Efficient Provisioning of Security Service Function Chaining Using Network Security Defense Patterns

Alireza Shameli-Sendi, Yosr Jarraya, *Member, IEEE*, Makan Pourzandi, and Mohamed Cheriet, *Senior Member, IEEE*

Abstract—Network functions virtualization intertwined with software-defined networking opens up great opportunities for flexible provisioning and composition of network functions, known as network service chaining. In the cloud, this allows providers to create service chains tuned to each application type while optimizing resources' utilization. This is particularly useful to accommodate different tenants' applications with different security needs. However, considering security provisioning from the single perspective of resources optimization may lead to deployment solutions that do not comply with well-known security-related best practices and recommendations. In this paper, we propose network security defense patterns (NSDP) aimed at leveraging the best practice and know-how from the security experts and at capturing various security constraints to efficiently select compliant security functions' deployment options. The placement problem being a NP-Hard problem to solve, we also propose a scalable networking and computing resources aware optimization framework to efficiently provision different NSDPs. We further show the feasibility of implementing NSDPs in the cloud infrastructure through the integration of our approach into an open source cloud framework, namely OpenStack, in our test laboratory. The simulation results show the effectiveness of our approach in selecting an optimal placement of the security functions for large data centers with hundreds of thousands of computing nodes, while complying with the predefined security constraints and improving the scalability compared to the current placement algorithms.

Index Terms—Cloud computing, SDN, NFV, Service chaining, Network security defense patterns, Optimal placement.

1 Introduction

Network Function Virtualization (NFV) is a promising network architecture, in which it implements network functions through software virtualization techniques and runs them on commodity hardware. NFV makes the provisioning of network functions more flexible and cost-effective, but like other emerging technology, it brings several challenges to network operators, such as their dynamic instantiation, migration, placement, and etc [1].

Service Function Chaining (SFC) is a technique for selecting and steering data traffic flows through various "network functions" based on Software Defined Networking (SDN) [2], [3]. New virtualization capabilities and SFC enable new ways of building an adequate security solution for cloud-based applications. Because, security functions can be dynamically instantiated, automatically deployed, and transparently inserted into the traffic flow to address different security needs for different application types. To achieve these objectives, NFV generally leverages advancements in technologies such as SFC. In particular, SFC and NFV enable the flexible dynamical service chain modifications to meet the real time demand.

In this paper, we focus on the efficient placement of virtual security function when deployed using SFCs, to make sure that they are used most effectively and least expensively. Several works in the literature propose approaches to address the optimal placement problem (known to be NP-hard [4], [7]). But to cope with the underlying complexity, they generally limit the optimization scope to a subset of the utilized resources (e.g. either computing or network resources) [4], [5]. Nevertheless, considering only the pure optimization problem is not sufficient for the placing the security functions. Indeed, creating any security functions almost anywhere in the network and composing them in different ways may lead to several deployment options, valid from optimization point of view but invalid from security point of view. These options may break tenants' security requirements as they do not consider the security deployment constraints and/or security-related best practices and recommendations. In this paper, we propose Security Defense Patterns Aware Placement (SDPAP) for an efficient placement which additionally to the resources takes into account security deployment constraints expressed through a fine grained representation of security functions using Network Security Defense Patterns (NSDP). To achieve this SDPAP, first we propose to transform high level security needs expressed by the tenants into a set of security patterns. Further, we introduce the concept of the security deployment constraints needed for correctly expressing the relationship between different security functionalities in different NSDPs. For example, the order is needed to express the correct relationship between placing an IDS and a VPN termination (see details in the section 2). Secondly, we efficiently provision security service functions in the cloud taking into account the security deployment constraints while taking into account deployment costs for each NSDPs. However, the optimal Virtual Network Functions (VNF) deployment is NP-hard making

- *A. Shameli-Sendi is with the Faculty of Computer Science and Engineering, GC, Shahid Beheshti University (SBU), Tehran, Iran.*
- *Y. Jarraya and M. Pourzandi are with Ericsson Security Research, Canada.*
- *M. Cheriet is with University of Quebec (ETS), Canada.*
  E-mail: a_shameli@sbu.ac.ir, yosr.jarraya@ericsson.com, makan.pourzandi@ericsson.com, mohamed.cheriet@etsmtl.ca

Manuscript received July 18, 2016; revised September 23, 2016.

it impossible for realistic use for large cloud environments. We therefore propose two heuristic algorithms to overcome the scalability for our optimization algorithm. We first apply the network partitioning heuristic to partition the network into several logical independent blocks based on the network topology. We then apply our second heuristic, called segmentation, into each partition. The segmentation further considers security function placement in each partition based on the security deployment constraints (see details in the section 5.2). The partitioning and segmentation techniques significantly decrease the algorithm's complexity and improves the overall scalability of the optimal placement up to 69K nodes.

Through studying the state of the art we found a rich set of different patterns used for an efficient implementation of SFCs in the cloud (e.g., single chain by considering the order of VNFs [6], multi-chain [4], or decomposing a VNF to more refined functions [7]). Those patterns are used individually to handle specific problems. In order to leverage the know-how from these patterns into some general re-usable approach to better implement these existing patterns and use it for future patterns, we conceived a new framework enabling the efficient implementation of network security patterns in the cloud using SFCs [8], [9].

The main contributions of this paper are as follows: (1) We identify a set of network security defense patterns (NSDP) to capture the know-how knowledge of network security experts. Additionally, we propose new patterns based on our own experience (probe-point, zone aware, location-aware). These patterns allow expressing different deployment solutions for different tenants in the cloud. In this paper, for the first time to the best of our knowledge, a general framework is proposed to implement these patterns for security functions placement in the cloud, (2) We propose to take into account different security deployment constraints in optimal placement of security functions. Taking into account security deployment constraints in the placement algorithms has not been previously addressed in the aforementioned research works. We introduced four new deployment constraints: *Region*, *Co-location*, *Distribution*, and *Waypoint*. These deployment constraints are necessary to eliminate incorrect deployments of security functions. Additionally, by taking into account these security deployment constraints we improve the scalability of our placement algorithm. Some security deployment constraints such as region based additionally improve the overall security by directing the placement algorithm toward the optimal solution from security point of view, and (3) We compare our approach to relevant state-of-the-art approaches and show the feasibility and scalability of our approach for large data centers' networks consisting up to 69K nodes.

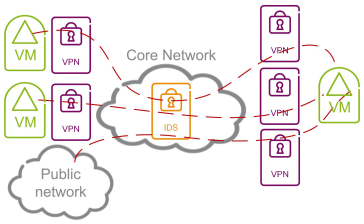
The rest of this paper is organized as follows: Section 2 describes the problem that we intend to solve. Section 3 provides the background and literature review. Section 4 describes our proposed security defense patterns and presents examples of simple and combined NSDPs that will be used throughout the paper to explain and show experiments using our approach. Section 5 explains the SDPAP approach for the efficient placement of network security defense patterns in the cloud. There, techniques used to tackle security issues, namely partitioning and NSDP-aware segmentation, are presented. Section 6 presents the mathematical formulation of the SDPAP placement algorithm. Section 7 discusses the SDPAP prototype, its integration into OpenStack, and the experimental results. Finally, Section 8 concludes the paper and provides insights for potential future work.

2 Problem Statement

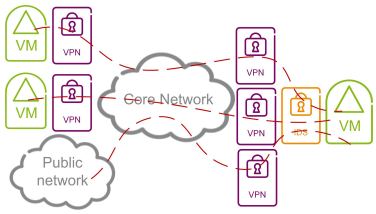
Generally, optimization of VNF deployment has as a main objective to find the best optimal solution from the performance and cost points of view. In contrast, security requirements emanating from tenants' specific security policies or general security-related best practices are not necessarily considered. This might be justified by the fact that taking into account these constraints may lead to a more complex problem that cannot be solved in a reasonable amount of time. For example, the order requirement of traversing certain security functions is generally recognized (e.g. in [5]) to make the optimal placement problem more difficult to solve. Thus, similarly, taking into account other security deployment constraints individually or in combination may lead to a more complex problem. At the same time if we omit security deployment constraints, the found optimal solution may eventually compromise security.

Assume a scenario in which a set of virtual VPN endpoints and a virtual intrusion detection system (IDS) need to be deployed to secure the communication between a set of virtual machines. Security requirements would be that (i) the traffic should be encrypted before reaching the untrusted core network and (ii) all traffic should be inspected by the IDS. If we omit these requirements and focus only on the optimal placement of these functions, we may find an invalid solution in which only parts of the traffic are inspected, or the traffic is transmitted as cleared in non-trusted parts of the network. For example, if we omit the first requirement and assume that the resources are available and less costly at the core network, the optimization algorithm may choose to deploy VPN terminations in the core. This deployment solution makes these VPN deployments useless, as unencrypted traffic may traverse a large section of the non-trusted network. Additionally, if we omit the second requirement, the optimization algorithm may choose to deploy the IDS at the core network and the VPN next to the edges, as shown in Figure 1a. This causes the encrypted traffic to traverse the IDS, making it impossible for the IDS to fully analyze the encrypted traffic. Thus, there is a need to constrain the location of the VPN end points to be at the edges near to the source and destination, respectively. Furthermore, a strict ordering of the VPN end points needs to be observed. A valid deployment taking into account these security deployment constraints is shown in Figure 1b. These are only some examples of well-known security best practices which should be taken into account for a valid and efficient solution.

SDPAP aims at efficiently providing a solution for these issues. These can be very challenging objectives for several reasons. First, well-known security best practices and recommendations are scattered in several documents and usually provided by the network security architect expert on a case-by-case basis. The latter works with a network engineer to determine the best combination of security products and services while balancing security, throughput and costs. Therefore, a solution that collects the security expert's knowledge and know-how and provides the means to automatically deploy it in the cloud can dramatically improve the overall security of the deployed solutions. Second, there is a need to consider an optimization model that simultaneously involves both computing and network resources in order to efficiently provision security in the cloud. Since today's data centers are tending to increase in size and, as a result of this large size, considering the whole placement solution space becomes difficult, there is a need for new heuristics to address the optimization



(a) Potentially optimal solution that does not meet all security requirements



(b) Possible solution that satisfies all security requirements

Fig. 1: Motivating Example: Valid and invalid IDS and VPN endpoint deployments. The deployment (1a) does not allow for all traffic, specifically the encrypted one, to be inspected by the IDS. The deployment (1b) satisfies all requirements

problem. Finally, considering multiple security requirements in addition to the optimization problem may lead to an increasingly difficult and complex problem.

In this work, we propose to provide the mechanisms to capture security requirements and best practices that enable automation of complying with network security deployment at the provisioning phase. As data centers are increasing in size, this results in a multitude of choices for the placement of VNFs. This increase in the potential number of solutions has been acknowledged to make the problem NP-hard. There is therefore need to derive the right techniques to scale to large data centers while optimizing resources.

In this paper, we assume the use of software-defined networking for network infrastructure management and hosting a set of cloud customers' applications. Each customer's application requiring different security needs, customer provides a high-level security deployment description of those needs to the cloud provider. Our objectives in this paper are manifold: (1) Capture the security-related best practices for the deployment of network security functions; (2) Optimize security provisioning costs by simultaneously (i) placing the VNFs in the optimal locations and (ii) finding the optimal routing paths through the proper VNF sequence for each traffic request, while respecting the capacity constraints and the security deployment constraints derived from security-related best practices; (3) Scale to large-sized cloud computing data centers.

3 Related Work

Placement Objective. In the context of optimal VNFs deployment, related works can be classified into four main categories. In the first category, a fixed number of middleboxes are assumed to be already deployed in the network, and the optimal solution attempts to find for each traffic flow the optimal routes through them. In [10], an ILP formulation is proposed to optimally steer flows through static middleboxes in the network. In the second category, a set of pre-defined routes is assumed and the optimal solution is found for placing the VNFs within static routes. Sekar et al. [5] propose an optimization approach that assumes predefined paths between sources and destinations and then tries to find the optimal placement of the virtual appliances on these paths. The third category tackles these two objectives separately. The objectives are prioritized and then run sequentially. For example, in [11], the first step is to dynamically find the optimal computing node to initiate a service function, and then traffic is routed through the initiated service. In contrast, Addis et al. [12] minimize first the maximal link utilization; and then, after setting the routes, as a second objective, they minimize the network core utilization. In the last category, both objectives are considered simultaneously to tackle the optimal placement. A few studies (e.g. [6], [7], [13], [14]) propose to tackle these objectives simultaneously. Our proposed placement model lies in the last category. The optimal placement of a service chain is known to be NP-hard. Thus, finding the optimal solution might not be affordable for large network within reasonable time. In contrast to all other solutions mentioned above, we propose to restrict the placement locations in the network based on the security deployment constraints for the security functions placement. Furthermore, we propose new heuristics based on partitioning the network topology.

Network Security Patterns. Different service function chaining patterns were found in literature. For example, order constraint of VNFs [6], [7], [13], [14], multi-chain [2], [4], and decomposition of service function into more concrete ones [7]. Those patterns are used individually to handle specific problems. Contrary to the existing works, we propose a new framework enabling the efficient implementation of network security patterns in the cloud using SFCs [8], [9]. Furthermore, none of the existing work expressed the security deployment constraints separately or took them in the placement algorithm. To the best of our knowledge, we are the first to explicitly capture and represent security deployment constraints and use them to direct the placement algorithm for eliminating the incorrect placements and make our approach more scalable.

Scalability. Sahhaf et al. [7] show the feasibility of their placement, which is based on only one pattern (decomposition), for two different topologies, one with 24 nodes and 37 edges, and another 110 nodes and 148 edges. In [4], they show the feasibility of the multi-chain placement for several networks, while the largest one has 252 nodes. Also, the model presented in [6] simulated for a network topology of 22 nodes and 64 links. While, the existing public cloud platforms consist of tens of thousands nodes [15]. The majority of existing placement approaches suffers the scalability problem. Our solution is more scalable compared to the existing work. We show the feasibility of our approach in a large dataset consists of 69,696 nodes and 196,608 edges.

Compared to our previous work [14], we propose in this paper a novel concept that is a set of network security patterns that can be used for SFC. These patterns provide the means to the tenant to express his logical security needs as they encompass best security practices. The second completely new contribution is to take into account security deployment constraints instead of a purely cost optimization constraints as proposed in [14]. These constraints are necessary to improve the choice of the placement solution from the security point of view. We additionally go beyond the use case approach presented in [14], and propose a framework, namely SDPAP, that implements those patterns and testes our approach in a cloud setup managed by Openstack. Finally, we experimented with various new deployment scenarios and using new heuristic, we improved the scalability of our approach by allowing to support large networks consisting of about 69K nodes compared to networks with only 9K nodes in [14].

4 Network Security Defense Patterns

In the literature, various network security best practice and deployment recommendations have been acknowledged to enable efficient security provisioning. These recommendations are meant mainly to choose the appropriate combination of security functions and the adequate deployment architecture for the defense mechanisms that allow achieving several desirable objectives, including but not limited to the correct behavior of the security functions, minimal resource waste, resiliency to failure, etc. For instance, in many cases, a strict ordering of some security functions is needed in a chain for appropriate functioning of the security functions. For example, to inspect encrypted traffic, the firewall FW-L4-7 should be placed before the VPN encryption end point or just after the VPN decryption end point. Furthermore, several security functions would work better at specific locations in the network. For example, a DDoS attack blocking function is known to be best deployed as close to the source as possible [16]. In the same vain, end-to-end security dictates that traffic encryption should be performed before traversing the untrusted network [17]. From the architectural standpoint, there are mainly two deployment modes: in-line and out-of-band. Security functions that need to act in real time (e.g. prevent malware and malicious activities) have to be deployed in-line with the network traffic. The out-of-band mode is used chiefly for monitoring purposes (e.g. inspecting traffic and generating alarms). These requirements also need to be captured. Capturing these recommendations would help us to enable efficient security provisioning and deployment automation. Therefore, we propose a set of NSDPs that model different deployment-related and security-related recommendations.

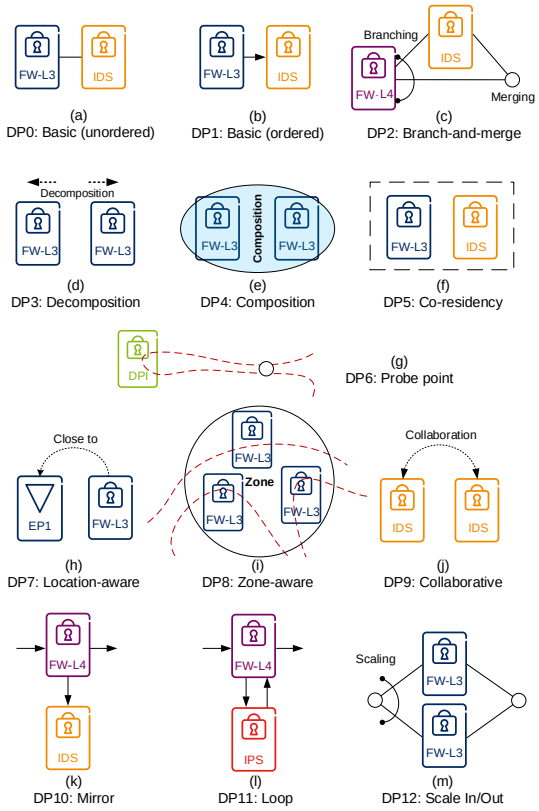


Fig. 2: Proposed set of network security defense patterns

4.1 Description of Patterns

Figure 2 illustrates a set of examples of NSDPs designed using a graphical representation. It is worth noting that a NSDP can be an elementary VNF or a combination using the patterns in Figure 2. We designate elementary VNF basic virtual security functions of different types, such as a virtual stateless firewall (FW-L3), a virtual stateful firewall (FW-L3-4), a virtual application layer firewall (FW-7), a Virtual Private Network gateway (VPN), a virtual Intrusion Detection/Prevention System (IDS/IPS), a virtual Deep Packet Inspection (DPI), and a virtual web application firewall (WAF). From now on, we name an elementary VNF as a Security Module (SM). For the sake of illustration, we use SMs to illustrate NSDPs; however, any SM in Figure 2 can itself be a NSDP to allow combination of NSDPs. Later in this section, we provide four examples of NSDP combinations. In the following, we present the proposed NSDPs.

Basic (unordered): This pattern captures a single-chained NSDP where there is no order between the security functions. The absence of order is represented by the undirected arrow. Figure 2a illustrates a NSDP consisting of a FW-L3 and an IDS where the implementation of those functions do not need to comply with any specific order of traversal.

Basic (ordered): This pattern [6], [7], [13], [14] illustrates an ordered single-chained NSDP where there is a strict order between the security functions. The order is represented using the directed edge. Figure 2b illustrates a NSDP where a FW-L3 should be traversed before an IDS.

Branch-and-Merge: This pattern [2], [4] is a multi-chain pattern that specifies the need of splitting the traffic into different types of flows where each type has to traverse different security chains. Figure 2c illustrates the case where some encrypted traffic (e.g. https), after traversing the firewall, would not be inspected by an IDS.

Decomposition: This defense pattern [7], [18], [19], [20] denotes a NSDP that splits the functionality of the security function (or NSDP) into two functions typically of the same type, where each of them implements only a subset of the overall functionality. This can be used for instance to de-localize the functionality into different locations. This pattern is a key step for implementing distributed security (e.g. distributed firewall [21]). Figure 2d illustrates a NSDP consisting of a firewall functionality decomposed into two firewalls.

Composition: This defense pattern [7], [19], [20], [22] is the complementary pattern of the previous one, as it allows merging the functionality of multiple functions of the same type into a composed function. For instance, it could be more beneficial to have a single security function instance for multiple flows. Another alternative use for this pattern is to constrain the bidirectional flows that should traverse the same stateful security function. Figure 2e illustrates a NSDP consisting of the combination of the functionality of two stateful firewalls. Stateful firewalling assumes that the firewall can see the bidirectional message exchange flow between two ends. Therefore, in the bidirectional flows example which consists of two stateful FW-L3 placements, we need to compose these two FW-L3 to enable a complete vision on bidirectional flow. If we do not apply this pattern, each flow will have its own stateful security function. Thus, each of these stateful security functions is not able to see the traffic of the other side.

Co-residency : This pattern [4], [5] specifies that two security functions should be placed on the same node. This may be needed to eliminate additional link delays between the two functions or

to avoid a security breach from occurring if these two functions are de-localized. For instance, Figure 2f illustrates a NSDP that requires that an IDS be placed at the same location as a firewall to detect potential malicious activities as soon as possible.

Probe Point: This security pattern specifies that the traffic should be steered through a security function with a fixed location. This would be the case, for example, of a security function implemented in a specific type of hardware pre-installed in a fixed location of the network. Figure 2g illustrates a DPI function that necessarily has to be traversed by the traffic.

Location-aware: This pattern specifies that some SMs should be at a certain distance from a reference location. For example, Figure 2h illustrates that FW-L3 should be deployed close to EP1. This is may be useful when we need a specific placement relative to a reference point where detection, network analysis and responding would be more efficient. For example, in the context of DDoS detection, the attack detection is most accurate close to the victim, while the response and separation of legitimate traffic from attack traffic is most successful close to the sources [23]. A firewall close to the source can eliminate bad traffic early and reduce an unwanted traffic footprint.

Zone-aware: This defense pattern defines a zone where multiple SMs should be placed. It can be used to specify that some SMs should be placed at the same zone managed by the same controller. Figure 2i illustrates that some firewalls related to different chains need to be localized in the same zone.

Collaborative: This pattern [24] specifies the need of a set of security functions to collaborate. For instance, Figure 2j specifies that two IDSs need to collaborate. In this case, a better visibility on the global security status can be obtained by these IDSs, which may increase the security.

Mirror: This pattern [10], [25] captures the deployment of a security function (or NSDP) in an out-of-band fashion with respect to another function or chain. This architecture allows specifying that the traffic should be simply mirrored to the out-of-band security function. This is typically the case of an IDS in Figure 2k that needs to analyze a copy of the received traffic without acting on it.

Loop: The loop defense pattern [25] is fundamentally different from mirroring because the traffic has to return back to the main service chain after being processed by all SMs in the loop path. Figure 2l illustrates that when the traffic reaches FW-L4 and is accepted, it should be directed to IPS, where the latter should act actively on malicious traffic and send back the sanitized traffic to the same node to proceed in the main service chain.

Scale in/out: This pattern [26], [27] increases the resiliency of the security measures in the face of volumetric attacks. Scaling in/out is completely different from decomposition pattern. Consider the example provided in Figure 2m. The scale in/out means creating dynamically new firewall instances with the same policy and there is no indication of extracting the rule sets for new. While, in the decomposition process, a set of rules are extracted from the main firewall and exported to some other firewall instance, e.g., a firewall close to the source of the malicious traffics. On the other words, decomposition means that the firewalls are instantiated on other places with different security policies meanwhile scaling in/out means the cloning firewalls with the same security policies.

4.2 Security Deployment Constraints

The security requirements expressed by the security patterns are implemented by the security deployment constraints in our proposed optimization algorithm. Table 1 presents how NSDPs are using these constraints. We introduce four new deployment constraints (Region, Co-location, Distribution, and Waypoint) in addition to the order constraint which existed before.

TABLE 1: Relationship between NSDPs and security deployment constraints

	Security Deployment Constraints				
Defense Patterns	Order	Region	Co-location	Distribution	Waypoint
Basic (unordered)					
Basic (ordered)	✓				
Branch-and-Merge			✓	✓	
Decomposition				✓	
Composition			✓		
Co-residency			✓		
Probe Point					✓
Location-aware		✓			
Zone-aware		✓			
Collaborative		✓		✓	
Mirror			✓		
Loop			✓		
Scale in/out				✓	

Order: This constraint enforces the order of SM deployment for the placement algorithm. The order is a list of SMs in a specific order with respect to the direction of traffic from a source end point to a destination end point. This constraint is used as the security appliances are chained in a specific order to perform different security functions on the traffic. For example, a tenant may expect the traffic reaching his services to pass first through a $FW-L3$, and then an IDS, to filter disallowed traffic ahead of the detection process on legitimate traffic.

Region: This constraint enforces the SMs to be placed in a specific region of a nodes. This region can be defined close to a endpoint. This constraint concerns three NSDPs: location-aware, zone-aware, and collaborative. For example, for location-aware pattern we can define a region close to a VM in order to filter out malicious traffic as soon as possible. For the zone-aware pattern the region can be defined as close to the cloud entry point to filter traffic at the entrance of the virtual data center. In the collaborative pattern, two collaborating IDSs are needed to be placed close to each other in the same region in order to cover the maximum of flows to the virtual data center.

Co-location: This constraint provides the possibility to place several SMs of the same or different types at the same node. For example, in the composition NSDP, since the traffic of opposite directions should be steered through the same stateful security appliances, they are instantiated at the same node as one SM (co-location of the same types).

Distribution: This constraint enforces that some SMs must be placed at different nodes. The distribution constraint is used in several NSDPs, including decomposition, scale in/out and branch-and-merge. For example, in the collaborative pattern with distribution constraint, the IDS SMs must be placed in different nodes in order to maximize the coverage of flows. In the decomposition pattern, the SM is divided into two new SMs which in turn must be placed in different nodes in order to improve SM resiliency, scalability and operability.

Waypoint: This constraint enforces the placement algorithm to make pass all SFC flows through a defined node. For example, in the probe point pattern, all flows need to pass through a node with specific SM functionality (e.g. DPI) in order to be analyzed.

4.3 Illustrative Examples of NSDPs

In this section, we explain four different deployment scenarios for the protection of different applications. The first and the second are two different defense models for protecting a web application. The

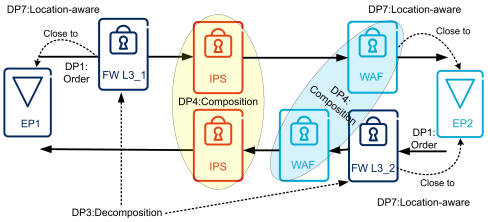


Fig. 3: Scenario 1: Four NSDPs - order, composition, decomposition, and location-aware

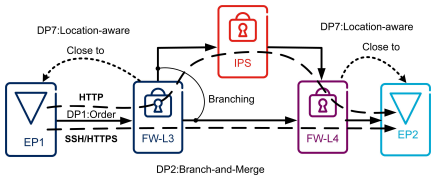


Fig. 4: Scenario 2: Three NSDPs - Order, Branch-and-Merge, and location-aware

third scenario is a more complex one in which the two preceding scenarios are combined. The last represents a NSDP for a typical 3-tier application (see [9], page 18).

In the first scenario, we assume that three security functions are requested by a cloud tenant to secure the communication between clients (EP1 is the first entry point in the cloud) and a Web server (EP2) using three security functions: FW-L3, IPS and WAF. Therein, four NSDPs are combined: basic ordered, location aware, composition, and decomposition. As illustrated in Figure 3, the first NSDP is *decomposition* where FW-L3 is decomposed into two sub-firewalls, FW-L3_1 and FW-L3_2. This pattern is combined with the *location-aware* defense pattern, which stipulates that FW-L3_1 should be placed close to EP1, and FW-L3_2 and WAF close to EP2. The combination of these two patterns allows eliminating the unwanted traffic footprint from each source early on before it reaches the core network. The third NSDP is *composition*, where we need to steer the traffic of any flow in both directions through the same IDS and WAF, for better visibility on the potential attacks. The two security chains in both directions are asymmetrically traversed by the traffic as specified by the order between different SMs.

The second scenario assumes that a cloud tenant needs to secure communications between clients and a Web server, where different kinds of traffic related to different protocols should be inspected by different security chains. As illustrated in Figure 4, SSH/HTTPS traffic requires traversing FW-L3 followed by FW-L4, whereas HTTP traffic has to traverse FW-L3, IPS and then FW-L4. This scenario consists of three NSDPs: *Order*, *branch-and-merge* and *location-aware*. The *branch-and-merge* pattern allows specifying that traffic between two end points should be split into two chains. The *location-aware* restricts the deployment of the FW-L3 to be close to EP1 and the deployment of FW-L4 to be close to EP2.

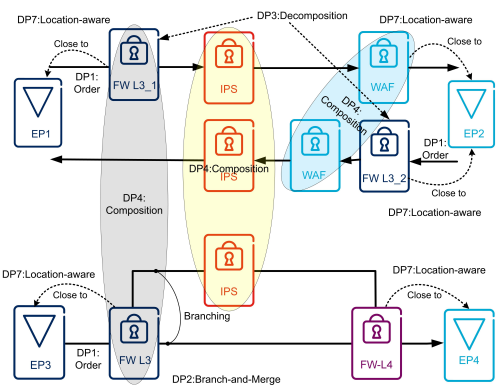


Fig. 5: Scenario 3: Combination of the scenario 1 of Figure 3 and scenario 2 of Figure 4

The third scenario represents the combination of the first and the second scenarios, where the NSDPs for securing multiple chains for different applications are requested simultaneously. This allows the specification of security requirements and best practices that concern multiple chains of multiple applications. For instance, for the sake of minimizing resource consumption (the number of spawned SMs), the same types of SMs are merged in each of the NSDPs into the same SM. As illustrated in Figure 5, the two firewalls, FW-L3-1 and FW-L3, have to be composed together and at the same time close to both the end points EP1 and EP2. Moreover, the two IPSs from the two scenarios are made to be composed together to form a single IPS. The benefits of such a scenario include the reduction of computing and networking resources, the number of instances of the same types and obtaining a global view to detect distributed attacks.

The Scenario 4 represents a defense model for a 3-tier application. As illustrated in Figure 6, the three communications between the elements of the 3-tier application have different security needs. Thus, three combinations of NSDPs can be extracted for each of these communications. The combined NSDP on the left, between the gateway and the presentation layer, includes five NSDPs, which are *location-aware*, *composition*, *decomposition*, *order* and *loop*. For example, the *composition* pattern is for the stateful security appliances (i.e. WAFs and IPSs) that ought to see both directions of the traffic. The combined NSDP on the right, between the business layer (Web server BL) and the DB server, includes *mirror*, *decomposition* and *location-aware* patterns. Mirror NSDP is seen where the two IDSs are connected to the chain in an out-of-bounds fashion to analyze a copy of the traffic.

5 Security Defense Pattern Aware Placement (SDPAP)

In this section, we first present a high-level view of our approach, namely Security Defense Pattern Aware Placement (SDPAP). It employs the aforementioned NSDP together with our proposed optimal placement algorithm to find the optimal deployment solution of security service chains. The NSDP guarantees compliance with well-known security related best practices. We then describe the two techniques that we propose to tackle the scalability issue. Finally, we present the algorithms underlying our approach.

Our approach assumes that the tenant/application deployer presents the cloud service provider (CSP) with a high level security architecture of the desired security to be implemented. This high level security architecture is then transparently mapped by the CSP provider into its cloud infrastructure. We detail our approach in the following. We assume that the CSPs provide pre-defined security templates to their tenants [1]. These templates help the tenants and application deployers to put in place best security practices. For example, for a 3-tier application, CSP can provide different templates: one would have FW-L3 between

1. e.g., see AWS CloudFormation https://aws.amazon.com/cloudformation/

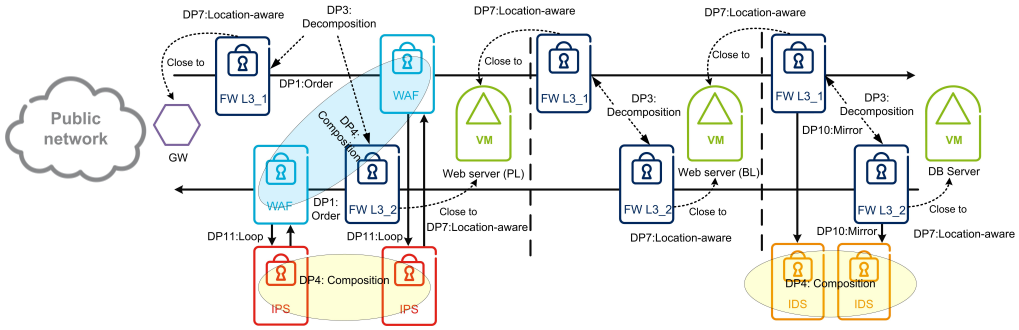


Fig. 6: Scenario 4: NSDP for a 3-tier application

layers, another would have only IDS between the business and data layers, another would place IPS for the first layer, or any other combination of the security appliances. Once the tenant has provided the high-level logical security view to the CSP, the CSP maps it into its cloud infrastructure. In practice, the logical view is translated into the implementation at cloud infrastructure level using single or multiple combined NSDPs. Therefore, the client does not need to be aware of the underneath network topology or computing resources (e.g., computing nodes CPU loads, bandwidth availability for different network links) to put in place the most efficient and effective security posture.

Placement and chaining security deployment constraints (e.g. ordering, co-location, location-aware) for the different SMs are extracted and used to drive our placement algorithm. Indeed, these constraints allow pruning invalid optimal solutions that do not meet the security requirements and recommendations as stated in the NSDP. Furthermore, we use these deployment constraints to improve the scalability of our approach by taking them into account in one of our heuristics, "segmentation".

To tackle the scalability problem, two heuristics are proposed: (1) partitioning, and (2) segmentation. We propose to partition the network topology into several independent blocks such that the optimal value stays the same with or without partitioning. After partitioning, we apply our second heuristic, called segmentation, into each partition. The segmentation idea limits the resources availability in each partition for all security modules exit in low-level implementation of NSDPs with respect the semantics underlying the security function and the best location where it would operate efficiently. Finally, based on the requested NSDPs and having applied our heuristics, we run our optimization algorithm in all partitions in parallel to determine the optimal placement of the SMs and the routes among them. At the end, the best solution is selected from one the partitions, the one how minimizes our objective function better (see Eq. 1). Details about our approach will be provided in the remaining sections.

5.1 Partitioning

Concerning the topology of the network, conventional data centers are commonly designed using a tree-like topology according to a three-tier architecture [28]. At the bottom level, servers organized in racks are connected to at least one Top-of-Rack (ToR) switch. ToR switches form the access tier. Each ToR switch connects to at least an aggregation switch at the aggregation tier; and finally, each aggregation switch connects with multiple core switches at the top level. A core switch connects to the various aggregation switches and provides connectivity to the outside world. The architecture is generally organized around tree-based topologies such as fat-tree, which is being considered by several related works (e.g. [15], [28], [29]). Thus, we will use fat-tree topology to illustrate our approach, but we believe that our approach can be applied on any tree-like topology design.

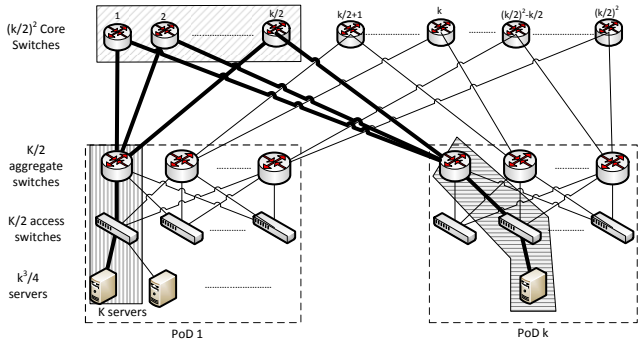


Fig. 7: Example of one block with three availability zones in 3-level k-ary fat-tree. Switches and links in the same block are illustrated by bold lines. A zone is illustrated by striped boxes.

An inherent characteristic of tree-based topology structure is the organization of the connectivity using sets of disjoint paths in the same PoD or in different PoDs between any two hosts [15]. For instance, if two hosts in two different PoDs have to communicate, both one of the aggregate switches and one of the core switches have to be involved. Once an aggregate switch is selected from the source ToR switch, the result is a set of possible paths that are disjointed with respect to the other paths associated with any other non-selected aggregate switch. On the basis of this observation, we propose to partition the network topology into several independent blocks. By independent, we mean that the paths in different blocks do not share any common node except the source, the destination and their respective ToR switches. Figure 7 illustrates the notion of block and zones in a k-ary fat-tree topology. Having a partitioned network topology, the optimization problem can then be addressed concurrently in these independent blocks by applying our SDPAP algorithm. Since these blocks consist of disjointed paths with respect to each other, once an aggregate switch is selected, the optimal solution lies always in one block. Therefore, the optimal value stays the same with or without partitioning. Thus, we propose to compute the optimal costs in all blocks in parallel and then compare them to find the optimal solution and the block in which to place our flows.

Given a graph G capturing the fat-tree topology along with the source and destination nodes provided as input, the Algorithm 1 describes the partitioning of G into blocks. Depending on the location of the communicating end points, we distinguish four possible cases determining which blocks are involved in the placement problem of two given security service chains:

Algorithm 1 TopologyPartitioning

```
Require: G: graph of the topology
Require: s-node: source node
Require: d-node: destination node
Ensure: Blocks[]: Set of blocks returned by this algorithm
  PhysSrc = G.getServer(s-node)
  PhysDst = G.getServer(d-node)
  if PhysSrc == PhysDst then                                  ▷ Case 1
      Blocks[1].addNodes({PhysSrc})
  else
      ToR-s = G.findSwitches(PhysSrc,up)
      ToR-d = G.findSwitches(PhysDst,up)
      if ToR-s == ToR-d then                                  ▷ Case 2
          Blocks[1].addNodes({PhysSrc,PhysDst,ToR-s)
      else
          Agg-s[] = G.findSwitches(ToR-s,up)
          Agg-d[] = G.findSwitches(ToR-d,up)
          sBlock.addNodes({PhysSrc,PhysDst})
          sBlock.addNodes({ToR-s,ToR-d})
          links = G.findLinks({PhysSrc,PhysDst},up)
          sBlock.addlinks(links)
          if Agg-s[] ∩ Agg-d[] ≠ ∅ then                       ▷ Case 3
              links = G.findLinks({ToR-s,ToR-d},up)
              sBlock.addlinks(links)
              sBlock.addNodes(Agg-s[])
              Blocks[1] = sBlock
          else                                                ▷ Case 4
              links = G.findLinks({ToR-s},up)
              i = 1
              for all link in links do
                  Block[i] = sBlock
                  Block[i].addLinks(link)
                  agg-s = G.findConnected(link, Tor-s)
                  Block[i].addNodes(agg-s)
                  links = G.findLinks({agg-s},up)
                  Block[i].addLinks(links)
                  cores-s[] = G.findSwitches(agg-s,up)
                  Block[i].addNodes(cores-s[])
                  links = G.findLinks(cores-s[],down)
                  Block[i].addLinks(links)
                  agg-d = G.findSwitches(cores-s, down)
                  Block[i].addNodes(agg-d)
                  Block[i].addLink(agg-d,ToR-d)
                  i = i + 1
              end for
          end if
      end if
  end if
  return Block[]
```

Case 1: Both source and destination VMs are located in the same physical node. In this case, there is a single node to instantiate any SM, which is the physical server hosting these VMs. This case can be easily addressed by instantiating the SMs at the server hosting the VMs.

Case 2: The VMs are located in different physical servers that are connected to the same ToR switch. This case maps to finding the spot with enough resources among physical servers hosting the source and the destination nodes, or in the ToR switch.

Case 3: The source and the destination are connected to 2 different ToRs but in the same PoD. In this case, there will be a single block, including the two physical hosts (i.e. at source and destination), the two ToR switches connecting the physical servers and the $k/2$ aggregate switches. We directly apply the SDPAP algorithm to this block of nodes.

Case 4: The source and the destination are connected to 2 different ToRs in two different PoDs. In this case, we partition the fat-tree into a set of blocks. Since in a k-ary fat-tree, we have $k/2$ aggregate switches per PoD, we will have in this case $k/2$ blocks. Each block will consist of six nodes (fixed part) plus k/2 other nodes (variable part) corresponding to the number of core switches. The fixed part consists of the two physical servers of both VMs, the two corresponding ToR switches and the two selected aggregate switches. Thus, in total, each block will contain $(K/2) + (3 * 2)$ nodes.

The number of blocks as well as the number of nodes and links within a single block depend on the value of the parameter k of the fat-tree and the location of the flow's end nodes. Note that in the variable sBlock is a sub-block that constitutes the common part between Case 3 and Case 4. The functions *findLinks* and *findSwitches* have two parameters: The first parameter is a list of (or single) nodes from where the search starts, and the second parameter specifies the direction of the search (up or down). The function *findConnected* is used to find a node given a link and the node connected at the other end of this link. The functions *addlinks* and *addNodes* are used to add the found links and nodes, respectively, to the block.

In terms of optimal placement, use-case 4 is the most complex among those enumerated. Therefore, we will consider use-case 4 as our main focus for optimal placement in the rest of this paper.

5.2 NSDP-aware Segmentation

The partitioning technique significantly decreases the algorithm's complexity and improves the overall scalability of the optimal placement algorithm. As will be seen later in the experimental results section, the algorithm still does not scale adequately enough. To address this problem, NSDP-aware segmentation defines the most adequate places/zones/clusters to deploy SMs. In practice, it means that in our placement algorithm we limit the availabilities of security functions in the network graph. We mean by availability, the existence of resources, specifically computing resources for the SM to function properly. The availability of each SM in a given node is the number of possible instances of this SM that can be created in this node. This is computed based on the overall available resources in each node and the total amount of resources needed by each SM. Thus, instead of exploring all possible nodes to place one SM in the search space, we restrict the feasible solution space for each SM to a predefined subset of a given graph. For example, in a DDoS attack an IDS has generally to be placed somewhere to see all the traffic to make the right decision. Furthermore, VPN terminations are useless if deployed in the core of the network, and a WAF or an IDS is generally deployed after the VPN termination. Additionally, some SMs (e.g. firewalls) are more efficient if deployed closer to the source or to the destination (c.f. a location-aware NSDP). Other examples would be (i) a zone-aware NSDP where we need to place a set of SMS in a cluster of nodes or (ii) in a scalable NSDP, where new SMs should be distributed across different nodes.

Therefore, from these observations, instead of making any security function available everywhere, we distribute the availability of each security function according to several criteria, including the semantics underlying the security function itself with the best location where it would operate correctly and efficiently, the trust, and the service chains we are aiming to place. By considering these facts in the distribution of availabilities, we can eliminate some kinds of inefficient and incorrect deployment of SMs. Note

that the segmentation preserves the order in the chain of SMs, providing the same functionality from the security point of view.

Thus, as a general segmentation model, we propose to divide a given service chain into *n* segments that we call security segments. For example, we can segment into three segments: Source segment (close to the source), destination segment (close to the destination), and core segment (the segment between the source segment and the destination segment). A security segment is defined as an ordered sequence of SMs that is generally composed of one or more security modules and that is part of a larger security chain. Additionally, we also divide in like manner each block into three zones. Then, we propose to assign each segment to one or many zone(s), where the SMs within the segment are made available in the zone's nodes. The assignment of segments to zones depends on the status of resource availabilities in these zones. The final deployment is most often a mixture of optimizations based on some or all of the factors mentioned above.

For this paper, we assume that the segmentation is done by the tenant's security expert on the basis of the type of each NSDP that resulted from SDPAP translation of the abstract logical view into a more granular security functionality. As the security experts deploy different security appliances in daily life on the basis of similar factors, we believe this assumption is realistic.

5.3 Multistage SDPAP

In the previous sections, we described how to optimize the placement for one request consisting of two chains. We could further improve the optimization mechanism by considering simultaneous resource optimization for several requests at the same time. The multi-stage SDPAP algorithm presented here uses partitioning and segmentation conjointly to solve the optimization of SMs in the cloud infrastructure for several requests. The Algorithm 2 describes our approach. First, the network is partitioned into different blocks and each block into different zones. Then, each service chain is divided into several security segments. Thereafter, each security segment is assigned to one or many zones. These assignments are used to generate the availabilities of SMs in the nodes of each zone in each block, using the function *defineAvailability*. Once the availability data are generated for all blocks, several instances of our solver are executed in parallel so that each thread is fed with the data concerning its associated block. Furthermore, each instance of the program is executed on the basis of the constraints extracted from the combined NSDP that results from the translation of abstract security to low-level implementation. Finally, the block with the minimal cost represents the solution in which the actual placement is performed.

Algorithm 2 Multi-stage SDPAP

Require: Chains[]: each chain has SMs with their types, and sizes
Require: s-node: source node
Require: d-node: destination node
Require: Constraints[]: constraints extracted from the translation of abstract security policies to low-level implementation security chains
Require: Status: cloud infra. links and nodes load
Require: G: Topology graph
Blocks[] = TopologyPartitioning(G, s-node, d-node) ▷ Partitioning
for all b in Blocks[] **do** ▷ Zoning
  b.Zones[] = defineZones(b)
end for
for all c in Chains[] **do** ▷ Segmentation
  c.Sub-chains[] = divide(c)
end for
for all c in Chains[] **do** ▷ Assigning segments to zones
  for all s in c.Sub-chains[] **do**
    for all b in Blocks[] **do**
      for all z in b.Zones[] **do**
        z.AssignedSeg[] = Assign(z, s, Status)
      end for
    end for
  end for
end for
for all b in Blocks[] **do** ▷ Define availabilities in zones
  for all z in b.Zones[] **do**
    z.SMAvailabilities[] = defineAvailability(z.AssignedSeg[])
  end for
end for
Sol[] = ParallelCompute(Chains[], Constraints[], Blocks[])
Sol[block-opt] = MinCost{Sol[b]; b in Blocks[]} ▷ Opt. solution
executePlacement(Sol[block-opt]) ▷ Deployment

6 Integer Linear Programming Formulation for SDPAP

In this section, we present notations, variables, objective function and constraints that are used in the integer linear programming (ILP) formulation of the NSDP optimal placement problem.

6.1 Notations and Variables

In our integer linear programming formulation, we use a flow per direction. At a high level, we need to decide, for each flow f generated by a source s_f and consumed by a destination d_f which nodes to be visited and where security functions have to be placed. These can be captured using two binary variables $x_{i,j,f}$ and $y_{i,l,f}$.

TABLE 2: ILP parameters and variables

$N = \{1, \cdots, n\}$	Set of network nodes
$F = \{1, \cdots, h\}$	Set of (unidirectional) flows between pairs of nodes
$M = \{1, \cdots, m\}$	Set of possible SMs
s_f (resp. d_f)	Node source (resp. destination) of the flow f, $s_f, d_f \in N$
L_f	Number of units of bandwidth needed by a flow f
$c_{i,j}$	Cost of allocating a unit of bandwidth on the link between nodes i and j
$L_{i,j}^{max}$	Maximum capacity in units of bandwidth supported by link (i,j)
$b_{i,l}$	Cost of instantiating a SM l at node i
$d_{l,f}$	Number of instances SM of type l required by the flow f
$q_{i,l}$	Number of instances SM of type l that can be instantiated at node i
$K_f \subset N$	Set of SMs that must be traversed by the flow f
$O_{l,f}$	Positive integer representing the order of the SM l with respect to the other SMs in the list K_f
$RProd_{i'}$	List of SMs that need to be instantiated with region constraint from reference node i'
$RFlow_{l,i'}$	List of flows that need to instantiate SM l with region constraint from reference node i'
$CProd$	List of SMs that need to be co-located
$CFlow_l$	List of flows that need to co-locate SM l. In no co-location case, it should contain one flow
$DProd$	List of SMs that need to be distributed
$DFlow_l$	List of flows that need to distribute the SM l
$WProd_{iw}$	List of SMs that are statically implemented in a specific location iw
$WFlow$	List of flows that need to visit a specific location

The first variable $x_{i,j,f}$ is defined for each flow f and each link between a pair of nodes i and j, and it records whether the flow f has to travel through the link (i,j). The second variable $y_{i,l,f}$ is defined for each combination of node i and SM l, and a flow f records whether the SM l has to be placed at node i and traversed by flow f. The solution to our optimization problem would be a set of $y_{i,l,f}$ and a set of $x_{i,j,f}$ that together denote the optimal placement of SMs and the optimal path to traverse these nodes. Table 2 summarizes parameters used in our formulation.

6.2 Objective Function

Our objective is to minimize the total costs consisting of the cumulative cost of allocating units of bandwidth to all flows from their respective source node to their respective destination node and the cumulative cost of placing all SMs requested by all flows.

Minimize:

$$\sum_{f\in F}\sum_{i\in N}\sum_{j\in N}(c_{i,j}\cdot L_f\cdot x_{i,j,f})+\sum_{f\in F}\sum_{i\in N}\sum_{l\in M}(b_{i,l}\cdot L_f\cdot y_{i,l,f}) \quad (1)$$

6.3 Constraints

6.3.1 General Constraints

The following is the description of the general constraints that represent the base of our optimization model when both computing and networking resources are optimized simultaneously.

This first constraint makes sure that for any node i, the sum of in-degree edges is equal to the sum of out-degree edges, which should be one. For the special case of the source and destination nodes, we assume one incoming edge into the source and one outgoing edge from the destination. This is mainly used to model that any flow produced by a source node is consumed only by the destination node and no part is consumed by intermediary nodes.

$$\sum_{j\in N} x_{j,i,f} + (\text{if } i = s_f \text{ then } 1) = \sum_{r\in N} x_{i,r,f} + (\text{if } i = d_f \text{ then } 1) \quad \forall \text{i} \in \text{N}, \text{f} \in \text{F} \quad (2)$$

The second constraint stipulates that the exact quantity of SMs given in the demand list should be instantiated.

$$\sum_{i\in N} y_{i,l,f} = d_{l,f} \qquad \forall l \in M, f \in F \quad (3)$$

The next constraint imposes the restriction that any SM can only be instantiated at a given node if resources are actually available there. The term $card(\mathit{CFlow}_l)$ counts the number of flows that need to co-locate their SM l to merge the requests for co-located SMs.

$$\sum_{f\in F} y_{i,l,f} - q_{i,l}\times card(\mathit{CFlow}_l) \le 0 \qquad \forall i \in N, l \in M \quad (4)$$

Moreover, a node should be included in the solution if the SM is to be instantiated at that node. This is stated in the following constraint:

$$\sum_{j\in N} x_{i,j,f} - y_{i,l,f} \ge 0 \quad \forall i \in N \setminus \{s_f\}, l \in M, f \in F \quad (5)$$

Next, we need a constraint to ensure that the allocated bandwidth does not exceed the maximum capacity of links, as follows:

$$\sum_{f\in F} L_f \times x_{i,j,f} \le L_{i,j}^{max} \qquad \forall i \in N, j \in N \quad (6)$$

This constraint avoids forming cycles in the solution path.

$$\sum_{i\in M} x_{i,j,f} \le 1 \quad \forall j \in N, f \in F$$
$$\sum_{j\in M} x_{i,j,f} \le 1 \quad \forall i \in N, f \in F \quad (7)$$

The next constraint states that if a link (i, j) is not visited by a flow f (i.e. $x_{i,j,f} = 0$), the value of $v_{i,j,f}$ should be set to zero, or less than n otherwise.

$$v_{i,j,f} \le n \times x_{i,j,f} \quad \forall i \in N, j \in N, f \in F \quad (8)$$

The constraint below is used to calculate the values of $v_{i,j,f}$ along the path formed by the solution. To take into account the precedence relation between pairs of SMs, we use an integer variable $v_{i,j,f}$ that captures the order of visiting the links by each flow f in the solution. If the link (i,j) is not visited by flow f, then $v_{i,j,f} = 0$. If the link (i, j) is visited before the link (k, r) for a given flow f, we need to be able to verify that $v_{i,j,f} > v_{k,r,f}$. The first link visited after the source node has the largest visiting order value (i.e. $n - 1$). Then, the visiting order of the successor links on the path is decreased by one between any two successive links until the destination node is reached.

$$\sum_{j\in N} v_{j,i,f} + (\text{if } i = s_f \text{ then } n) = \sum_{j\in N}(v_{i,j,f} + x_{i,j,f}) \quad \forall i \in N \setminus \{d_f\}, f \in F \quad (9)$$

This constraint enforces that all $x_{i,j,f}$ and $y_{i,l,f}$ are binary variables and all $v_{i,j,f}$ are integer variables.

$$x_{i,j,f}, y_{i,l,f} \in \{0, 1\}, v_{r,i,f} \ge 0, \quad \forall f \in F, i, j \in N, l \in K_f \quad (10)$$

6.3.2 Security Deployment Constraints

In the following, we present the ILP formulation of the security constraints that we use to implement different NSDPs.

Order Constraint. The following constraint instructs that any SM l listed in K_f should be instantiated by flow f in the order it is requested as specified by $O_{l,f}$. Thus, for any two SMs l and k in K_f with a consecutive order such that $O_{l,f} = O_{k,f} - 1$, SM l should be placed before SM k such that the node i where $y_{l,i,f} = 1$ is visited before the node j, where $y_{k,j,f} = 1$. If this is the case, the constraint verifies that $\sum_{r\in N} v_{r,i,f} \ge \sum_{r\in N} v_{r,j,f}$.

$$n \times (\sum_{a\in N} x_{a,i,f} - y_{i,k,f}) - \sum_{r\in N} v_{r,i,f} + n \ge n \times y_{j,l,f} - \sum_{r\in N} v_{r,j,f}$$
$$\forall f \in F, i, j \in N \setminus \{d_f\}, i \ne j, l, k \in K_f, l \ne k, O_{l,f} = O_{k,f} - 1 \quad (11)$$

Region Constraint. This constraint implements the location-aware NSDP. It allows indicating a specific location at a specific distance from a reference location. For instance, to implement "close to node i'", we need first to define a region of nodes with a certain distance from the reference location i', which we denote by $R_{i'}$. Then, for all flows $f \in F$ that need to instantiate SMs $l \in K_f$ close to i', the following constraint indicates that l should necessarily be instantiated in a node within $R_{i'}$:

$$\sum_{j\in R_{i'}} y_{j,l,f} = 1 \qquad \forall l \in \mathit{RProd}_{i'}, f \in \mathit{RFlow}_{l,i'} \quad (12)$$

Co-location Constraint. The following constraint states that for two different flows $f1$ and $f2$, if $f1$ and $f2$ are in $CFlow_l$, and l is in $CProd$, then both flows should instantiate the SMs l at the same node i.

$$y_{i,l,f1} = y_{i,l,f2} \quad \forall i \in N, l \in CProd, \forall f1, f2 \in CFlow_l, f1 \neq f2 \tag{13}$$

A slight modification to the constraint allows expressing co-location of two SMs for the same flow.

$$y_{i,l,f1} = y_{i,l',f2} \quad \forall i \in N, l, l' \in CProd, l \neq l' \forall f1 \in CFlow_l, f2 \in CFlow_{l'}, f1 = f2 \tag{14}$$

Distribution Constraint. To implement this constraint, we define a set of SMs that need to be distributed $DProd$ and the set of flows that need to distribute each of these SMs, namely $DFlow_l$.

$$y_{i,l,f1} = 1 - y_{i,l',f2} \quad \forall i \in N, l, l' \in DProd, f1 \in DFlow_l, f2 \in DFlow_l, f1 \neq f2 \tag{15}$$

Waypoint Constraint. This constraint enforces the placement algorithm to chose a specific location for a specific SM (e.g. a node that implements SM functionality in a specific hardware). Let i_{wp} be such a location. We define a list of static SMs that are implemented in this location, denoted by $WProd_{iw}$. Also, we define a list of flows that need to put these products in these fixed nodes. The waypoint constraint is expressed as follows:

$$y_{i_{wp},l,f} = 1 \quad \forall l \in WProd_{iw}, f \in WFlow_l \tag{16}$$

7 Experimental Results

7.1 SDPAP Implementation and OpenStack Integration

As a proof of concept, we implemented our algorithms and integrated them into OpenStack[2] and OpendDayLight (ODL)[3]. To realize the service chaining and traffic steering in the network, we use OpenDaylight (ODL). ODL is a SDN controller enabling rich and flexible networking functionalities for SDPAP. The OpenStack Icehouse version[4] is used with a networking plug-in mechanism to connect to the ODL controller. SDPAP prototype consists of four main modules: Interface (SDPAP-I), Manager (SDPAP-M), Evaluator (SDPAP-E), and Orchestrator (SDPAP-O), as seen in Figure 8. The tenant interacts with SDPAP-I to communicate his security needs using a high-level abstract description of the security policies. SDPAP-M translates the abstract security policies to low-level implementation security chains. It uses different NSDPs individually or in combination to implement needed security functionalities. At the end of the translation process, SDPAP-M extracts the service chains and related security deployment constraints. Moreover, SDPAP-M divides each service chain into different security segments.

Moreover, SDPAP-M interacts with OpenStack and ODL to collect networking and computing information, including the network topology and the loads on different nodes and links. Using this information, SDPAP-M builds a graph annotated with costs and availabilities. SDPAP-M applies a partitioning and zoning process to the graph. In this step, SDPAP-M updates the resource availability (SM availability) in each zone of each partition based

2. http://www.openstack.org
3. http://www.opendaylight.org/
4. http://www.openstack.org/software/icehouse/

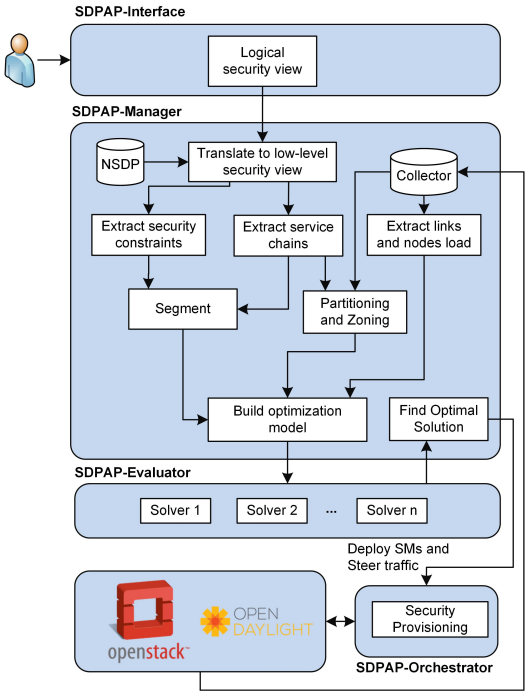


Fig. 8: SDPAP architecture.

on the divided security segments. SDPAP-M then sends different partitioned and zoned graphs along with the extracted service chains annotated with the types, sizes, and precedence of the required SMs to SDPAP-E. The latter runs different instances of a mathematical solver to generate different placement solutions. Finally, SDPAP-M sends the best solution to SDPAP-O, which in turn uses OpenStack and ODL to deploy SMs and steer traffic accordingly. Using this prototype, we deployed all of NSDPs and proved the feasibility of our approach in our cloud test setup.

7.2 Simulation Setup

We used the optimization framework GLPK (GNU Linear Programming Kit) on a machine with 6 cores Intel Westmere (XEON L5638) clocked at 2.30 GHz with 24 GB RAM. For the sake of generality, to get closer to real world data centers, where the CPU load on different hosts and the bandwidth load on the links varies dramatically according to time, applications running in the cloud, etc., we used random values to represent the loads. This also has been applied in other related works (e.g. [10]). Thus, we randomly generated costs of bandwidth units and costs of SMs in a range of [0, 1]. Furthermore, the quantity of SMs available at each node was randomly generated in the interval [0,5] with uniform distribution according to the selected segmentation. We also randomly generated link capacities. The experiments were run for different fat-tree sizes, from 36 nodes for k=4 to 4,260,096 nodes for k=256, with different scenarios. Each simulation was repeated 10 times, and the average is presented.

Note that an implementation of our framework in the cloud should use the cloud management resource tracking capabilities for obtaining these values. In the cloud, elasticity and dynamic nature are among the main cloud characteristics (e.g., scaling in/out the applications, migrating VMs from overloaded computing nodes). The cloud management needs to track the cloud infrastructure resources in real time for providing the resiliency

TABLE 3: Comparing the optimal placement execution time in non-partitioning and partitioning heuristics for different k, for the first scenario

K	Node	Link	Time	
			No-partitioning	Partitioning
4	36	48	0.1s	0s
8	200	384	0.9s	0s
16	1,296	3,072	79.4s	0.1s
24	4,056	10,368	2573.8s	0.1s
32	9,248	24,576	$\geq$ 3h	0.3s
48	30,000	82,944	$\geq$ 3h	1s
64	69,696	196,608	$\geq$ 3h	2.6s
128	540,800	1,572,864	$\geq$ 3h	39.4s
256	4,260,096	12,582,912	$\geq$ 3h	804.8s

and the optimal resource management [5]. For example, in OpenStack the physical and virtual resources data are collected through Ceilometer OpenStack service [6]. Ceilometer measures different resources availability and consumption to the cloud subscribers. Ceilometer measures among others the CPU load, memory usage, and hard disk usage for each computing node, also the network bandwidth for different physical switches and virtual networks. We assume that for an optimal functionality, our platform must be connected to Ceilometer-like services. In the regards to the costs for bandwidth and SMs, they can vary from one platform to another depending on many technical but also commercial and economic factors. A good example of cost structure for SMs can be found from Amazon cost pricing schema [7].

7.3 SDPAP Placement Scalability

We present the feasibility and scalability of optimal placement of NSDP(s), using the scenarios explained in the previous section. As noted before, the optimal placement of a service chain is known to be NP-hard. We show how our heuristics make this problem feasible in a large network consisting of about 4 million nodes.

Table 3 shows the advantage of our first heuristic, which is partitioning. We ran our algorithm, with and without partitioning (no segmentation for both cases) for the first scenario (see Figure 3), for k values from 4 to 256. As can be seen, the execution time after partitioning is remarkably faster. As explained in Section 5.1, a non-partitioning graph in a fat-tree consists of several blocks. Thus, the general optimization problem using ILP cannot scale beyond 1,296 nodes (k=16), which takes 79.4s. Although the improvement in our algorithm execution time is significant with partitioning, we realized that for k values larger than 64 the execution times become inappropriate. This makes our optimization algorithm usage difficult in the cloud infrastructure, where the decision needs to be conducted near real time. Therefore, the segmentation of SMs' availability, as presented next, is needed to improve the execution time after partitioning.

Table 4 illustrates the execution time of SDPAP (i.e. with partitioning and segmentation) for the placement of the four scenarios explained in Section 4.3, for K up to 256. For scenario 1, NSDP-aware segmentation speeds up about 5 times better the execution time for k=48 up to 256. For example, for k=64 the execution time decreases from 2.6s to 0.6s, and for k=128 it goes from 39.4s to 7.7s. In scenario 2, unlike the previous one, the communication between source and destination is unidirectional. As this scenario has fewer number of SMs compared to the previous one and there is a symmetric service chain between the end points, SDPAP scaled for K=256. The execution time for k=128 decreases from 15.2s to 0.5s, and for k=256 it goes from 347.3s to 3.6s.

The scenario 3 (see Figure 5), is a combination of scenarios 1 and 2 when two different requests are demanded by the tenant for securing two different applications at the same time. In this scenario, we assume that the source and destination of both requests are in the same racks. Thus, according to the proposed algorithm, it is possible to run both requests at the same time in different blocks in parallel. Since both requests are executed in the same graph, the same type of SMs between two requests are merged on the basis of the proposed NSDPs. Table 4 shows the feasibility of this complex scenario in run time, too. As seen, the SDPAP algorithm could speed up 5 and 6 times better the execution time. Even the 9.2s execution time for the k=128 can be an acceptable result (it went from 58.2s to 9.2s).

The scenario 4 is about placing 9 SMs for securing a 3-tier application, as illustrated in Figure 6. For the case where each tier is in a specific location (rack), we interpret this as three independent requests in our optimal placement algorithm: *GW-VM(PL)*, *VM(PL)-VM(BL)*, and *VM(BL)-VM(DB)*. Each request consists of two dependent bidirectional flows. All three requests are executed in parallel and the execution time depends to the complex one, which is the security service chain between *GW* and *VM(PL)*. As seen in Table 4, this security defense model takes 15.9s for k=128 to be executed by the NSDP-aware placement algorithm, which is 2.6 times faster than no segmentation.

In spite of the improvements for the case of $K < 128$, the placement execution time is still too high for $k = 128$ and 256; therefore, we consider our NSDP-aware approach to be scalable for up to k=64, with 69,696 nodes, for near real-time demand.

7.4 Comparison between SDPAP Placement and Other Approaches

The need of considering constraints related to the proper functionality of virtual security functions in the context of optimal deployment of virtual security functions has been recognized in many related works. In our approach, we capture the security experts' know-how and best in bread security practice through the usage of NSDPs in order to put in place the application's security posture. These security patterns benefit from the experience from the previous work and experts' experience in the field providing a most effective way to implement security. As for many existing solutions and related work, very few works considered only a single security aspect (e.g. order [6], [7], [13], [14], multi-chain [2], [4] decomposition into concrete functions [7]) leading to the solutions which are optimal though do not cover all the security aspects. Indeed, several other important aspects such location/zone restrictions and collaboration constraints, have not been considered in existent works.

In the last part of this section, we compare the efficiency of NSDP-aware placement to other placement algorithms in terms of network and computing costs. Figure 9 compares 5 different approaches on the basis of a state-of-the art review for scenario 1, the case of a fat-tree of size k=128. All the approaches have only one objective, except for the SDPAP. Max-C and Max-N seek to optimize computing and network utilization, respectively. Max-C only optimizes computing resource utilization without taking into account the routing cost. In contrast, Max-N only optimizes

5. https://www.manageengine.com/products/applications_manager/hyper-v.html

6. https://github.com/openstack/ceilometer

7. https://aws.amazon.com/ec2/pricing/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2616867, IEEE Transactions on Services Computing

TABLE 4: Execution time for with and without NSDP-aware segmentation, for the four scenarios

K	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
	No Seg.	NSDP-aware Seg.	No Seg.	NSDP-aware Seg.	No Seg.	NSDP-aware Seg.	No Seg.	NSDP-aware Seg.
4	0s	0s	0s	0s	0s	0s	0s	0s
8	0s	0s	0s	0s	0s	0s	0s	0s
16	0.1s	0s	0s	0s	0.1s	0s	0s	0s
24	0.1s	0s	0.1s	0s	0.1s	0s	0.2s	0.1s
32	0.3s	0.1s	0.2s	0s	0.4s	0.1s	0.3s	0.1s
48	1s	0.2s	0.7s	0s	1.6s	0.3s	1s	0.4s
64	2.6s	0.6s	1.5s	0.1s	4.5s	0.8s	2.6s	1.3s
128	39.4s	7.7s	15.2s	0.5s	58.2s	9.2s	41.5s	15.9s
256	804.8s	169.6s	347.3s	3.6s	1,152.1s	179.3s	816.6s	181.2s

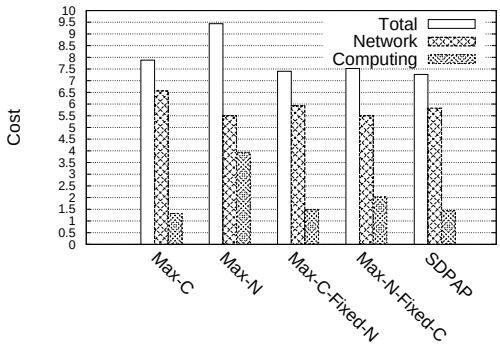


Fig. 9: Comparing five different heuristics in terms of placement cost, for the scenario 1, for the case of a fat-tree of size k=128

network resource utilization. Max-N-Fixed-C assumes specific nodes to deploy SMs, and then it finds the best path to traverse them. In contrast, Max-C-Fixed-N assumes a static route between a source and a destination and then attempts to select the best nodes within static route.

As can be seen in Figure 9, when only one objective is considered, computing or network resource utilization, the cost of the non-considered resource in the respective approach stays high. For example, Max-C has the lowest cost for computing, but has the worst one for network. Similarly, MAX-N, has the lowest cost for network, but has the worst one for the computing. In Max-C-Fixed-N and Max-N-Fixed-C, the costs of network and computing are static, respectively. Thus, if the static cost is not the worst, the total costs of these two approaches are improved but still are higher than SDPAP costs. As can be seen, by considering both objectives, SDPAP is able to attune both costs simultaneously. Thus, our approach has the minimum total cost for each request since the network and computing costs are increased together. If we compare the result of SDPAP with other heuristics for each objective separately, we observe that for network resource utilization, SDPAP is 6% more costly compared to the best case (compare SDPAP network cost, 5.828, to Max-N, 5.499). SDPAP is 9% more costly for computing cost (the value of costs for SDPAP and the best approach, i.e. Max-C, are 1.439 and 1.321, respectively). Even for a simple scenario, scenario 1, which involves only 6 SMs and 2 end points, we see 2% improvement in the total cost of networking and computing resources. This 2% improvement is only for one rather simple placement. We anticipate that in a large data center running 100s of thousands of virtual machines involving thousands of placements the overall gains are much more than this number.

In the end, we have shown that even though the primary goal of the SDPAP approach is to provide better security through the usage of NSDPs, SDPAP improves the scalability of the placement solutions by extending them to fat-trees of size k=64. Furthermore, it presents a more optimal placement solution from the total cost point of view compared to existing approaches .

8 Conclusion

To the best of our knowledge, we are the first to propose a network security pattern based approach for cloud infrastructure and its optimal placement in the cloud based on security deployment constraints. We believe our approach breaks away from the traditional centralized network security approach, i.e. concentrating more and more security functionality in the same appliance in order to have more efficient implementation. We consider building an application targeted security solution by assembling needed security functionality as opposed to starting from the available security capabilities of the purchased security appliances and then trying to retrofit application security needs onto the existing appliance security capabilities. From this perspective, we assemble a set of security functions, deploy them optimally and dynamically and connect them through SDN, all on the basis of the security needs captured through NSDPs. These security modules thereafter can be composed/decomposed inside larger security nodes to achieve a more efficient implementation.

In this paper, we propose a distributed security policy framework, called SDPAP, based on optimal placement of security functionality captured through NSDPs and security deployment constraints. We prototyped our approach using OpenStack and OpendDayLight and evaluated it through simulation for different scenarios. Our results show that our solution is more efficient in terms network and computing resources and scalability for all the covered scenarios. SDPAP implements a more secure solution by deploying tenants' specific security needs using best security practices captured through NSDP. Furthermore, we believe our security solution by providing an optimal placement for finer granularity security functions distributed among less loaded nodes results in a system more resilient to attacks, such as DDoS attacks.

Future work: Our proposed approach as presented in this paper comprises some limitations. For instance, we focused mainly on fat-tree based network topologies as it is widely used in today's data centers. However, we believe that we can adapt our approach to other emerging topologies by considering graph partitioning algorithms. Furthermore, there is a need to automate the translation from the high-level abstract description view into the low-level implementation view of security functions. However, there is extensive work on translating high-level security policy description language to low-level implementation which could be adapted to our approach.

Acknowledgment

This work is partly funded by Natural Sciences and Engineering Research Council of Canada Research Chair on Sustainable Smart Eco-Cloud, NSERC-950-229052 and by the NSERCCRDPJ 424371-11: ECOLOTIC Sustainable and Green Telco-Cloud.

REFERENCES

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[2] W. Liu, H. Li, O. Huang, M. Boucadair, N. Leymann, Z. Cao, Q. Sun, and C. Pham, "Service function chaining (sfc) general use cases," *Work in progress, IETF Secretariat, Internet-Draft draft-liu-sfc-use-cases-08*, 2014.

[3] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 4, pp. 1955–1980, 2014.

[4] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simplefying middlebox policy enforcement using sdn," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 2013, pp. 27–38.

[5] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 24–24.

[6] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," pp. 1–6, 2015.

[7] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, and P. Demeester, "Network service chaining with optimized network function embedding supporting service decompositions," *Computer Networks*, vol. 93, pp. 492–505, 2015.

[8] B. Johnson, "Opnfv sfc," Tech. Rep., https://docs.google.com/presentation/d/1gbhAnrTYbLCrNMhMXin0lxjyg-7IHNPjrlBTIjwAzys/edit?pli=1#slide=id.p.

[9] R. Penno and P. Quinn, "Odl: Service function chaining," Tech. Rep., https://docs.google.com/presentation/d/1JOWN-Xf0_GrV5v_n5q9_j9nUQcM8MjlNl3Sw1YN1bxU/edit?pli=1#slide=id.p4.

[10] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, V. Sekar, and A. Akella, "Stratos: A network-aware orchestration layer for virtual middleboxes in clouds," *arXiv preprint arXiv:1305.0209*, 2013.

[11] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan *et al.*, "Steering: A software-defined networking for inline service chaining," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 2013, pp. 1–10.

[12] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," 2015.

[13] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 2015, pp. 50–56.

[14] Y. Jarraya, A. Shameli-Sendi, M. Pourzandi, and M. Cheriet, "Multistage ocdo: Scalable security provisioning optimization in sdn-based cloud," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 572–579.

[15] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402967

[16] S. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 4, pp. 2046–2069, Fourth 2013.

[17] A. Gordon, *The Official (ISC)2 Guide to the CCSP CBK*. Wiley / Sybex, 2015.

[18] V. Sekar, R. Krishnaswamy, A. Gupta, and M. K. Reiter, "Network-wide deployment of intrusion detection and prevention systems," in *Proceedings of the 6th International COnference*. ACM, 2010, p. 18.

[19] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes." in *NSDI*, 2013, pp. 227–240.

[20] A. R. Khakpour and A. X. Liu, "First step toward cloud-based firewalling," in *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*. IEEE, 2012, pp. 41–50.

[21] B. Hedlund, "What is a distributed firewall?" VMWare, 2013.

[22] J. Xu, J. Yan, L. He, P. Su, and D. Feng, "Cloudsec: a cloud architecture for composing collaborative security services," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 703–711.

[23] A. Shameli-Sendi, M. Pourzandi, M. Fekih-Ahmed, and M. Cheriet, "Taxonomy of distributed denial of service mitigation approaches for cloud computing," *Journal of Network and Computer Applications*, vol. 58, pp. 165–179, 2015.

[24] S. T. Zargar, H. Takabi, and J. B. Joshi, "Dcdidp: A distributed, collaborative, and data-driven intrusion detection and prevention framework for cloud computing environments," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*. IEEE, 2011, pp. 332–341.

[25] N. Pappas, "Network ids & ips deployment strategies," *2nd April*, 2008.

[26] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014, pp. 7–13.

[27] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, "xomb: Extensible open middleboxes with commodity servers," in *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*. ACM, 2012, pp. 49–60.

[28] R. Alshahrani and H. Peyravi, "Modeling and simulation of data center networks," in *Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM-PADS '14. New York, NY, USA: ACM, 2014, pp. 75–82. [Online]. Available: http://doi.acm.org/10.1145/2601381.2601389

[29] Z. Cao, M. Kodialam, and T. V. Lakshman, "Traffic steering in software defined networks: Planning and online routing," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing*, ser. DCC '14. New York, NY, USA: ACM, 2014, pp. 65–70.

Alireza Shameli-Sendi is currently an Assistant Professor at Shahid Beheshti University. Before joining SBU, he was Postdoctoral Fellow at Ericsson, Canada and Postdoctoral at ETS and McGill universities in collaboration with Ericsson. He received his Ph.D degree in computer engineering from Ecole Polytechnique de Montreal, Canada. He obtained his B.Sc. and M.Sc. from Amirkabir University of Technology. His primary research interests include information security, intrusion response system, and cloud computing. He is a recipient of Postdoctoral Research Fellowship Award and Industrial Postdoctoral Fellowship Award from Canada.

Yosr Jarraya is currently a Researcher at Ericsson, Canada. She was previously Research Associate and Postdoctoral Fellow At Concordia University, Montreal. She received a Ph.D. in Electrical and Computer Engineering from Concordia University and a M.Sc. in Telecommunications from Ecole Superieure des Communications de Tunis (SupCom), Tunisia. Her research interests include cloud computing, software-defined networking, and security.

Makan Pourzandi is a researcher at Ericsson, Canada. He received his Ph.D. degree in Computer Science from University of Lyon, France and his M.Sc. Degree in Computer Science from Ecole Normale Superieure de Lyon, France. He co-authored a book, has 11 granted patents and more than 50 research papers in journals and conferences. His current research interests include security, cloud computing, software security engineering, and cluster computing.

Mohamed Cheriet received his M.Sc. and Ph.D. degrees in Computer Science from the University of Pierre et Marie Curie (Paris VI) in 1985 and 1988 respectively. Dr. Cheriet is expert in cloud computing and network virtualization. In addition, he is an expert in Computational Intelligence, Pattern Recognition, Mathematical Modeling for Image Processing, Cognitive and Machine Learning approaches and Perception. Dr. Cheriet has published more than 300 technical papers in the field. He holds Canada Research Chair Tier 1 on Sustainable Smart Eco-Cloud.