hitting the firewall increases, so does the firewall load; therefore, reducing the number of ingress segments results in a better overall firewall performance and lower latency experienced by the communication sessions. We compare our measurements for the *Reg-FW* and the *eSDN-FW* settings, omitting the *No-FW* since it does not include firewall.

Figure 7 presents the total ingress segments for both *Reg-FW* and *eSDN-FW*. The measurements is obtained for the representative data-center traffic matrix consisting of 90% of TCP sessions and 10% of UDP sessions, as measured in [13]. It demonstrates that in *eSDN-FW*, the total number of segments ingressing the firewall is reduced as compared to *Reg-FW*, mostly through the elimination of *accept* segments. In *eSDN-FW accept* sessions are rerouted directly towards their destination, and only the first segments of these sessions hit the firewall (*resolution* step), while the rest of the segments avoid firewall processing, reducing the firewall load.

[5]In this paper, we use the terms 'packets' and 'segments' interchangeably.

TABLE I. NUMBER OF INGRESS DENY SEGMENTS TO THE FIREWALL, FOR VARYING FAT-TREE SIZES

Firewall Settings	Number of Hosts		
	16	32	64
Reg-FW	600	2400	6000
eSDN-FW	1	4	10

In the same manner, *EnforSDN* also reduces the number of *deny* segments (See Table I). In *Reg-FW*, the firewall drops the SYN segments of *deny* TCP sessions, which results in SYN retransmissions (every initial RTO of 1sec). In *eSDN-FW*, the firewall drops only the first SYN segment transmission of each *deny* TCP session. Although, *EnforSDN* reduces the number of *deny* segments by 99.9%, one might argue that this is a negligible contribution to the overall load of the firewall since the number of *deny* segments (SYN segments) is relatively low. However, in cases of DDoS attacks, and in particular SYN flood attacks, *EnforSDN* can demonstrate better security capabilities as compared to *Reg-FW*. To summarize, in our setup, *EnforSDN* reduces 50%-60% of total ingress segments to the firewall, as compared to *Reg-FW*.

For virtualized appliances, reducing the appliance load is even more critical, as compared to the physical network appliance case. As most NFV deployments are bound by I/O, *EnforSDN* enables better utilization of NFVs over a given physical server and allows more flexibility in consolidation of NFVs, as suggested in [3].

E. Number of Required Open-Flow Rules

In this section, we measure the total number of Open-Flow rules deployed in the network during the policy *enforcement* phase, namely at steady-state, after the policy *configuration* of all sessions have been resolved through the policy *resolution* phase. Figure 8 presents comparison of the total number of deployed Open-Flow rules for each session type, namely *allow*, *process*, and *deny*, for different Fat-Tree sizes, between the *Reg-FW*, *eSDN-FW*, and *No-FW* settings. Notice that through this comparison, we don't apply any flow aggregation method.

The number of deployed Open-Flow rules for sessions with *accept* rules is significantly lower for *eSDN-FW* and *No-FW*, as compared to *Reg-FW*. In *Reg-FW*, the *accept* sessions are required to be routed first through the firewall, and only then towards their destination. The resulting forwarding path is typically much longer path bypassing the firewall. In particular, in Fat-Tree based data-centers, paths between hosts connected to different aggregation switches are typically longer, being routed first to a core switch (top-level) and then downwards the required destination [9], [12]. Hence, directing a flow through a firewall (or any intermediate host) in Fat-Tree topology can result in approximately doubling the routing path length. On the other hand, in the *eSDN-FW* setting, most of the packets in each *accept* session are forwarded on a direct path, like in an ideal *No-FW* case; only the first packets of these sessions, sent before the policy *resolution*, experience the longer path and the firewall processing delay. As longer paths obviously pass through more forwarding devices, the number of required Open-Flow rules increases. For instance, as can be seen in Figure 8, the total number of Open-Flow rules for *accept*

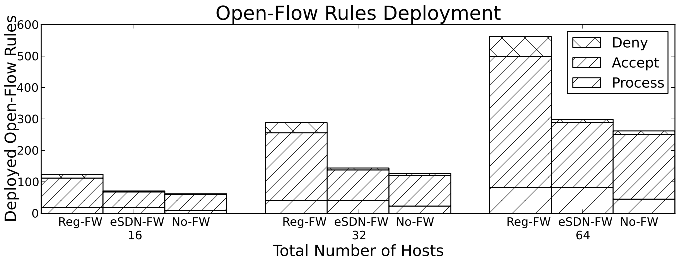


Fig. 8. Total number of Open-Flow rules deployed for *process*, *accept* and *deny* sessions at steady state, for varying Fat-Tree sizes

sessions in 64-hosts network is 408 for *Reg-FW* setting and 207 for both the *eSDN-FW* and the *No-FW* settings; i.e., reduction of approximately 50%.

The ratio between the number of deployed Open-Flow rules in *Reg-FW* setting and in the *eSDN-FW* setting is higher for *deny* sessions than for *accept* sessions. This is because the SDN controller is not aware of the sessions' security rules in *Reg-FW*. Therefore, despite the fact that *deny* sessions are dropped at the firewall, the SDN controller is still required to configure a full routing path for these sessions, through the firewall towards their destination.

On the other hand, in *eSDN-FW* and *No-FW*, the SDN controller is aware of each session's security rule after it is resolved through the *resolution* step. Hence, it is required to deploy only one Open-Flow rule for each *deny* session, into the nearest forwarding-device to the source host. For instance, Figure 8 shows that for a 64-hosts network the number of Open-Flow rules for *deny* sessions is 72 in *Reg-FW* setting, and only 11 in *eSDN-FW* and *No-FW* settings, resulting in reduction of 85%. For *process* sessions, *Reg-FW* and *eSDN-FW* require the same number of Open-Flow rules to be deployed in the network, twice as much as the number required by the idealistic setting. Note that, the *No-FW* result is unachievable in the presence of network middlebox.

On the other hand, *EnforSDN* probably reduces the flow aggregation potential . Therefore, *EnforSDN* introduces trade-off between the amount of flows that enjoy *EnforSDN*-optimized forwarding and the Open-Flow rules footprint over the flow tables. In order to cope with this issue, *EnforSDN* manager can decide whether to employ *EnforSDN* according to the required flow rules footprint and current flow-tables occupancy.

IV. Discussion

This work is a starting point on a way towards full-fledged implementation of *EnforSDN*. In this section, we highlight several advanced architectural and operational aspects.

Flow Initialization: With *EnforSDN*, once the flow policy is resolved and the appliance has made its decision regarding the policy *enforcement*, flow's path in the network is recomputed and new OpenFlow rules are deployed into the flow tables of the forwarding devices along the new flow's path. This means that initial OpenFlow rules for the flow are no longer relevant. In forwarding devices that remain on flow's path, initial rules are overridden with the new rules computed based on the appliance decision, while stale rules might remain in forwarding devices that are no longer on the flow's path. We suggest that initial rules of new flows receive relatively lower priority and shorter timeout than the rules introduced after the service policy decisions were made. It guarantees that irrelevant rules for the *enforcement* phase are timely invalidated and don't consume OpenFlow table space [16].

Middlebox Feedback Loop: It might seem that *EnforSDN* compromises the network security by rerouting *accept* sessions directly to their destinations. Since the session is redirected to bypass the middlebox, the firewall won't be able to inspect the entire flow and determine whether it's a legal or malicious flow, which masqueraded for legitimate flow during the beginning of the flow. To overcome this, we plan to implement a sampling method whereby flow's packets are periodically sent through the middlebox, so it can re-inspect the flow. One solution is to define a timeout for OpenFlow rules deployed for the policy *enforcement* phase, and the middlebox defines the action to be performed upon rule's expiration. When OpenFlow rule expires, the OpenFlow switch communicates to the SDN controller, which will perform one of the following actions: (1) Redirect the flow through the middlebox so it can inspect the flow and make new decision, which will be deployed by *EnforSDN* manager. (2) Duplicate sub-set of the flow packets, and send packet copies to the middlebox (e.g., by [17]). As opposed to the previous option, this solution doesn't affect the flow, since its original routing is preserved.

NFV Migration: One of the advantages of NFV is the ability to migrate appliances based on the current network, computing requirements, and appliance's states and rules. However, it is also required to invalidate OpenFlow rules which route flows towards the old appliance location, and deploy new OpenFlow rules to route the flows towards its new location. Clearly, the more flows are handled by the network appliance, the more appliance's states/rules and OpenFlow rules have to be moved or invalidated and deployed. *EnforSDN*-enabled appliances are capable of communicating with the network controller; hence, handling a significant lower number of packets and are less loaded (see section III-D), compared to regular non-cooperative appliance. Therefore, *EnforSDN* also has the potential to reduce the migration cost.

Denial of Service Attacks: *EnforSDN* scales-out the *enforcement* capability of the network appliances and enables the network to withstand Denial of Service (DoS) by exploiting any proper SDN-based forwarding device to drop DoS packets as close to the network perimeter. For example, in our evaluation (section III-D) of the *EnforSDN*-enabled firewall, we demonstrate that *EnforSDN* dramatically reduces the number of *deny* segments hitting the firewall. Thus, *EnforSDN* is potentially more immune to DoS attacks, e.g. SYN flood attacks, than the regular uncooperative firewall.

Stateful Appliances: *EnforSDN* benefits are derived from decoupling the policy *resolution* and the policy *enforcement* steps, entrusting different steps to different entities. Therefore, once the *enforcement* step of a given flow is delegated to a remote *enforcement* instance, the middlebox loses the ability to keep track of the flow's state. Hence, *EnforSDN* mainly target stateless network appliances; however, we argue that stateful appliances deployments, in some cases, can also benefit from *EnforSDN* implementation. First, pre-processing by a stateless service can filter the traffic and reduce the number of flows to be handled by stateful service; hence, reducing the stateful appliance's load and utilizing the rest of *EnforSDN* benefits, as described in [18]. Second, Amazon has presented semi-stateful firewall [19], which applies new security rules only for new inbound sessions, while keeps enforcing any in-progress sessions with the older security rules. In such case, *EnforSDN* can request the *EnforSDN* manager to capture only new inbound sessions and route them through the appliance for *resolution*, while directing the existing sessions according to the already resolved rules. Third, any stateful appliance capable of periodic flow state sampling, can employ *EnforSDN* using one of the *Firewall Feedback Loop* methods described above. Fourth, the next version of open vSwitch (OVS 2.4) will support stateful flow rules [20]. It will allow *EnforSDN* to enforce staeful rules over the forwarding devices.

V. Related Work

Our work refers to several topics covered by recent research and industry contributions. In the following, we briefly cover this landscape and describe the most relevant related work.

First, **middlebox** notion have historically been controversial [21] and, as such, have attracted a lot of research and industry attention. To overcome middleboxes-induced issues, researchers suggested to redesign or to distribute the middleboxes [22], [23], [16], to eliminate middleboxes by placing their functions elsewhere [24], to re-architect the network to better accommodate the middleboxes [1], [25], [26], to automate middlebox placement and to steer traffic through them efficiently [27], and even to outsource the middlebox-provided services [4], [28], [29]. Second, **Software Defined Networking (SDN)**, enables novel middlebox-related solutions, e.g. transmitting traffic through middleboxes placed in flexible locations [30], [31] or incorporating some of the middleboxs' functionality in the SDN controller itself. Third, **Network Function Virtualization (NFV)** [32], has surfaced new research questions related to multiplicity of middleboxes, their flexible placement and migration, load balancing, etc. Moreover, NFV operations can be greatly facilitated by the SDN for many types of deployments and use cases [30].

In this work, we confront the fundamental question of enforcing the network functions and policies in modern virtualized environments. We call neither for evicting or outsourcing the middleboxes [4], [28], [29] nor to turning them all to virtual entities [3], [5]. Instead, we stand for treating middleboxes, both virtual and physical, as required first-class entities. Our approach builds upon the SDN paradigm, extending it with the well defined separation of concerns between the service policy *resolution* and the service policy *enforcement*, orchestrated by a centralized control plane intelligence.

Recent academic works most closely related to ours are FlowTags [33], Stratos [30], and FlowGuard [34]. **FlowTags** architecture is one of the first academic attempts in incorporating the middleboxes into the realm of SDN by extending the middleboxes with the ability to mark the data packets with information to be processed by the control or the data plane entities. Although very useful, this approach affects the data packets and thus requires network wide changes, while the extensions required by *EnforSDN* are only to the control APIs. **Stratos** orchestration framework takes care of efficient provisioning and composition of the network services chains and as such is complimentary to *EnforSDN*; actually, building a new orchestration framework or integrating with the already proposed one, e.g. Stratos, is certainly one of the next steps in our work. **FlowGuard** solution is tasked with resolving the global policy violations in SDN environments and eliminating possible conflicts between the firewall policies and the SDN policies. Similarly to our work, FlowGuard's policy resolution framework can require flow eviction and adding packet dropping flows into the SDN framework in order to maintain consistent global behavior. In our approach, the focus is on separation of duties between the SDN controller and the appliances as well as on communication between them. Several control and management platform have been presented (e.g, Frenetic [35], and OpenNF [36]). **Frenetic** provides a control platform and functional-reactive language for programming Open-Flow networks, in particular allowing to implement firewalls. **OpenNF** provides network appliance state management, which allows efficient reallocation of flows across network appliances instances. OpenNF also provides southbound API for network appliances that allows a controller to request the export or import of network appliance state without changing how network appliances internally manage state. On the other hand, *EnforSDN* focuses on conceptual model of separating the policy *resolution* and the policy *enforcement* phases, a model that allows separation of concerns between the controller and the appliances. Therefore, Frenetic and OpenNF can serve as a platform for *EnforSDN* implementation.

Industrial approach most similar to ours is that of vArmour [37] where the SDN controller interacts with the SDN agents in forwarding devices and in firewall appliances to enforce network-wide access control and security policies. While very interesting and innovative, vArmour's approach is closed, proprietary and tailored to a specific business application they sell. *EnforSDN* was conceived and developed independently and focuses on separation of duties between the SDN controller, the SDN switches and the SDN-enabled middleboxes, and on creating well defined interfaces between them.

VI. Conclusions and Future Work

We have presented *EnforSDN*, a novel approach for implementing large and important subset of network services in SDN-based networks. This paper starts with surveying the middlebox-induced problems, goes on to outlining the underlying solution concept, to describing the solution architecture, and demonstrating the evaluation results that confirm the performance and the network utilization improvements.

We have exemplified the *EnforSDN* approach for single-instance deployment of firewall. Additional work is required to extend *EnforSDN* to support other types of appliances that can be enabled to cooperate beneficially with the network forwarding control, as described in Section II. The evaluation presented in this work is emulation based. Future work plans include building the *EnforSDN* prototype and evaluating its performance and functionality in the realistic SDN setting.

VII. Acknowledgment

The research leading to results published in this paper is partially supported by the European Communitys Seventh Framework Programme (FP7/2001-2013) under grant agreement number 619572, in the context of the COSIGN Project.

References

[1] D. A. Joseph, A. Tavakoli, and I. Stoica, “A policy-aware switching layer for data centers,” *SIGCOMM Comput. Commun. Rev.*, 2008.

[2] R. Potharaju and N. Jain, “Demystifying the dark side of the middle: A field study of middlebox failures in datacenters,” in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC ’13. New York, NY, USA: ACM, 2013, pp. 9–22.

[3] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, “Design and implementation of a consolidated middlebox architecture,” in *Proc. NSDI*, 2012.

[4] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, “Making middleboxes someone else’s problem: Network processing as a cloud service,” *ACM SIGCOMM Computer Communication Review*, 2012.

[5] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, “The middlebox manifesto: Enabling innovation in middlebox deployment,” in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 2011.

[6] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[7] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, “Extending networking into the virtualization layer.” in *Hotnets*, 2009.

[8] “EnforSDN evaluation code, 2014 [online],” https://github.com/EnforSDN/trunk/.

[9] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in *ACM SIGCOMM*, 2008.

[10] C. Gomez, F. Gilabert, M. E. Gomez, P. Lopez, and J. Duato, “Deterministic versus adaptive routing in fat-trees,” in *IPDPS 2007*. IEEE, 2007, pp. 1–8.

[11] X.-Y. Lin, Y.-C. Chung, and T.-Y. Huang, “A multiple lid routing scheme for fat-tree-based infiniband networks,” in *Parallel and Distributed Processing Symposium, 2004*. IEEE, 2004, p. 11.

[12] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, “Optimized infinibandtm fat-tree routing for shift all-to-all communication patterns,” *Concurrency and Computation: Practice and Experience*, 2010.

[13] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center tcp (dctcp),” *ACM SIGCOMM*, 2011.

[14] (2005, October) netfilter/iptables project. http://www.netfilter.org/projects/.

[15] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, “Understanding tcp incast throughput collapse in datacenter networks,” in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 73–82.

[16] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu *et al.*, “Ananta: cloud scale load balancing,” in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 2013, pp. 207–218.

[17] V. Mann, K. Kannan, A. Vishnoi, and A. S. Iyer, “Ncp: Service replication in data centers through software defined networking,” in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 561–567.

[18] H. Andrade, B. Gedik, and D. Turaga, *Fundamentals of Stream Processing: Application Design, Systems, and Analytics*. Cambridge University Press, 2014.

[19] (2010, June) Building three-tier architectures with security groups. [Online]. Available: aws.amazon.com/blogs/aws/building-three-tier-architectures-with-security-groups

[20] J. Pettit and T. Graf, “Stateful connection tracking & stateful nat,” 2014.

[21] B. Carpenter and S. Brim, “Middleboxes: Taxonomy and issues,” in *RFC 3234, February*, 2002.

[22] S. M. Bellovin, “Distributed firewalls,” *Journal of Login*, 1999.

[23] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith, “Implementing a distributed firewall,” in *Proceedings of the 7th ACM conference on Computer and communications security*. ACM, 2000, pp. 190–199.

[24] J. Lee, J. Tourrilhes, P. Sharma, and S. Banerjee, “No more middlebox: integrate processing into network,” *SIGCOMM Comput. Commun. Rev.*, 2010.

[25] S. Guha and P. Francis, “An end-middle-end approach to connection establishment,” *SIGCOMM Comput. Commun. Rev.*, 2007.

[26] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker, “Middleboxes no longer considered harmful,” in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, 2004.

[27] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, “SIMPLE-fying middlebox policy enforcement using sdn,” *SIGCOMM Comput. Commun. Rev.*, 2013.

[28] S. K. Fayazbakhsh, M. K. Reiter, and V. Sekar, “Verifiable network function outsourcing: Requirements, challenges, and roadmap,” in *Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, 2013.

[29] G. Gibb, H. Zeng, and N. McKeown, “Outsourcing network functionality,” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012.

[30] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, “Stratos: A network-aware orchestration layer for middleboxes in the cloud,” *CoRR*, 2013.

[31] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, “Toward software-defined middlebox networking,” in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012.

[32] ETSI NFVISG, “Network functions virtualisation. introductory white paper,” 2013.

[33] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, “Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags,” in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, 2014.

[34] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, “Flowguard: building robust firewalls for software-defined networks,” in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 97–102.

[35] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, “Frenetic: A network programming language,” in *ACM SIGPLAN Notices*, vol. 46, no. 9. ACM, 2011, pp. 279–291.

[36] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, “OpenNF: enabling innovation in network function control,” in *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 2014, pp. 163–174.

[37] Y. Sun, M. Xu, J. Lian, and C. Shieh, “System and method for dynamic security insertion in network virtualization,” Oct. 17 2013, US Patent App. 13/861,220.