A Scalable Stateful Approach for Virtual Security Functions Orchestration

Niloofar Moradi, Alireza Shameli-Sendi*, and Alireza Khajouei
Faculty of Computer Science and Engineering, Shahid Beheshti University (SBU), Tehran, Iran
Email: {nil.moradi@mail.sbu.ac.ir,a_shameli@sbu.ac.ir,a.khajooyee@mail.sbu.ac.ir}

Abstract—Previous works suggested different approaches to implementing service chaining. Their goal is to enhance the performance of the middleboxes and satisfy the expectations of the cloud providers and users. To meet these expectations, the delay factor, i.e., flow through the low-cost paths, as well as the best node processing factor, are considered. Achieving these two goals simultaneously turns the middlebox optimal placement into an NP-hard problem. Therefore, when the problem size is large, it is infeasible to obtain an optimal solution at a reasonable time. One of the important issues which has not been considered in the previous works is stateful optimal placement when receiving a new request. Due to resource constraints as well as financial costs for the customers, it is not possible to create functions for all requests. Therefore, not only it is possible to integrate the same network functions between new flows, but it will also be examined between new on-demand network functions as well as existing ones. Our proposed approach not only reduces the creation of network functions that can be cost-effective for the customer but also because of the migration of previous network functions (integration with on-demand network functions) to optimize new requests, overall, it will optimize the entire network cost over time. We formulated the problem as 0-1 programming problem. The results of this paper are based on a fat-tree data center. To show that our stateful solution is scalable in large networks, we use network zoning and topology partitioning heuristics. Our simulations show that we were able to scale our placement model to a network with 54K nodes and 1.5M edges.

Index Terms—NFV, Security functions, Service chaining, Optimal placement, Stateful placement.

1 Introduction

Network Function Virtualization (NFV) is a paradigm to facilitate dynamic provisioning of network services through virtualization technologies [1], [2]. In this scene, network services can be implemented by software modules on hardware with virtualization capability, i.e., Virtual Network Functions (VNFs) [3]. Software-Defined Networking (SDN) [4], [5], on the other hand, complements NFV and provides dynamic traffic engineering and dynamic management facilities between virtual functions. This happens through full programmability of forwarding capabilities [6], [7].

Software middleboxes such as Intrusion Detection System (IDS), Firewall (FW), Deep Packet Inspection (DPI), Traffic Monitoring, Load Balancer (LB), etc. are widely used in the cloud to detect and counter attacks [8]–[10]. They are deployed/destroyed dynamically based on cloud user application needs [11], [12]. For example, assume a cloud user has a distributed cloud-based application, and suddenly its performance goes down. At this moment, several IDSs can be deployed in different locations in the cloud as a collaborative detection strategy [13] to see if a distributed denial-of-service attack is occurring or not. As mitigation solutions, we may deploy either a load balancer to clone some components of a distributed application or several FWs on the border of the cloud to filter some traffic. After some time, if the traffic was normal, we can destroy all created middleboxes, IDSs, LB, and FWs. Note that in this paper, we focused on east-west traffic to secure the communication of different applications

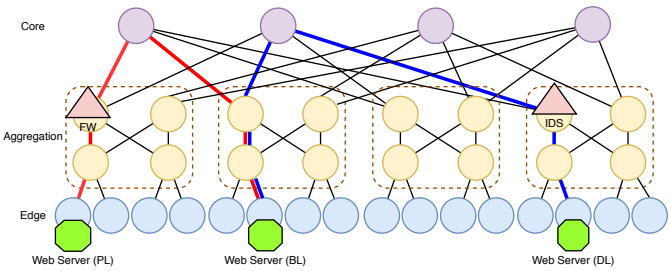


Fig. 1: Fat-tree K=4, a distributed application and two network functions (FW and IDS)

with dynamic placement of demanded middleboxes as depicted in Figure 1.

The service chain consists of a series of middleboxes that flows must pass through them in a specific order [3]. As a general picture, a service chain is needed to be deployed for traffic, between two VMs, to manage users' requests, improve performance, or increase security. Since the cloud data center hosts thousands of users and applications, the real-time demand for service chains creation or deletion is high. Therefore, it is essential to improve the performance of these middleboxes and enhance the quality of the services [14], [15].

In recent years, many efforts have been performed to tackle the middleboxes placement optimally such that the policy enforcement (e.g., inline service sequence in a service chain) is satisfied [16]–[18]. The two main objectives in optimal placement are minimizing traffic end-to-end delay and processing cost on network switches. End-to-end delay is the sum of delay through all links of the path [19], [20]. The link delay can be modeled based

• *Alireza Shameli-Sendi (Corresponding author).

on four fundamental factors: i) current available bandwidth, ii) consumed bandwidth, iii) the number of flows steering through the link, and iv) sending rate of each steered flow. Moreover, choosing an efficient location to deploy the middlebox in the network is compassed with: i) the node's available space (vCPU and vMem), ii) its resource consumption, iii) the current active middleboxes, and iv) the input traffic rate to each middlebox. We believe that formulating the cost of nodes and edges according to the above parameters requires independent research and is beyond the scope of this paper.

Our optimal middlebox placement method considers both objectives simultaneously, where it selects efficient switches to deploy middleboxes while routing the traffic through the links with the lowest end-to-end delay. The remarkable point is that addressing one of the objectives alone necessarily does not lead to accomplishing another one. Thus, addressing both objectives is our goal. In addition to previous works, we also consider another goal that optimizes the placement with considering available deployed middleboxes. To meet this goal, the job collocation operation is performed between requested security functions and current ones.

Combining similar functions will have many benefits [21]. For example, we can reduce the cost of installation or resource consumption such as CPU and memory or shared storage space. Therefore, it is very important that we can break the security functions into pieces that have similar tasks between them so that we can take advantage of the job collocation. This may cause concern for different customers, but for each customer, the collocation policy can be easily applied. The job collocation operation can be done either among the new requests or between the new requested security functions and the currently active ones. We call our solution stateful placement which differentiates our work from previous ones.

Generally, the middlebox placement problem itself is an NP-hard problem and in particular, most recent works are not scalable in the large network. Therefore, applying an stateful approach to common placement problem also makes it harder to be solved in real-time for current data centers. Several heuristics have been used to overcome the scalability problem. In general, the main contributions of the paper are as follows:

- Unlike previous works in this area, the objectives of our optimal placement are based on five main objectives: i) minimizing end-to-end delay, ii) minimizing the switch processing cost, iii) minimizing the flow impact on network by collocating the flow's security functions on the same switch. In this way, we only pay just one cost (message parsing and needed pre-processing) per-flow in the network, iv) minimizing the number of the same type of security functions and related costs inside the new requests. This is done by integrating the same type on the same switch, v) reducing the cost of creating new requested security functions by taking into account the existing functions. This can be done by examining the current capacity of the currently deployed functions and optimizing the routing of the requested traffic to that area, or migrating the current function to another optimal area.
- To the best of our knowledge, we are the first to formulate the stateful approach for service chaining placement as 0-1 programming problem. As much as possible, the creation of new request security functions is prevented. This is done by checking the possibility of integration of the same type of security functions.
- The applicability of the placement problem is very important for large networks. So we have used some heuristics (i.e. restricting the old function migration area, limiting the area of new functions creation), and our approach is scalable to a network with 54K nodes and 1.5M edges.

The rest of paper is organized as follows. Section 2 presents the related work. Section 3 explains the problem we want to solve and explains the advantage of using an stateful placement approach with a simple example. Section 4 presents how to model an stateful placement problem using integer linear programming. Section 5 first shows the results in a simple fat-tree topology, comparing stateful and stateless placement approaches, and then evaluates how our model can be applied to large data centers. Finally, in Section 6 we conclude our work.

2 Related Work

In recent years, NFV has received enormous research efforts. The optimal location of network functions has been widely welcomed in the industry and the university. The optimization objectives that are common to all works are to reduce the delay between the origin and destination of a flow and place the network functions in the most optimal nodes. Considering these two objectives together makes the problem NP-hard [22]. Many early works have just modeled it as an integer linear program (ILP) [16], [17]. After that, many researchers focused on the issue of scalability problem [23]–[25], since the execution time of the algorithm was not acceptable to large networks.

The goal of the placement algorithm proposed by Eramo et al. [26] is to reduce server usage which ultimately leads to reduced energy consumption. Hawilo et al. [27] have addressed another issue, and that is the reduction of delays between dependent VNFs. Manias et al. [28] used machine learning techniques to solve the placement problem. In this way, they try to reduce the delay in the next placements.

In [29], they proposed to consider different options of transforming abstract network functions like a firewall to implementation-close function like openflow before executing an optimal placement algorithm. They believe that this operation of breaking down and bringing functions closer to reality will be a great help in correct estimation of resources and the correct selection of physical servers. They used Interoute topology, an international telecommunications service provider, which consists of 110 nodes and 148 edges. To reduce the cost of placement, Woldeyohannes et al. [23] suggested that we categorize the flows according to the proximity of their paths. It helps to reduce the creation of similar network function instances. They experimented with a practical ISP network topology, the Rocketfuel topology [30] with 100 nodes and 294 links.

Mohammadkhan et al. [24] proposed that instead of solving all the requests at once, we break them down into smaller parts and then execute them in sequence. The network topology of AS-16631 from Rocketfuel, with 22 nodes and 64 links was used in [24]. Liu et al. [3] proposed a greedy algorithm and a simulated annealing approach to tackle the scalability issue. They used the well-known topologies such as Abilene, 11 nodes and 14 links, and fat-tree, 320 nodes and 2,432 links, for performance evaluation. In our previous work [25], to tackle scalability, the graph from the origin to the destination of the flows were partitioned into several sub-graphs and then the resource availability in them

was limited so that a specific function can only be established in the appropriate sub-graph according to the task it performs. In [25], experiments were performed with different size of fat-tree topologies varied between 36 to 4,260,096 nodes and 48 to 12,582,912 links. In another of our recent works [21], a new idea of placement cost reduction has been proposed, introducing two other objectives in addition to the objectives of previous works. One goal is to reduce the creation of similar functions between requests received together, and the other is to reduce the cost of pre-processing related to the common functions of a flow. In [31] only the reduction of the number of virtual functions and their integration as much as possible is considered for new requests. Moreover, balancing the resources in the network has not been one of their goals.

In our approach, we captured different problem, minimizing the overall resource consumption through an stateful placement approach. Our solution also takes into account all previously deployed network functions when processing a new request. Therefore, as far as possible, it refrains from creating new functions. For our stateful approach to have a real-time response in a large network, we assume that current network flows cannot optimize their path. Therefore, previous network functions created in the network can only migrate in their own flow path. The idea of zoning has also been applied to improve response time. This idea limits the places where network functions can be located.

3 Problem Statement

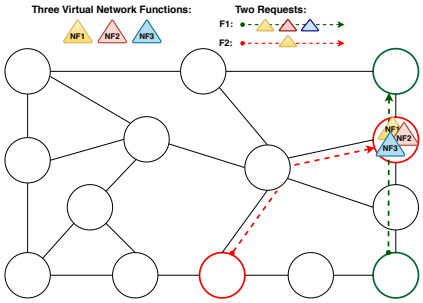
To secure the communication between VMs, we may deploy several security middleboxes (e.g. Firewall, IDS, VPN) dynamically over time. For example, we may deploy a FW layer three and an IDS between a web application (VM1) and a database (VM2). Since there are two traffics between VM1 and VM2 (VM1 to VM2 and vice versa) and each traffic needs to meet, first, firewall and then IDS, there are at least two and at most four security middleboxes in this case.

If we collocate all security middleboxes of a flow on a node, only one pre-processing cost is paid for all of them. This strategy of deployment is called ”parser collocation” [21]. So, in the parser collocation, each traffic desires to collocate all security middleboxes on a node, if resource limitation is satisfied. In the aforementioned example, based on the parser collocation strategy, we will have two nodes, each consists of two VMs (FW and IDS).

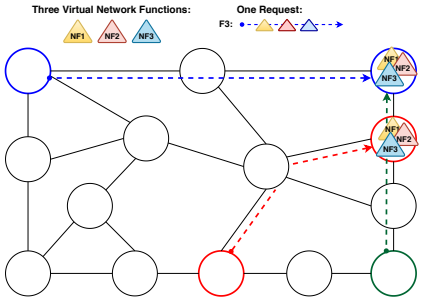
In reality, it is not possible to create the unlimited same type of security function for each traffic because of budget limitation. As explained, in our example, the parser collocation strategy leads us to create two FWs and two IDSs, for two traffics. We may re-steer the new request to the already deployed middlebox in case of the same functionality. This strategy is called ”job collocation” [21]. Therefore, in this strategy, we may see several traffics collocate the same type of middleboxes on a VM while they may traverse separate paths to visit other middleboxes on different locations. In the aforementioned example, based on the job collocation strategy, we will have two nodes, each consists of only one VM (one FW and one IDS).

Note that if these two strategies are applied at the same time, there is only one node in the network which hosts two VMs (FW and IDS).

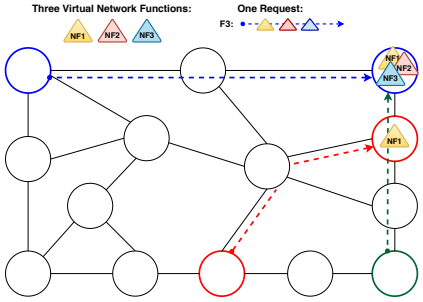
In this section, we want to explain the case that in optimizing the new placement, the previous optimization situation needs to be considered as one of the optimization algorithm input. Imagine, in



(a) Initial optimal placement for two flows with four network functions



(b) Non-inremetal placement



(c) stateful placement

Fig. 2: Motivating example: Comparison of placement in stateless and stateful approaches. In the first placement when the network is empty, two requests, f_1 and f_2, come with the creation of four network functions. Figures (b) and (c) show the cost difference in the placement for processing a new request, f_3 to create three functions.

new request, some new flows ask for a set of security functions which have been previously placed in the network. Therefore, the solver has this chance to collocate the requested security functions with existed ones (for the same type), if the optimization cost is the lowest among all solutions.

Note that the majority of existing solutions does not integrate the result of new placement and previous ones. We call it *stateless* placement. Therefore, it applies high cost to the network, since many security functions should be deployed to the network in each placement. If the optimization is carried out by integrating new and previous requests, called *stateful* placement, the new flows and requested security functions may distribute in the network and collocate with already deployed security functions. Therefore, stateful placement saves many resources in the network. Note that in this approach, maybe there is not any parser or job collections among the new flows in current placement, while in the stateless placement, it happens certainly since the solver should minimize

the optimization cost among the new flows.

The optimal algorithm can make different decisions on stateful approach: 1) A decision can be re-routing the new flows through the already deployed security functions in the network, without creating any functions, 2) Another decision can be to migrate the current security functions to a particular location and then pass the related previous and current flows through them, 3) Another solution would be creating the same type of security functions in the network and then passing all new flows and some of the previous ones through them.

We now illustrate the advantages of stateful placement by presenting a simple example. Figure 2a shows the initial placement state in which there are two flows, with requests for three middleboxes NF_1, NF_2, and NF_3 for flow f_1 and middlebox NF_1 for second flow. Assume that the cost to create VM is \$2, the cost to pre-processing messages for parser collocation is \$1, and the cost in case of per-middlebox collocation is \$0.5. Note that, job collocation cost reduction should be adjusted by the cloud provider. In the best case, we can assume that only one VM is needed for all the same functions. In reality, a VM cannot handle the traffic from the all related flows. In this example, we assume that in job collocation only one-quarter of the cost of a VM is reduced compared to the cost of creating it. In the initial placement there is four VM creation, two parser collocation, and one job collocation as depicted in Figure 2a. Therefore, the related cost is \$9 ($\underbrace{4 \times \$2}_{\text{VM creation}} + \underbrace{2 \times \$1}_{\text{parser}} - \underbrace{2 \times \$0.5}_{\text{job earned}}$).

Let us suppose that there is a request for a new flow, f_3, to create three new middleboxes (NF_1, NF_2, and NF_3). If we perform the placement based on stateless approach, because it does not consider the current middleboxes in the network, there is no optimization between the current request and the previous ones. At best, to reduce costs, the three requested middleboxes are created in one node, as depicted in Figure 2b. So the set-up cost for the third flow is \$7 ($\underbrace{3 \times \$2}_{\text{VM creation}} + \underbrace{1 \times \$1}_{\text{parser}}$).

If the placement is based on the approach presented in this paper, the cost will be different from that of the stateless approach. As shown in Figure 2c, the solver answer is that an optimal node is found so that the new flow and the first flow middleboxes can perform the job collocation operation. Although in this approach, the advantages of job collocation are obtained between new and old requests, there may also be costs associated with migrating existing middleboxes from one node to another. As can be seen in the figure, old flows, such as flow f_2, which previously had the advantage of job operation over middlebox NF_1, generate new costs as the NF_1 remains alone in the same node. Also, the first flow must pay a parser fee because of the migration of its function to a new node. So the set-up cost for the third flow is \$6 ($\underbrace{3 \times \$2}_{\text{VM creation}} + \underbrace{2 \times \$1}_{\text{parser}} - \underbrace{6 \times \$0.5}_{\text{job earned}} + \underbrace{2 \times \$0.5}_{\text{losing job}}$). So our approach, in comparison to the previous one, was able to reduce the cost by \$1 in a small example and in the first placement. This amount of cost can increase greatly with time and many requests.

4 Proposed Solution

The model presented in this paper is based on integer linear programming formulation. We extend the formulas suggested in our previous published paper for the stateful problem [21]. Therefore, in the following, the objective function, variables, and constraints of the problem will be explained in detail.

4.1 Notations and variables

Table 1 shows the inputs of the stateful placement problem. In the following, we will explain each of the parameters.

System configuration. $N = \{1, \cdots, n\}$ shows all nodes in the network. Since our network is a fat-tree topology, it consists of three layers of switches (edge, aggregation, and core) and one layer of servers. So N will include all the layers. L includes all links, (i, j), in the network topology. Our assumption in this paper is that nodes and edges are not damaged, so their number is constant in all placements. Two parameters, $L_{i,j}$ and n_i, in our problem are updated in each placement. The former indicates the current bandwidth of each edge in the network and the latter indicates the current capacity in each node.

Our placement algorithm selects links and nodes with more available bandwidth and more capacity. Note that the only criterion for selecting the edge is not the current bandwidth, but the number of flows passed and their traffic rate are also important criteria. In terms of nodes, the number of processors and more memory are the criteria, but the created functions in the node and the traffic rate that is entered into them must also be considered. In this paper, a formula for combining these criteria for the cost of edge and node is not provided. We believe that a comprehensive investigation is needed. Therefore, the only criterion for the cost of the edge is the current bandwidth and the only criterion for the node is the current capacity of the node.

Application. Because we want to solve the stateful placement problem, there are two lists of flows in our problem: i) $F = \{1, \cdots, h\}$ and ii) $F' = \{1, \cdots, h'\}$. The former is related to the current requests and the latter is related to the previous requests that is currently available on the network. For each flow, we need to specify the start $s_f \in N$ and end $d_f \in N$ nodes. Since our simulation network is a fat-tree network, the start and end nodes are the first layer nodes, the leaf nodes of the tree. Each flow $f \in F$ needs to specify a bandwidth requirement b_f. The number of flows in our problem depends on the type of application architecture. For example, suppose we have a web-based system based on three-layer architecture: *presentation layer (PL)*, *business layer (BL)*, and *data layer (DL)*. Imagine, we need a security function for every connection. So there will be four connections in our problem: i) *PL* to *BL* , ii) *BL* to *DL* , iii) *DL* to *BL*, and iv) *BL* to *PL*.

Security Functions. $M = \{1, \cdots, m\}$ represents a list of all security functions. Each flow can select a set or all of them depending on the type of flow and its application. Like [21], we consider two types of costs for each function: i) *per-flow*, c^p and ii) *per-function*, c^j_m. c^p indicates the pre-processing costs. The cost of pre-processing (parsing and decomposing messages) is determined by the flow, regardless of how many security functions a flow has requested. But it is important to note that if all the functions of a flow are in a node, we have to pay a pre-processing cost, but if they are distributed in different nodes, we have to pay an independent cost for each function in each node. So a concept called *per-flow collocation* or *parser collocation* is formed. The implication is that our placement algorithm tries to put all the functions of a flow in a node as much as possible in order to reduce the placement cost so that it can reduce the pre-processing cost.

Each security function, $m \in M$, also has its own cost after pre-processing, and that cost is related to the particular job of the function, which is called *per-function* cost, c^j_m. This type of cost can also be reduced if similar security functions related to different

TABLE 1: All the inputs needed to solve the stateful optimization problem.

Network	$N = \{1, \cdots, n\}$	set of network nodes
	$L = \{(i,j)\}$	set of physical links in the network
	$l_{i,j}$	current maximum available capacity in units of bandwidth in link (i,j). It should be updated after each placement
	n_i	current maximum processing capacity (i.e. vCPU, vMem) of node i available for function. It should be updated after each placement
Flows	$F = \{1, \cdots, h\}$	set of new requested flows
	s_f	flow source node, $s_f \in N$
	d_f	flow destination node, $d_f \in N$
	b_f	number of units of bandwidth needed by a flow f
	$F' = \{1, \cdots, h'\}$	set of current flows in the network
	s'_f	old flow source node, $s'_f \in N$
	d'_f	old flow destination node, $d'_f \in N$
	$b_{f'}$	number of units of bandwidth needed by a flow f'
Middleboxes	$M = \{1, \cdots, m\}$	set of possible security functions
	c^p	cost for pre-processing messages (per bandwidth unit)
	c^j_m	function-specific cost for security function $m \in M$ (per bandwidth unit)
	P	cost reduction factor in case of *per-function* collocation
	$r_{f,m}$	set to 1 if security function $m \in M$ is required for $f \in F$
	sa_f	sampling rate for flow $f \in F$ as given by application
	$r_{f',m}$	set to 1 if security function $m \in M$ is required for $f' \in F'$
	$sa_{f'}$	sampling rate for flow $f' \in F'$ as given by application
	$k_{i,f',m'}$	set to 1 if security function $m' \in M$ has been placed in node i for $f' \in F'$ in previous placement
	$p_{i,f'}$	the previous requested flows' routing paths
	$t_{f',m'}$	set to 1 if security function $m' \in M$ required by $f' \in F'$ has job collocation in previous placement

TABLE 2: Output of the stateful optimization problem.

$x_{i,j,f}$	routing decision variable for link $(i,j) \in L$ and flow f
$y_{i,f,m}$	security function placement decision variable for security function m for node i of flow f
$y'_{i,f',m'}$	previously deployed security function placement decision variable for security function m' for node i of flow f'

flows are placed in a node. So there is a concept called *per-function collocation* or *job collocation* in our model. The function-specific cost is defined as $c^j_m \times b_f \times sa_f$, in which b_f expresses the flow requested bandwidth and sa_f represents the flow sampling rate. The amount of job collocation reduction that similar functions create in a node is based on the P parameter. For example, if two similar functions are in a node and the value of P is fifty percent, the cost of each function is reduced by half.

To find out which of functions in M each flow wants, we use the matrix R. Therefore, the binary variable $r_{f,m}$, which is based on R, indicates whether the flow f requires function m.

After executing the placement problem, we will have three outputs. The first output is binary variable $x_{i,j,f}$, which determines which routes are selected from the origin of a flow f to its destination. The second output, binary variable $y_{i,f,m}$, indicates the optimal locations to create functions. Note that the selected locations do not always mean function creation. Rather, it sometimes indicates the use of an old function that exists in that location or migrated ones to that location. Another important point is that the location of the functions must be along the flow path. The third variable, $y'_{i,f',m'}$, represents the migration of old functions. That is, if an old function is positioned in a new location, the value of the variable will be one.

4.2 Objective Functions

We formalize our optimization as seven objective functions:

1) For the first objective, we look at minimizing traffic end-to-end delay. The cost of the edge is modeled based on its bandwidth. That is, the higher the bandwidth of an edge, the lower its cost ($\frac{1}{l_{i,j}}$). It is obvious that the selection of high bandwidth edges for a flow reduces the end-to-end delay.

2) The second objective is looking at minimizing processing cost, appropriate nodes for hosting security functions. If we assume that the available resources in the node i are n_i, the cost of a node is modeled as $\frac{1}{n_i}$. With this, selecting nodes with a lot of processing capacity is favored. Note that function-specific costs (c^j_m) along with the bandwidth needs and sampling rate of the flow ($b_f \times sa_f$) indicates the function cost in this part.

3) In the third objective, *per-flow* collocation is considered. This type of cost reduction is related to the functions belonging to a flow. If a flow distributes its own functions in different nodes, a pre-processing cost must be paid for each function. Conversely, if all functions are to be created in a node, then we need to pay only one pre-processing cost. Therefore, for different modes of distribution of functions related to a flow, different pre-processing costs must be paid.

4) For the fourth objective, *per-function* collocation is considered. This type of cost reduction is related to similar functions related to different flows. In other words, instead of creating a large number of similar functions and paying for the installation and other costs of related virtual machines, we can reduce this number even to the cost of one virtual machine (at best). We have modeled the percentage reduction of each function by considering the P variable, which depends on the rate of traffic received and the type of function processing. So in our model, a new function in three conditions can perform the job collocation operation: i) with new similar functions requested in the same node ($\sum_{f \in F} y_{i,f,m} > 1$), ii) with old functions migrating to the same node ($\sum_{f' \in F'} y'_{i,f',m} \geq 1$), iii) with current functions in the same node ($\sum_{f' \in F'} (k_{i,f',m} \cdot \sum_{i \in N} y'_{i,f',m}) \geq 1$)).

5) In the fifth objective, we check whether the old functions that have migrated to the current node have performed the collocation operation, in which case the placement cost should be reduced. Collocation, in this case, can occur i) between the migrated functions, ii) between the migrated and new function, and iii) between the migrated function and the current function in the node. The important point is that this cost reduction should be considered when the old function in the previous placement failed to perform the job collocation ($t_{f',m'} = 0$).

6) The sixth objective considers a penalty for old functions if they have performed job collocation in the previous placement ($t_{f',m'} = 1$), but now either the function is left alone or it has moved to a new node where the same function is not present in that node.

7) The seventh objective is to check the cost of pre-processing for old functions that may be applied to the new placement. For example, suppose that in the previous placement, all the functions related to a flow were placed in a node, and we paid only one pre-processing cost. Now, in the new location, all old functions have migrated to new and separate nodes in order to perform job collocation with the new requested functions. Therefore, for each migration ($\sum_{m' \in M} {y'}_{i,f',m'} \geq 1$), a parser cost must be considered. As the formula shows, this cost is considered in a new node if there is no function of that corresponding flow in that node ($\sum_{m' \in M} k_{i,f',m'} \leq 0$).

$$
\begin{aligned}
min \quad & \sum_{i \in N} \sum_{j \in N} \sum_{f \in F} \frac{1}{l_{i,j}} \cdot b_f \cdot x_{i,j,f} \quad + \\
& \sum_{i \in N} \sum_{f \in F} \sum_{m \in M} \frac{1}{n_i} \cdot c_m^j \cdot b_f \cdot sa_f \cdot y_{i,f,m} \quad + \\
& \sum_{i \in N} \sum_{f \in F} \frac{1}{n_i} \cdot c^p \cdot b_f \cdot sa_f \cdot U_{if} \quad - \\
& \sum_{i \in N} \sum_{f \in F} \sum_{m \in M} \frac{1}{n_i} \cdot c_m^j \cdot b_f \cdot sa_f \cdot P \cdot V_{ifm} \quad - \\
& \sum_{i \in N} \sum_{f' \in F'} \sum_{m' \in M} \frac{1}{n_i} \cdot c_{m'}^j \cdot b_{f'} \cdot sa_{f'} \cdot P \cdot V'_{if'm'} \quad + \\
& \sum_{i \in N} \sum_{f' \in F'} \sum_{m' \in M} \frac{1}{n_i} \cdot c_{m'}^j \cdot b_{f'} \cdot sa_{f'} \cdot P \cdot W'_{if'm'} \quad + \\
& \sum_{i \in N} \sum_{f' \in F'} \frac{1}{n_i} \cdot c^p \cdot b_{f'} \cdot sa_{f'} \cdot U'_{if'}
\end{aligned}
$$

$$
U_{if} = \begin{cases} 1 & if \sum_{m \in M} y_{i,f,m} \geq 1 \\ 0 & otherwise \end{cases}
$$

$$
V_{ifm} = \begin{cases} 1 & if((\sum_{f \in F} y_{i,f,m} > 1 \ \| \\ & \sum_{f' \in F'} y'_{i,f',m} \geq 1 \ \| \\ & \sum_{f' \in F'} (k_{i,f',m} \cdot \sum_{i \in N} {y'}_{i,f',m}) \geq 1) \\ & \ \& \ (y_{i,f,m} = 1)) \\ 0 & otherwise \end{cases}
$$

$$
{V'}_{if'm'} = \begin{cases} 1 & if((\sum_{f \in F} y_{i,f,m'} \geq 1 \ \| \\ & \sum_{f' \in F'} y'_{i,f',m'} \geq 1 \ \| \\ & \sum_{f' \in F'} (k_{i,f',m'} \cdot \sum_{i \in N} {y'}_{i,f',m'}) \geq 1) \\ & \ \& \ (t_{f',m'} = 0) \ \& \ (y'_{i,f',m'} = 1)) \\ 0 & otherwise \end{cases}
$$

$$
W_{if'm'} = \begin{cases} 0 & if((\sum_{f \in F} y_{i,f,m'} \geq 1 \ \| \\ & \sum_{f' \in F'} y'_{i,f',m'} \geq 1 \ \| \\ & \sum_{f' \in F'} (k_{i,f',m'} \cdot \sum_{i \in N} {y'}_{i,f',m'}) \geq 1) \\ & \ \& \ t_{f',m'} = 1) \\ 1 & otherwise \end{cases}
$$

$$
{U'}_{if'} = \begin{cases} 1 & if(\sum_{m' \in M} {y'}_{i,f',m'} \geq 1 \\ & \ \& \quad \sum_{m' \in M} k_{i,f',m'} \leq 0) \\ 0 & otherwise \end{cases} \tag{1}
$$

4.3 Constraints

The following statement indicates that the cycles must be avoided from the origin of a flow f to its destination. Therefore, the number of entries to a node and the number of exits from a node must be zero or one. That is, we have either not passed a node, and if we pass, the number of entries is one or the number of exits is one.

$$
\begin{aligned}
\forall j \in N, f \in F: \quad & \sum_{i \in N} x_{i,j,f} \leq 1 \\
\forall i \in N, f \in F: \quad & \sum_{j \in N} x_{i,j,f} \leq 1
\end{aligned} \tag{2}
$$

The next constraint ensures that the path of a flow from beginning to end is a continuous path. In two exceptions, the source node s_f and the destination node d_f of a flow f, an input edge and an output edge are assumed.

$$
\begin{aligned}
\forall i \in N, \forall f \in F: \quad & \sum_{j \in N} x_{j,i,f} + (1 \text{ if } i = s_f) = \\
& \sum_{r \in N} x_{i,r,f} + (1 \text{ if } i = d_f)
\end{aligned} \tag{3}
$$

The next constraint verifies that the total bandwidth requested by all currents passing through a link does not exceed its current capacity.

$$
\forall i, j \in N : \sum_{f \in F} b_f . x_{i,j,f} \leq l_{i,j} \tag{4}
$$

We have a list of security functions for the problem we want to solve. Each flow may want a set of that list. Therefore, it does not require the creation of all functions, and it must be based on the required matrix, R. Therefore, for each function, the sum of all the values of $y_{i,f,m}$ in the entire network must be equal to the requested value, $r_{f,m}$.

$$
\forall m \in M, \forall f \in F: \quad \sum_{i \in N} y_{i,f,m} = r_{f,m} \tag{5}
$$

We need a constraint to ensure that the requested functions of a flow are created along that path, from origin to destination. If this is not the case, the solver will go through a cheap route to reduce the cost of placement and then create the requested functions on the optimal nodes that are not necessarily part of the related flow path.

$$
\begin{aligned}
& \forall i \in N \setminus \{s_f, d_f\}, m \in M, f \in F: \\
& \sum_{j \in N} x_{i,j,f} - y_{i,f,m} \geq 0
\end{aligned} \tag{6}
$$

$$
\begin{aligned}
\forall i \in N : & \sum_{f \in F} \sum_{m \in M} (c^p + c^j_m) \cdot y_{i,f,m} \quad - \\
& \sum_{f \in F} c^p \cdot U_{if} \quad - \\
& \sum_{f \in F} \sum_{m \in M} c^j_m \cdot P \cdot V_{ifm} \quad + \\
& \sum_{f' \in F'} \sum_{m' \in M} (c^p + c^j_{m'}) \cdot {y'}_{i,m',f'} \quad - \\
& \sum_{f' \in F'} c^p \cdot {U'}_{if'} \quad - \\
& \sum_{f' \in F'} \sum_{m' \in M} c^j_{m'} \cdot P \cdot {V'}_{if'm'} \quad + \\
& \sum_{f' \in F'} \sum_{m' \in M} c^j_{m'} \cdot W_{if'm'} \cdot t_{f',m'} \cdot (1 - P) \quad \leq \quad n_i \\
& where \\
& U_{if} = \begin{cases} \sum_{m \in M} y_{i,f,m} - 1 & if \sum_{m \in M} y_{i,f,m} > 1 \\ 0 & otherwise \end{cases} \\
& V_{ifm} = \begin{cases} 1 & if((\sum_{f \in F} y_{i,f,m} > 1 \ \| \\ & \sum_{f' \in F'} y'_{i,f',m} \geq 1 \ \| \\ & \sum_{f' \in F'} (k_{i,f',m} \cdot \sum_{i \in N} {y'}_{i,f',m}) \geq 1) \\ & \& \ (y_{i,f,m} = 1)) \\ 0 & otherwise \end{cases} \\
& {U'}_{if'} = \begin{cases} \sum_{m' \in M} {y'}_{i,f',m'} - 1 & if \sum_{m' \in M} {y'}_{i,f',m'} \\ & > 1 \\ 0 & otherwise \end{cases} \\
& {V'}_{if'm'} = \begin{cases} 1 & if((\sum_{f \in F} y_{i,f,m'} \geq 1 \ \| \\ & \sum_{f' \in F'} y'_{i,f',m'} \geq 1 \ \| \\ & \sum_{f' \in F'} (k_{i,f',m'} \cdot \sum_{i \in N} {y'}_{i,f',m'}) \geq 1) \\ & \& \ (y'_{i,f',m'} = 1)) \\ 0 & otherwise \end{cases} \\
& W_{if'm'} = \begin{cases} 0 & if \sum_{f \in F} y_{i,f,m'} \geq 1 \ \| \\ & \sum_{f' \in F'} y'_{i,f',m'} \geq 1 \ \| \\ & \sum_{f' \in F'} (k_{i,f',m'} \cdot \sum_{i \in N} {y'}_{i,f',m'}) \geq 1 \\ 1 & otherwise \end{cases}
\end{aligned} \tag{7}
$$

The constraint (7) checks that the cost of creating all the new functions or migrated previously deployed functions in a node does not exceed the current node resources. The first part of the constraint considers the total cost of a new function m, which are the cost of message pre-processing c^p and the cost of security function-specific c^j_m. In the second part, if several new functions related to a new flow in that node want to be created, only one pre-processing fee is paid. In the third section, the cost of collocation with similar functions (new, old migrated to the current node, or old node in the node) is reduced by a percentage of the p value. The fourth section checks that if an old function has migrated to this node, its cost should be considered. If two or more of the old functions of a current flow have migrated to that node, only one pre-processing cost must be considered.

The fifth section also checks whether the collocation cost reduction has conducted between the old migrated function and the similar functions in that node, whether new or old. The last section checks whether a previously deployed function in the current node (which was in the same node) has lost the saving cost of job collocation operation from the previous placement. It occurs when all similar functions of that function have migrated from that node, or no similar function has migrated to that node, or no new function is created in that node.

As mentioned earlier, the old flow path does not change only that the corresponding functions can migrate in the flow path, between nodes, in order to reduce the current placement cost. The next constraint checks that the old function does not deviate from the corresponding flow path when migrating.

$$
\begin{aligned}
\forall i \in N \setminus \{s'_f, d'_f\}, f' \in F', m' \in M : \\
p_{i,f'} - y'_{i,f',m'} \geq 0
\end{aligned} \tag{8}
$$

TABLE 3: Security optimization problem parameters initialization.

Network	$N = \{1, \cdots, n\}$	fat-tree k={4,8,•...,128}
	$l_{i,j}$	1Mbps...100Mbps
	n^{cpu}_i	1...100
	n^{mem}_i	1G...100G
	H^{min}_f	6
	H^{max}_f	6
Flows	$F = \{1, \cdots, h\}$	3-40 flows
	b_f	1Mbps...5Mbps
	$F' = \{1, \cdots, h'\}$	0-36 flows
	$b_{f'}$	1Mbps...5Mbps
Middleboxes	$M = \{1, \cdots, m\}$	1-3 NF(s)
	$c^{j^{cpu}}_m$	1..3
	$c^{j^{mem}}_m$	1G..3G
	$c^{p^{cpu}}$	1
	$c^{p^{mem}}$	1G
	P	50%
	sa_f	1%...100%

5 Experimental Results

5.1 Simulation Setup

Table 3 shows the list of variables used in the simulation and the range of its values. The topology chosen to simulate the stateful placement is the fat-tree network topology. To show the scalability of the idea, different sizes from 36 nodes for k=4 to 540,800 nodes for k=128 are considered. The link capacity is generated randomly between $1Mbps$ to $5Mbps$. Regarding the capacity of nodes (CPU and memory) sizes are generated randomly between 1 to 100. H^{min}_f and H^{max}_f, the minimum and maximum number of links for the path chosen for flow, are set to six (Note that our traffic is east-west in the fat-tree network topology, from server to server). This is due to the structure of the fat-tree topology, so the minimum and maximum values can be changed according to the network structure.

The number of flows in our simulation is from 3 to 40. Each flow can request a different amount of bandwidth, b_f. So a random number is generated between $1Mbps$ and $5Mbps$. The amount of current flows (F') in the network, when we are performing

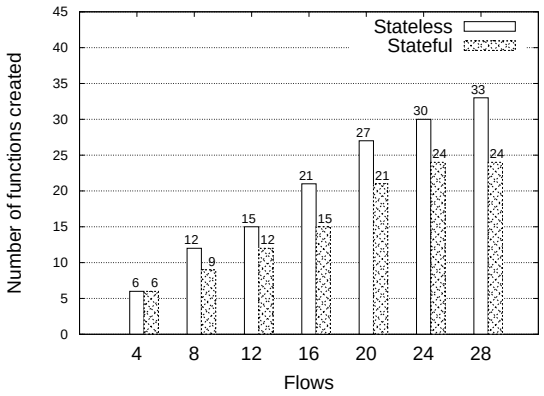


Fig. 3: Comparison of the number of network functions created in stateful and stateless placement approaches

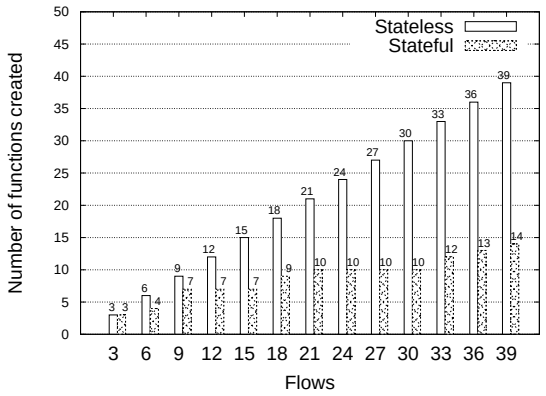


Fig. 4: Comparison of the number of network functions created in stateful and stateless placement approaches. Functions are not similar between flows

the optimal placement for new flows, varies from 0 to 36. In the first placement, the number of flows in the network is zero. Each security function has different requested resources, c_m^j, in terms of vCPU and vMem based on the task it performs. Therefore, the amount of processor, $c_m^{j^{cpu}}$, requested varies between one and three, and the amount of memory, $c_m^{j^{mem}}$, requested varies between $1G$ and $3G$. We have considered the amount of savings in the job collocation to be 50% (parameter P). To perform traffic pre-processing, we have considered the equivalent cost of one CPU ($c^{p^{cpu}}$) and $1G$ memory ($c^{p^{mem}}$). This cost is considered in each node for each flow if there is at least one function of a flow. Finally, the sampling rate for each flow, sa_f, is generated randomly between 1 and 100 percent. We have executed our program on a machine with 4 cores Intel(R) Core(TM) i5-2450M clocked at 2.5 GHz with 4 GB RAM. Moreover, we used the GLPK (GNU Linear Programming Kit) as the mathematical optimization problem solver.

In the results section, two types of simulations have been performed. In the first experiment, four new flows are added to the network in each placement and each of which requests one to three function(s). Moreover, there are similar functions between flows. In the second experiment, three new flows are added to the network in each placement and each of which requests only one network function such that functions are not similar between flows. In both experiments, the number of flows is done up to forty. In places where the simulation was not performed up to forty flows, it was due to the scalability issue. The parameters initialization, presented in Table 3, are the same for the two experiments.

5.2 Results

Figure 3 shows the number of network functions created in stateful and stateless placements approaches, in fat-tree K=4. In this experiment, at each substitution, we have four requested flows, from different origins to different destinations. Each flow can request one to three network function(s) ($NF_1 : 2vCPU, 2GvMem$, $NF_2 : 3vCPU, 2GvMem$, and $NF_3 : 1vCPU, 2GvMem$). As can be seen in the figure, and is expected, since the stateless approach does not take into account the current functions created in the network, it will provide more network functions than the stateful approach. Also, the rate of creation of functions in the stateful approach for new requests decreases sharply as time passes and functions increase. It should be noted that for the four flows,

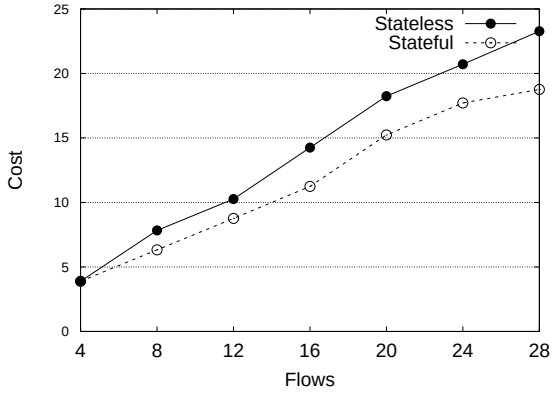


Fig. 5: Comparison of the cost of stateful and stateless approaches

each of which may request three functions, in the maximum case, without any job collocation, there will be twelve NFs, which did not occur as shown in the figure. At best, when we collocate the same type of functions, we will have three functions that have occurred over time, depending on the network's current cost conditions. As shown in the figure, in the seventh placement, the stateless approach has created nine more functions, which over time has become much more costly for the network and cloud customers.

To demonstrate the value of the stateful approach in network resource conservation, we performed another experiment where it was not possible to perform job collocation into the requested new flows. In this experiment, we have performed the placement up to thirteen times. In each placement, flows f_1, f_2, and f_3 call for the network functions NF_1, $NF2$, and NF_3, respectively. Therefore, network functions can only be merged between new requests and existing ones on the network. Figure 4 shows the results of this simulation. As can be seen, the stateless approach creates three functions at each placement, and since there are no similar functions between requests, it cannot reduce the number of functions in any way. In contrast, the stateful approach does not create new functions in many placements when the number of current network functions increases.

Figure 5 compares the cost of stateful and stateless approaches, for the case where each flow could request one to three network function(s) such that it was possible to merge functions between

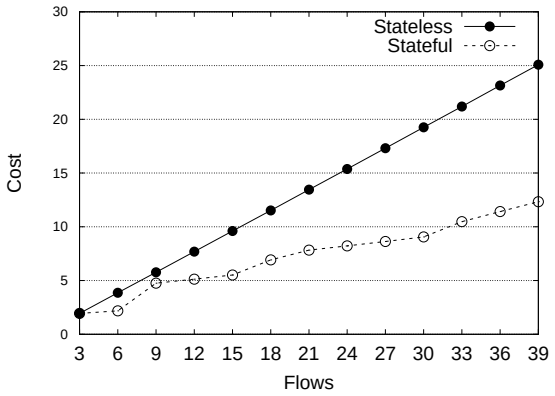


Fig. 6: Comparison of the cost of stateful and stateless approaches. Functions are not similar between flows

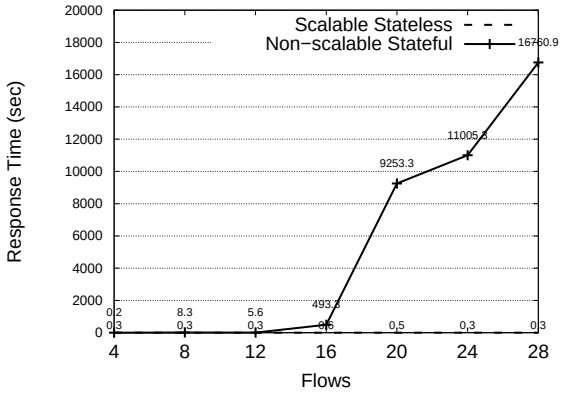


Fig. 7: Comparison of response time of scalable stateless and non-scalable stateful approaches

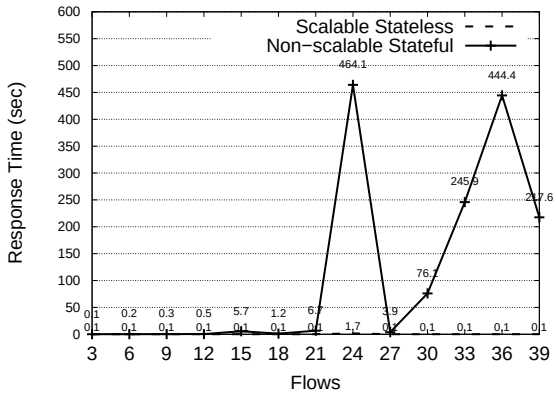


Fig. 8: Comparison of response time of scalable stateless and non-scalable stateful approaches. Functions are not similar between flows

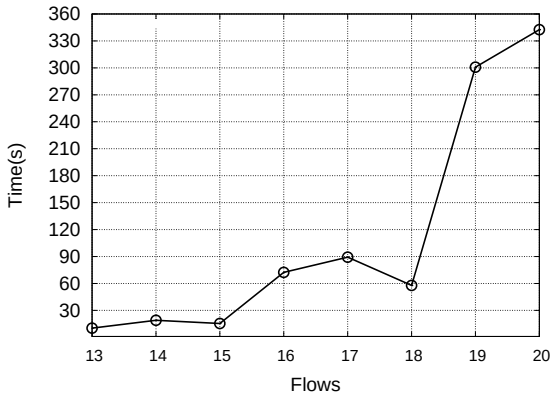


Fig. 9: Improving the response time of the stateful approach in the first experiment by sequential execution of requests (see Figure 7)

new requests. As can be seen, the stateful approach has a significant reduction in cost due to the job collocation operation between new requested network functions and the previously created ones. As shown in the figure, the cost of the stateless approach is increasing with a very high slope. It is important to note that in the seventh placement, our approach has saved by about 20%.

Figure 6 shows the cost comparisons as there are three new flows in each placement, each only requesting a different function from the other flows. As seen, the cost of each placement in the stateless approach is the same because it is not able to perform job collocation between new flows as well as with current network functions. Therefore, the total cost of placements increases linearly. In contrast, the stateful approach has a much lower cost reduction compared to the stateless approach. As seen in Figure 6, the cost slope in the stateful approach decreased by 57% compared to the stateless approach.

Figure 7 compares our proposed non-scalable stateful approach to scalable stateless for the first experiment. To overcome the scalability issue, we applied zoning heuristic on stateless approach, so that the topology is divided into three zones. The first zone, which is close to the source of the flow, includes the edge and aggregation switches. The second zone includes the core switches. The third zone, which is close to the flow destination, includes the aggregation and edge switches. As shown in the figure, the response time of our stateful approach in the first, second, third, and fourth placements is 0.2 seconds, 5.6 seconds, 8.3 seconds, and 8.2 minutes, respectively. When the number of flows reaches 20, the response time will be about two and a half hours. This would be, respectively, three hours and four hours and forty minutes for the number of 24 and 28 flows. It is obvious that applying the heuristic idea leads to better response time, but on the other hand, it increases the cost of placement. The impact of the zoning heuristic is to make the parser collocation ineffective, because the functions belonging to the flow have to be created in different regions. Creating any function in each node, if there is no function of a similar flow in that node, will have a pre-processing cost.

Figure 8 shows the result of the second experiment. The important point to note is the irregularity of the response time of the stateful approach. The reason is that each flow requests a network function, and it is more likely that the answer will be faster sometimes.

Before applying the zoning heuristic, we examined the idea of sequential execution for concurrent requests. As the result of the first experiment shows in Figure 7, our stateful solution was no longer acceptable when 16 flows with 15 network functions (see Figure 3) were in the network and the response time was about 8.2 minutes. The results of the second experiment (see Figure 8), where the number of functions in the three new concurrent flows was also low, implying that when the current network functions are increased, the stateful approach is not scalable. Instead of

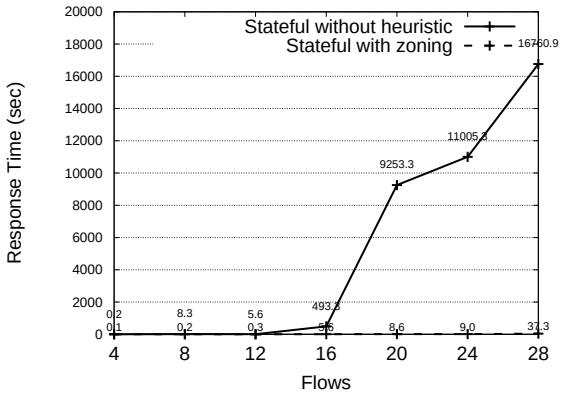


Fig. 10: The effect of zoning heuristic on reducing the stateful approach response time

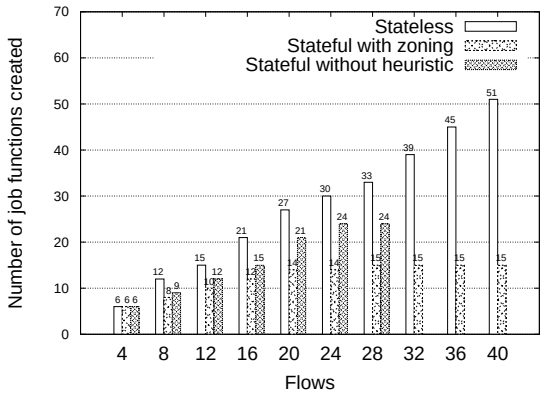


Fig. 11: Comparison of the number of network functions created in scalable and non-scalable stateful and scalable stateless approaches in the first experiment

applying four requests simultaneously, we decided to execute them sequentially. We repeated the same experiments for flows 13 to 20 as shown in Figure 9. As can be seen in the graph, the total response times for the four flows from 13 to 16, in sequential execution, decreased from 8.2 minutes to less than two minutes. This time has been reduced to about 13.7 minutes from two and a half hours for the next four flows.

In the stateless approach, since no job collocation operation is performed between the new and the old flows, the only saving is between the flows that are received together, because the merge operation can occur between similar network functions. Therefore, if we execute concurrent requests sequentially, there will never be a cost reduction. This will not happen in an stateful approach, because when we are finding the optimal locations for the functions of a flow, all current network functions are always considered to reduce costs. So although sequential or simultaneous execution of flows in the stateful approach does not change the cost of placement and it seems that sequential execution is better because of response time, as shown in the graph when the number of deployed network functions increase the response time for a new request increases dramatically. For example, in the case of flow 20, it takes about five minutes to find the right optimal location, which may not be suitable for security functions.

Figure 10 shows the result of applying the heuristic idea (zoning) to the stateful placement approach. As can be seen, the results have improved dramatically. For example, in the case of 24 flows that took about four hours and forty minutes, it now takes about 37.3 seconds. We were also able to run experiments for more flows, using the zoning heuristics. If we want to summarize the results on fat-tree, K=4, we can conclude that if one to twenty-four requests (network functions 3 to 72) are received at the same time, our scalable approach can operate at an acceptable response time.

Figure 11 shows the number of functions created in scalable and non-scalable stateful, and scalable stateless approaches. As can be seen, zoning has made the creation of functions much less. The reason is that we force similar functions to be created in one area. It is important to note that the best cost reduction in creating functions occurs when the order of functions is the same in all flows. In many applications, we need a special order between network functions. For example, we want to eliminate certain traffic before the traffic reaches our application. So we need traffic to pass through the firewall first. Then the traffic is

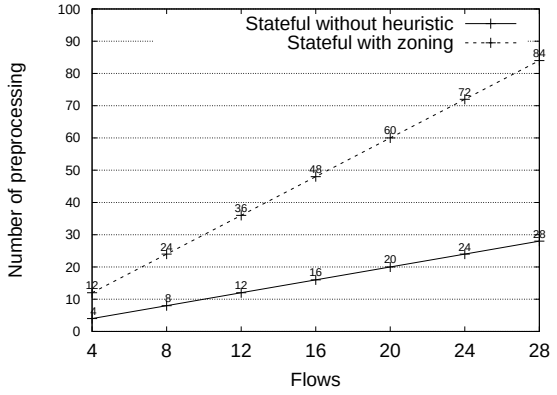


Fig. 12: Comparison of the number of pre-processing in scalable and non-scalable stateful placement approach in the first experiment

passed to the intrusion detection system to analyze the attack. As shown in the graph, when the number of flows reaches 40, our scalable stateful approach has created about three and a half times less network functions than the scalable stateless approach in the network. Note that for an accurate comparison, both stateless and stateful approaches use the same heuristics, zoning.

Although zoning has the benefits of reducing response time and creating functions, it also has a penalty, which is to make further pre-processing. Because zoning causes functions for a flow to be created at different nodes, in which example, each flow that could have at least one pre-processing would have to pay three parser costs. Figure 12 compares the costs associated with the parser in the stateful approach without heuristic and using zoning. Next, we want to compare the cost of stateful approach with zoning heuristic with other approaches. As Figure 13 shows, the stateful approach with zoning heuristic is less costly than the normal stateful approach. Although zoning increases the cost of pre-processing, it produces much less network functions than other approaches, as Figure 11 shows. As can be seen, the cost slope of the stateful approach has been reduced by zoning heuristic about 46% compared to the stateless approach.

We now want to examine the scalability of the stateful approach presented for large networks. For this purpose, we examine ten flows, each requesting three network functions, in networks of

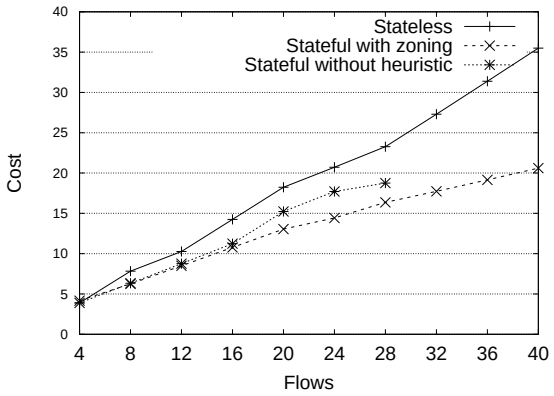


Fig. 13: Comparison of the cost of stateful and stateless approaches

Algorithm 1: Topology partitioning and zoning in fat-tree

```
Require: T[]: fat-tree topology
Require: s-node: flow source node
Require: d-node: flow destination node
Ensure: sub-T[]: sub fat-tree topology
  l-edge= T[].findlink(s-node) {Zone 1}
  n-edge= T[].findnode(l-edge)
  sub-T[].addlink(l-edge)
  sub-T[].addnode(n-edge)
  l-agg[]= T[].findlink(n-edge)
  n-agg[]= T[].findnode(l-agg[])
  sub-T[].addlink(l-agg[])
  sub-T[].addnode(n-agg[])

  l-core[]= T[].findlink(n-agg[]) {Zone 2, part 1}
  n-core[]= T[].findnode(l-core[])
  sub-T[].addlink(l-core[])
  sub-T[].addnode(n-core[])

  l-edge= T[].findlink(d-node) {Zone 3}
  n-edge= T[].findnode(l-edge)
  sub-T[].addlink(l-edge)
  sub-T[].addnode(n-edge)
  l-agg[]= T[].findlink(n-edge)
  n-agg[]= T[].findnode(l-agg[])
  sub-T[].addlink(l-agg[])
  sub-T[].addnode(n-agg[])

  l-core[]= T[].findlink(n-agg[]) {Zone 2, part 2}
  n-core[]= T[].findnode(l-core[])
  sub-T[].addlink(l-core[])
  sub-T[].addnode(n-core[])
```

different sizes. Assume that all ten flows have the same origins and destinations. The common origins and destinations make it possible for us to use an important attribute in the fat-tree topology, which is to extract the problem graph because we do not need to give the whole graph to the solver, and this heuristic is called, *topology partitioning*. Algorithm 1 shows how the related graph can be extracted from the fat-tree topology along with three zones which is based on 3-level of fat-tree topology property. In Table 4, we can see the results of applying this scenario in different topologies. As can be seen, in a network of 540,800 nodes and 1,572,864 edges, the response time with both heuristic, *topology partitioning* and *zoning*, is very acceptable and takes about 3.4 seconds.

TABLE 4: Applying topology partitioning and zoning heuristics in stateful approach in the fat-trees with different sizes

			Time(s)	
K	Node	Link	Topology Partitioning (TP)	TP+Zoning
4	36	48	$0.1s$	$0s$
8	200	384	$0.2s$	$0s$
16	1,296	3,072	$0.5s$	$0.1s$
24	4,056	10,368	$0.7s$	$0.1s$
32	9,248	24,576	$1.2s$	$0.2s$
48	30,000	82,944	$2.6s$	$0.4s$
64	69,696	196,608	$5.6s$	$0.7s$
128	540,800	1,572,864	$17.9s$	$3.4s$

6 Conclusion

This paper introduced a scalable stateful approach for optimal placement of virtual network functions in cloud data centers. The main idea is to reduce the number of network functions in the network to conserve resources and to balance customer costs. Therefore, when receiving new requests, previous network functions created in the network are also considered in the new placement. It is clear that not all previous placements can be optimized when addressing new requests, since processing the new requests itself is an NP-hard problem. To make the results acceptable, we applied four heuristics. The first heuristic is to not re-routing the previous flows. The second idea is to move the previously created network functions only in the direction of the corresponding flows, which is actually in line with the first heuristic. The third heuristic is to create network zoning to limit the locations where new functions can be created based on the structure of the fat-tree topology and the type of security functions. The last heuristic is to extracting the graph that relates to new flows for placement, not the entire network. The results in the paper have shown that our stateful approach has been able to improve the cost of network functions placement significantly compared to previous solutions. In the future, we want to analyze our model for maximizing the Quality-of-Experience of the virtual streaming service in the virtual content delivery network (CDN).

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

[3] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin, "Improve service chaining performance with optimized middlebox placement," *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 560–573, 2017.

[4] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[5] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[6] Q. Duan, N. Ansari, M. Toy *et al.*, "Software-defined network virtualization: an architectural framework for integrating sdn and nfv for service provisioning in future networks." *IEEE Network*, vol. 30, no. 5, pp. 10–16, 2016.

[7] U. Fiore, P. Zanetti, F. Palmieri, and F. Perla, "Traffic matrix estimation with software-defined nfv: Challenges and opportunities," *Journal of computational science*, vol. 22, pp. 162–170, 2017.

[8] P. Vizarreta, M. Condoluci, C. M. Machuca, T. Mahmoodi, and W. Kellerer, "Qos-driven function placement reducing expenditures in nfv deployments," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–7.

[9] M. Obadia, J.-L. Rougier, L. Iannone, V. Conan, and M. Brouet, "Revisiting nfv orchestration with routing games," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2016, pp. 107–113.

[10] X. Song, X. Zhang, S. Yu, S. Jiao, and Z. Xu, "Resource-efficient virtual network function placement in operator networks," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–7.

[11] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, "Deep packet inspection as a service," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 271–282.

[12] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simplefying middlebox policy enforcement using sdn," in *ACM SIGCOMM computer communication review*, vol. 43, no. 4. ACM, 2013, pp. 27–38.

[13] 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.

[14] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.

[15] S. Ahvar, H. P. Phyu, S. M. Buddhacharya, E. Ahvar, N. Crespi, and R. Glitho, "Ccvp: Cost-efficient centrality-based vnf placement and chaining algorithm for network service provisioning," in *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2017, pp. 1–9.

[16] 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.

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

[18] 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," in *Local and Metropolitan Area Networks (LANMAN), 2015 IEEE International Workshop on*. IEEE, 2015, pp. 1–6.

[19] A. Hmaity, M. Savi, L. Askari, F. Musumeci, M. Tornatore, and A. Pattavina, "Latency-and capacity-aware placement of chained virtual network functions in fmc metro networks," *Optical Switching and Networking*, vol. 35, p. 100536, 2020.

[20] M. M. S. Maswood, C. Develder, E. Madeira, and D. Medhi, "Energy-efficient dynamic virtual network traffic engineering for north-south traffic in multi-location data center networks," *Computer Networks*, vol. 125, pp. 90–102, 2017.

[21] M. Jabbarifar, A. Shameli-Sendi, and B. Kemme, "A scalable network-aware framework for cloud monitoring orchestration," *Journal of Network and Computer Applications*, vol. 133, pp. 1–14, 2019.

[22] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach," *IEEE Transactions on Services Computing*, 2017.

[23] Y. T. Woldeyohannes, A. Mohammadkhan, K. Ramakrishnan, and Y. Jiang, "Cluspr: Balancing multiple objectives at scale for nfv resource allocation," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1307–1321, 2018.

[24] 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," in *Local and Metropolitan Area Networks (LANMAN), 2015 IEEE International Workshop on*. IEEE, 2015, pp. 1–6.

[25] A. Shameli-Sendi, Y. Jarraya, M. Pourzandi, and M. Cheriet, "Efficient provisioning of security service function chaining using network security defense patterns," *IEEE Transactions on Services Computing*, vol. 12, no. 4, pp. 534–549, 2019.

[26] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008–2025, 2017.

[27] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 643–655, 2019.

[28] D. M. Manias, M. Jammal, H. Hawilo, A. Shami, P. Heidari, A. Larabi, and R. Brunner, "Machine learning for performance-aware virtual network function placement," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

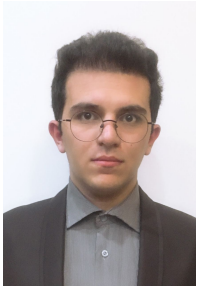
[29] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, and P. Demeester, "Network service chaining with optimized network function embedding supporting service decompositions," *Computer Networks*, vol. 93, pp. 492–505, 2015.

[30] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[31] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network function placement for nfv chaining in packet/optical datacenters," *Journal of Lightwave Technology*, vol. 33, no. 8, pp. 1565–1570, 2015.

Niloofar Moradi is currently a M.Sc. student at Computer Science and Engineering Faculty, Shahid Beheshti University, Tehran, Iran. She received her B.Sc. degree from Amirkabir University of technology, Tehran, Iran. She received Bronze Medal in Iranian Mathematic Olympiad. Her current research interests include cloud computing and mathematical optimization.

Alireza Shameli-Sendi is currently an Assistant Professor at Shahid Beheshti University. Before joining SBU, he was a 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 Montreal University (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. In addition, he received the best researcher award, in industrial track, at SBU, in 2018.



Alireza khajouei received his B.Sc. degree in Software Engineering from Sharif University of Technology, Tehran, Iran, in 2018. He is currently a M.Sc. student at the Computer Science and Engineering Faculty, Shahid Beheshti University, Tehran, Iran. He received Silver Medal in Iranian Mathematical Olympiad. His current research interests include virtual machine placement, virtual network function placement and mathematical optimization.