2015 IEEE 8th International Conference on Cloud Computing

Multistage OCDO: Scalable Security Provisioning Optimization in SDN-based Cloud

Yosr Jarraya*, Alireza Shameli-Sendi†, Makan Pourzandi* and Mohamed Cheriet‡
* Ericsson Security Research, Ericsson Canada, Montreal, Qc, Canada
† Department of Computer Science, McGill University, Montreal, Qc, Canada
‡ Synchromedia Lab, École de Technologie Supérieure (ETS), University of Quebec, Montreal, Qc, Canada
Emails: {yosr.jarraya, makan.pourzandi}@ericsson.com, alireza.shameli-sendi@cs.mcgill.ca, mohamed.cheriet@etsmtl.ca

***Abstract*—Cloud computing is increasingly changing the landscape of computing, however, one of the main issues that is refraining potential customers from adopting the cloud is the security. Network functions virtualization together with software-defined networking can be used to efficiently coordinate different network security functionality in the network. To squeeze the best out of network capabilities, there is need for algorithms for optimal placement of the security functionality in the cloud infrastructure. However, due to the large number of flows to be considered and complexity of interactions in these networks, the classical placement algorithms are not scalable. To address this issue, we elaborate an optimization framework, namely OCDO, that provides adequate and scalable network security provisioning and deployment in the cloud. Our approach is based on an innovative multistage approach that combines together decomposition and segmentation techniques to the problem of security functions placement while coping with the complexity and the scalability of such an optimization problem. We present the results of multiple scenarios to assess the efficiency and the adequacy of our framework. We also describe our prototype implementation of the framework integrated into an open source cloud framework, i.e. Openstack.**

***Keywords*—Cloud; Security Provisioning; Optimization; Segmentation; Decomposition; SDN; OpenStack;**

I. Introduction

Security has been mentioned by many companies as the major obstacle to a wide adoption of cloud in the enterprise IT market. With advancements in virtualization, a growing interest has been seen in replacing dedicated hardware-based network functions (NFs), e.g., intrusion detection systems (IDSs), load balancers, caching proxies, etc. with software-based NFs running on generic compute resources (e.g. commodity hardware servers and switches). This new trend, known as network functions virtualization (NFV) [1], intertwined with software-defined networking (SDN) opens up great opportunities to deliver new and innovative services. Software-based NFs can be deployed at various locations in the data center, which creates new possibilities to optimize network and computing resources. Furthermore, SDN allows to program traffic flows from a logically centralized controller. This allows for a flexible distribution of traffic among network nodes and thus fosters optimizing traffic steering at the switches and, more importantly, at flows granularity. For medium to large data centers, these two problems are known to be hard to solve [2], [3], [4] and they have been generally addressed separately to cope with the complexity of the compound problem.

Deployment of NFs are typically structured as logical chains, a.k.a service chains defined as "the required functions and associated order that must be applied to packets and/or frames"[5]. Different flows may require heterogeneous sequences of NFs in different orders as specified by flows policies [6]. Enforcing the order of traversal, a.k.a. policy-awareness [2], significantly increases the complexity of the problem and has been generally considered hard to solve within these optimization problems [3], [4]. Additionally, the stateful nature of several NFs imposes more constraints to the problem as bidirectional traffic of the same flow should traverse the same stateful NF, which represents a critical requirement for them to operate correctly [2].

In this paper, we propose an innovative approach to address the problem of security functions provisioning in the data center while being adequate and scalable. We mean by adequate that the right security modules (SMs) are deployed based on the tenant's virtual application security needs, while respecting the order of traversal and the correct processing of stateful NFs. Our main contribution is to elaborate OCDO, Ordered Cloud Defense Optimization algorithm, and a set of techniques that synergistically combined together achieve a correct provisioning of security while coping with the complexity and the scalability of such an optimization problem. The first technique, namely decomposition, leverages network topology characteristics and fosters concurrent optimization. The second technique, namely segmentation, relies on distributing the availabilities of security functions among a set of network nodes based on their semantics. The benefits of our approach are manifold. Firstly, it takes into account simultaneously the optimization of both computing and networking resources. This allows achieving higher levels of efficiency. Secondly, it adequately addresses the precedence requirements between security functions and the steering of bidirectional traffic, particularly for the correct operation of stateful security modules. Finally, our approach can scale to large data center topologies. These benefits and how we achieve them will be detailed in the paper.

The remaining part of this paper is organized as follows. Section II presents the related works. Section III describes in the problem that we intend to solve. Section IV presents the mathematical formulation of OCDO, our optimization framework. Section V describes our approach and its underlying components, namely decomposition and segmentation. Section VI briefly describes OCDO prototype and its integration into OpenStack. Section VII is dedicated to present and discuss

2159-6190/15 $31.00 © 2015 IEEE
DOI 10.1109/CLOUD.2015.82

the numerical results obtained by simulating our approach on different use cases. Section VIII concludes the paper.

II. Related Works

Related works can be divided into three threads. The first thread (e.g. [3], [7], [8], [9], [10], [11]) is assuming a fixed number of middleboxes and the proposed approaches optimally distribute the traffic flows among all possible routes through them. For instance, compared to CoMb [3], we are aligned with their distributed view of network security, however, we do not force all middleboxes for the same session to run on the same node, as this might be not suitable for certain types middleboxes (eg. firewalls and load balancer need to be deployed as clusters of active-standby pairs). We do require a session to pass by the same nodes hosting stateful security functions. A second thread (e.g. [12]) is assuming a set of pre-defined routes within the network and propose optimal placement of virtual middleboxes in the nodes through the routes. However, this class of works do not benefit from the dynamic flow forwarding offered by SDN. In a third thread, to which our work subscribes, research initiatives (e.g. [2], [4]) propose a hybrid approach where both computing and networking resources optimization are considered. However, in contrast with our work, the majority of studies propose to address these problems (placement of middleboxes and flows routing) separately, while we propose to address them simultaneously. This has the benefit of discarding solutions that would point out available computing resources but scarce network resources. Furthermore, several works [3], [4] have recognized that taking into account the order of traversing the security modules makes the optimal placement problem NP-hard. In this paper, we target to address the problem without relaxing the order constraint while making it scalable.

III. Problem statement

We assume a multi-tenant cloud data center with SDN. Using NFV, security functions can be implemented as software modules (SMs) deployed in the cloud infrastructure. These SMs can be further instantiated in computing nodes (e.g. physical servers) and in different networking nodes (e.g. inside or associated to switch, routers), or any other node provided that it can offer the required amount of resources (computing, memory, storage, etc). Security modules include but not limited to firewall layer 3 (FW-L3), firewall layer 4-7 (FW-L4-7), intrusion detection system (IDS), intrusion prevention system (IPS), Web application firewalls (WAF), virtual private network end nodes (VPN), etc. Each security module has a specific type (e.g. FW-L3), implementing a specific security policy (e.g. filtering from a specific source) and a required amount of resources (e.g. CPU, memory, storage). The different sizes of SMs may be modeled as VM instance sizes (e.g. Amazon instances models[1] or OpenStack flavors[2]) implementing some security functions, where each instance is configured with a certain number of vCPUs, amount of storage, etc.

We also assume that cloud tenants are enabled with appropriate tools to specify for their virtual cloud applications (vApps), the appropriate security modules with their types, sizes, and policies along with their precedence dependencies, which depends of the application needs, and the required bandwidth units (similarly to [4]). The security modules types and their precedence relations can be provided to the tenant through a set of templates that we call network defense patterns. These patterns can be derived from best practices in the network security field and organized in a database accessible by the tenants. They mainly specify the adequate order between different types of SMs and their right locations relatively to the source and destination nodes. An example of order, a FW-L3 is usually placed ahead of a FW-L4-7 to filter disallowed ports ahead of analyzing legitimate sessions. An example of an appropriate location for a firewall that used to shun malicious/unwanted traffic is at a location closer to the source of this traffic. For lack of space, we leave the description of network defense patterns for future work.

Concerning the topology of the network, conventional data centers are commonly designed using a tree-like topology according to a three-tier architecture [13]. 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. [14], [4], [13]). Tree-based topologies share similar connectivity but only differ in how addressing and routing are implemented. As such, 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.

Given these assumptions and settings, our main goal is to find an optimal placement and routing for the SMs requested by the tenants to secure their vApps. Our optimization objective is to consolidate/load balance (depending on the cloud admin objectives) networking and computing resources, while taking into account these requirements: (1) precedence relations enforcement between SMs. (2) steering traffic of any flow, in both directions, through the same stateful SMs, if any (3) scalability to handle medium to large data centers, with up to tens of thousands of nodes.

IV. OCDO Mathematical Formulation

In this section, we present the mathematical formulation of OCDO. We identified the traveling purchaser problem (TPP) [15] as it models to some extend our problem and fits into our objectives. It aims at determining a tour of one purchaser, from and to a depot, that needs to buy a set of items in several markets such that the total amount of travel and purchase costs is minimized. We are interested in the capacitated and symmetric version, where the quantities of products are limited depending on the market and the cost of traveling is the same for both directions of the links.

Mapping our problem into TPP, the products would be the SMs and the markets would be the network nodes where to place/instantiate these SMs. A purchaser would be in our case a traffic flow with the need to purchase a list of SMs in a

[1] Amazon instances types: https://aws.amazon.com/ec2/instance-types/
[2] Openstack: https://wiki.openstack.org/wiki/Horizon-NFV-configuration

TABLE I. PARAMETERS IN OUR OPTIMIZATION PROBLEM

$M = \{1, \cdots, m\}$	Set of network nodes
$P = \{1, \cdots, h\}$	Set of (unidirectional) flows between pairs of nodes
$N = \{1, \cdots, n\}$	Set of possible SMs
S_p (resp. D_p)	Node source (resp. destination) of the flow p, $S_p, D_p \in M$
L_p	Number of units of bandwidth needed by a flow p
$c_{i,j}$	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
$d_{l,p}$	Number of instances SM of type l required by the flow p.
$q_{i,l}$	Number of instances SM of type l that can be instantiated at node i.
$K_p \subset N$	Set of SMs that must be traversed by the flow p
$O_{l,p}$	Positive integer representing the order of the SM l with respect to the other SMs in the list K_p
CProd	List of SMs that need to be collocated
$CPur_l$	List of flows that need to collocate product l. In the default case (no collocation), it should contain one flow

specific order before reaching its destination. To handle the order of purchasing SMs while traversing the nodes through a valid path (not a circuit), we extend TPP in [16] to support multiple purchasers, with different source and destination nodes, and to handle multiple flows with asymmetric demands of SMs and different orders. In this paper, we also propose to add collocation constraints that provides the possibility to control placing several SMs of the same or different flows to be purchased at the same node. This may serve several security-related needs. For instance, for the stateful SMs to function properly, they need to see both directions of the same traffic flow. We use a purchaser per direction and we use the collocation constraint to enforce these requirements for all stateful SMs.

At a high-level, we need to decide, for each flow p generated by a source s_p and consumed by a destination d_p, which nodes to be visited and where security functions have to be placed. These can be captured using two binary variables $x_{i,j,p}$ and $y_{i,l,p}$. The first variable $x_{i,j,p}$ is defined for each flow p and each link between a pair of nodes i and j and it records whether the flow p has to travel through the link (i,j). The second variable $y_{i,l,p}$ defined for each combination of node i and SM l, and a flow p records whether the SM l has to be placed at node i and traversed by flow p. The solution to our optimization problem would be a set of $y_{i,l,p}$ and a set of $x_{i,j,p}$ that together denote the optimal placement of SMs and the optimal path to traverse these nodes. Table I summarizes parameters used in our formulation. To take into account the precedence relation between pairs of SMs, we use an integer variable $v_{i,j,p}$ that captures the order of visiting the links by each flow p in the solution. If the link (i,j) is not visited by flow p, then $v_{i,j,p} = 0$. If the link (i,j) is visited before the link (k,r) for a given flow p, we need to be able to verify that $v_{i,j,p} > v_{k,r,p}$. The first link visited after the source node has the largest visiting order value (i.e. $m-1$). Then, the visiting order of the successor links on the path is decreased by one between any two successive links until reaching the destination node.

As our objective is to consolidate cloud resources (c.f. §III), we should minimize the number of nodes and the number of links involved in the SMs placement and in flows steering in the data center network. Thus, we associate the lowest purchase costs to the SMs that are available on the already loaded nodes and the lowest traveling costs for links that are already having existing flows traveling through them. Costs for SMs and bandwidth can be calculated as ratios of used capacity over maximum capacity, measures that can be provided by the cloud management system. Thus, our objective is to minimize the total costs (c.f. Eq. 1) consisting of the cumulative cost of allocating units of bandwidth to all flows from their respective source node to their respective destination node (first summation of Eq. 1) and the cumulative cost of placing all SMs requested by all flows (second summation of Eq. 1). The mathematical formulation of our problem is provided in Figure 1. Note that, using this formulation, we can also tackle load balancing objectives. This can be done by associating purchase costs to SMs and traveling costs for bandwidth units that increase with the already existing load on the nodes and links.

Therein, constraint (2) also known as the flow conservation, denotes 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 only consumed by the destination node and no part is consumed by intermediary nodes. Constraint (3) denotes that the exact amount of SMs given in the demand list should be instantiated. Constraint (4) states that any SM can only be instantiated at a given market if it is actually available there. The term $card(CPur_l)$ counts the number of flows that need to collocate their SM l to merge the requests for collocated SMs. Constraint (5) indicates that a node should be included in the solution if the SM is to be instantiated at that node. Constraint (6) ensures that the allocated bandwidth does not exceed the maximum capacity of links. Constraints (7) avoid forming cycles in the solution path. Constraint (8) states that if a link (i,j) is not visited by a flow p (i.e. $x_{i,j,p} = 0$), the value of $v_{i,j,p}$ should be set to zero, or less than m, otherwise. Constraint (9) is used to calculate the values of $v_{i,j,p}$ along the path formed by the solution. Constraint (10) instructs that any SM l listed in K_p should be instantiated by flow p in the order it is requested as specified by $O_{l,p}$. Thus, for any two SMs l and k in K_p with a consecutive order such that $O_{l,p} = O_{k,p} - 1$, SM l should be purchased before SM k such that the node i where $y_{l,i,p} = 1$ is visited before the node j, where $y_{k,j,p} = 1$. If this is the case, the constraint verifies that $\sum_{r \in M} v_{r,i,p} \geq \sum_{r \in M} v_{r,j,p}$. Constraint (11) is the collocation constraint. It states that for two different flows $p1$ and $p2$, if $p1$ and $p2$ are in $CPur_l$, and l is in *CProd*, then both flows should instantiate the SMS l at the same node i. Constraint (12) is to enforce that x, y are binary variables and v is an integer variable.

The above formulation faithfully captures the specific requirements of our problem, however, solving this model involves a large discrete optimization problem which is difficult to solve in a reasonable time. As expected, the constraint added to handle the order requirement increases OCDO convergence time. Another factor that makes the problem difficult is a large search space driven by an increased (1) number of nodes m, (2) number of flows h and (3) availabilities of different SMs in different nodes $q_{i,l}$. In order to cope with such complexity, we propose to use a divide and conquer technique, which will be detailed in the next section.

Objective:

$$\textbf{\textit{Min}} \sum_{p\in P}\sum_{i\in M}\sum_{j\in M}(c_{i,j}\cdot L_p\cdot x_{i,j,p}) + \sum_{p\in P}\sum_{i\in M}\sum_{l\in N}(b_{i,l}\cdot y_{i,l,p}) \quad (1)$$

Subject to the following constraints:

$$\sum_{j\in M} x_{j,i,p} + (\text{if } i = S_p \text{ then } 1) = \sum_{r\in M} x_{i,r,p} + (\text{if } i = D_p \text{ then } 1) \quad \forall \text{i} \in \text{M}, \forall \text{p} \in \text{P} \quad (2)$$

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

$$\sum_{p\in P} y_{i,l,p} - q_{i,l} * card(CPur_l) \leq 0 \qquad \forall i \in M, \forall l \in N \quad (4)$$

$$\sum_{j\in M} x_{i,j,p} - y_{i,l,p} \geq 0 \quad \forall i \in M \setminus \{S_p\}, l \in N, p \in P \quad (5)$$

$$\sum_{p\in P} L_p * x_{i,j,p} \leq L_{i,j}^{max} \qquad \forall i,j \in M \quad (6)$$

$$\sum_{i\in M} x_{j,i,p} \leq 1, \sum_{j\in M} x_{i,j,p} \leq 1 \qquad \forall i \in M, \forall p \in P \quad (7)$$

$$v_{i,j,p} \leq m \times x_{i,j,p} \quad \forall i \in M, \forall j \in M, \forall p \in P \quad (8)$$

$$\sum_{j\in M} v_{j,i,p} + (\text{if } i = S_p \text{ then } m) = \sum_{j\in M}(v_{i,j,p} + x_{i,j,p}) \quad \forall i \in M \setminus \{D_p\}, \forall p \in P \quad (9)$$

$$m \times (\sum_{a\in M} x_{a,i,p} - y_{i,k,p}) - \sum_{r\in M} v_{r,i,p} + m \geq m \times y_{j,l,p} - \sum_{r\in M} v_{r,j,p}$$
$$\forall p \in P, i,j \in M \setminus \{D_p\}, i \neq j, l,k \in K_p, l \neq k, O_{l,p} = O_{k,p} - 1 \quad (10)$$

$$y_{i,l,p1} = y_{i,l,p2} \qquad \forall i \in M, l \in CProd, p1 \in CPur_l, p2 \in CPur_l, p1 \neq p2 \quad (11)$$

$$x_{i,j,p}, y_{i,l,p} \in \{0,1\}, v_{r,i,p} \geq 0, \quad \forall p \in P, i,j \in M, l \in K_p \quad (12)$$

Fig. 1. An Integer Linear Programming Formulation for OCDO

V. Multistage Scalable Optimization Approach

In §IV, we addressed the offline network planning problem, where aggregate demands in terms of flows and SMs, assumed to be known in advance, are provisioned within the network such that they fit into the capacities of links and nodes. However, in a SDN-based clouds, the demands are handled per request [4]. Thus, we propose to address a per request optimization problem, which is also known as online (reactive) forwarding in SDN. To generalize, each request in our case concerns the establishment of a session between a pair of end nodes[3] (e.g. VMs). Thus, for each request, we consider two sets of SMs, each with a predefined precedence order among its elements and each aims at securing one direction of the traffic between a pair of end nodes.

Figure 2 shows an example of security functions topology requested by a cloud tenant to secure the communication between two VMs. Therein, the tenant requests four security modules, specifically Fw-L3-1, IDS, WAF, and Fw-L3-2, to secure both directions of the communication between the VMs with an asymmetric order of SMs in each direction. The tenant also requests that both directions traverse the same IDS and WAF.

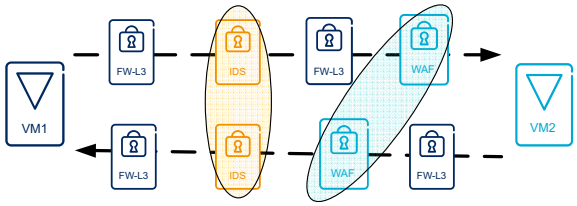

Fig. 2. Example of Bidirectional Security Functions Between VM1 and VM2 Specified by a Cloud Tenant

This use case can be addressed using OCDO with two purchasers (noted as OCDO-2), one purchaser per direction. To ensure that the traffic in both directions visit the same stateful node, we use the collocation constraint as explained in §IV. The online per-request strategy limits the number of simultaneous purchasers in OCDO to only two addressing one of the complexity factors described in §IV.

Taking into account the two other factors contributing to the complexity of the problem (i.e. number of nodes and availability of SMs) explained in §IV, we propose two techniques that composed together enable handling large data centers and multiple tenants requests to secure their virtual applications in the cloud while significantly improving the execution time (c.f. §VII). First, to address the problem of large number of nodes, we propose to decompose the topology of the network into a set of blocks, where each block is formed by independent paths (c.f. §V-A). Second, we propose to segment the availability of SMs among the nodes based on the position of these nodes with respect to the communicating endpoints (c.f. §V-B).

A. Network Topology Decomposition

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 [14]. For instance, if two hosts in two different PoDs have to communicate, 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, one ends up with a set of possible paths that are disjoint with respect to the other paths that are associated with any other aggregate switch. Based on this observation, we propose to decompose 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 3 illustrates the notion of block and zones in a k-ary Fat-tree topology. Having a

[3]This can be further extended by considering sources and destinations to be sub-networks or else.

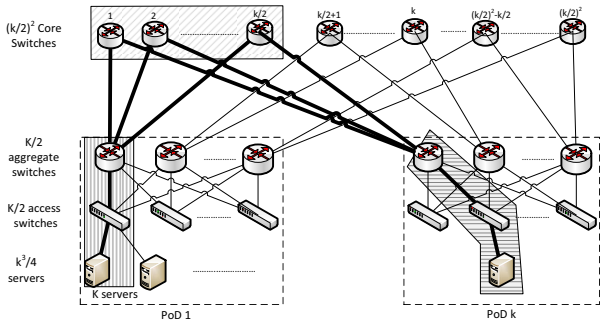

Fig. 3. Example of one Block with Three Availability Zones in 3-level k-ary Fat Tree. Switches and Links belonging to a single block are illustrated using bold lines. The three zones inside the block are illustrated using striped boxes.

decomposed network topology, the optimization problem can be then addressed concurrently in these independent blocks using our applying OCDO-2 algorithm. Since these blocks consist of disjoint 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 and without decomposition. Thus, we propose to compute the optimal costs in all blocks in parallel and then compare them together to find the optimal solution and the block where to place our flows.

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. Given a graph G capturing the fat-tree topology along with the source and destination nodes provided as input, Algorithm 1 describes the decomposition of G into blocks. Note that therein the variable sBlock is a sub-block that constitute the common part between Case 3 and Case 4. The functions *findLinks* and *findSwitches* have two parameters: The first 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. The function *getSwAndlinks* calls these functions and used as a shorthand to add nodes and links to block[i] from a given switch provided as parameter to this function. We distinguish four possible cases:

- Case 1: Both source and destination VMs are located in the same physical node. In this case, the physical server is a single node to instantiate any SM. This case can be easily addressed by instantiating the security modules at the server hosting the VMs.
- Case 2: The VMs are located in two different physical servers that are connected to the same ToR switch. This case maps to finding the spot with enough resources among physical servers, 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 $k/2$ aggregates switches. We directly apply OCDO-2 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 decompose 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 for this case $k/2$ blocks. Each block will consists of six nodes (fixed part) plus k/2 other nodes (variable part) corresponding to the number of cores 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.

From now on, we solve the use case 4 as being the most complex among the enumerated ones.

Algorithm 1 TopologyDecomposition Algorithm

```
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].getSwAndlinks(link, agg-s,up)
                  cores-s[] = G.findSwitches(agg-s,up)
                  Block[i].getSwAndlinks(link, cores-s[],down)
                  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[]
```

B. Segmentation of products availabilities

The decomposition reducing the number of nodes in each block decreases the algorithm complexity and improves the overall scalability of OCDO-2 algorithm. Though, for large number of k, the experiments descried in §VII show that the algorithm still does not scale enough. To address this problem, we propose to rethink the availabilities of security functions by making some SMs available only in some zones within a block. Indeed, based on our experience and and the literature for best practices on the deployment of network security appliances, we realized that there are well-known security functions deployment patterns, called here network defense patterns, that one should take into account in the placement problem, beyond the simple mathematical criteria. For example, an IDS has generally to see all traffic to make the right decision. Further, VPN terminations are useless if deployed in the core of the network and a WAF or an IDS are 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. Therefore, based on these observations, instead of making any security function available everywhere, we distribute the availability of each security function depending on 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 inefficient and incorrect deployment of SMs.

Thus, we propose to divide a given service chain into three independent segments that we call security 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 similarly each block into three zones. Then, we propose to assign each segment to a zone, where the SMs within the segment are made available in the zone's nodes. The final deployment is most often a mixture of optimizations based on some or all factors mentioned above. The main contribution in this paper being making OCDO-2 algorithm scalable for large problems, we then propose to tackle the segmentation algorithm in future work. For this paper, we consider that the segmentation is done by the tenant security expert. As the security experts deploy different security appliances in daily life based on the similar factors, we believe this assumption is realistic.

C. Multistage OCDO

The multistage OCDO algorithm uses the decomposition and segmentation conjointly to solve the optimization of SMs in the cloud infrastructure. Algorithm 2 describes our approach. We first decompose the network into different blocks and then segment the security chain between source and destination nodes. Afterward, we assign segments to different zones. These assignments are used to generate the availability of SMs in the nodes of each zone in each block, using the function *defineAvailabilityData*. The actual quantity of SMs of different types $q_{i,l}$ are defined based on the actual resources (i.e. CPU, memory, storage) in each node i in the block and the zone it belongs to. For instance, if a SM l' is available in a given zone Z, we have $q_{i,l'} \neq 0$ for every node $i \in Z$, otherwise, $q_{i,l'} = 0$. Once the availability data is generated for all blocks, several OCDO-2 programs (i.e. instances are equal to the number of blocks) are executed in parallel such that each thread is fed with the data concerning its associated block. OCDO-2 is then executed for all blocks, simultaneously. Finally, the block with the minimal cost represents the solution in which the actual placement is performed.

Algorithm 2 Multi-stage Algorithm

```
Require: chain: Ordered SMs with their types and sizes
Require: s-node
Require: d-node
Require: G: Topology graph
Require: data: OCDO-2 input without availabilities (q_{i,l})
  Blocks[] = TopologyDecomposition(G, s-node, d-node)
  Segments[Seg, AssignedZone] = Segment(chain)
  for all b in Blocks[] do
      SMAvailabilities[b]= defineAvailabilityData(Segments[][])
  end for
  Sol[] = ParallelComputeOCDO-2(data,SMAvailabilities[])
  Sol[block-opt] = MinimumCost{Sol[b]; b in Blocks[]}
  executePlacement(Sol[block-opt])
```

VI. OCDO Prototype

As a proof of concept, we implemented our algorithms and integrated them into Openstack[4] using OpendDayLight (ODL)[5] for network steering inside Openstack. We integrated our prototype into our Cloud Defense Orchestration (CDO) framework. The latter consists of CDO-Manager (CDO-M) that is designed to interact mainly with Openstack control services. CDO-M collects networking and computing information from Openstack, including the network topology, the loads on different nodes and links, and the availabilities of resources at the nodes for different type/sizes of security modules. Using these information, CDO-M builds the graph annotated with costs and availabilities. CDO-M then communicates this graph along with the bidirectional service chain annotated with the types, sizes, and precedence of the required SMs to our extension of Nova scheduler that uses our CDO-evaluator (CDO-e). The latter is a module that implements actually our optimization approach. It implements the topology decomposition algorithm, SM availabilities distribution algorithm, and multi-stage placement algorithm. CDO-e interacts with several instances of the mathematical solver to communicate different decomposed graphs and the bidirectional service chain and to receive the optimal placement solution. We use Nova to instantiate the right security module in the right node. To realize the service chaining and traffic steering in the network, we use OpenDaylight (ODL), the SDN controller, which enables rich and flexible networking functionalities for CDO.

VII. Numerical Results

In the following, we describe the simulation experiments and the related numerical results to demonstrate the scalability of our approach. We also study the distance with respect to optimality.

[4]http://www.openstack.org/software/icehouse/

[5]http://www.opendaylight.org/

A. Simulation Setup

We use a machine with 6 cores Intel Westmere clocked at 2.30 GHz with 24 GB RAM. We used as mathematical optimization problem solver the GLPK [6]. 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 drastically based on time, applications running in the cloud, etc., we use random values to represent the loads. This also has been applied in other related works (e.g. [2]). Thus, we randomly generate costs of bandwidth units and costs of SMs in a range of $[0, 1]$. Furthermore, the quantity of SMs available at each node is randomly generated in the interval [0,5] with uniform distribution according to the selected segmentation. We also randomly generate links capacities. The experiments were run for different fat tree sizes (from k=4 to k=128), with different number of SMs (but we show only the case of 4 SMs as presented in Fig. 2), with and without decomposition and for different segmentation choices.

We consider two sorts of segmentation: Seg_N denotes a segmentation that generates non-overlapping availabilities of SMs in adjacent zones and Seg_O a segmentation with overlapping in SMs availability in adjacent zones. Let Z_s, Z_d, and Z_c denote respectively zone at source PoD, zone at destination PoD and zone at the core, in given block. For the sake of this paper, we have chosen two segmentations as we believe they describe well how in real world SMs are deployed: Seg_N ={ Z_s:= FW-L3; Z_c := IDS; Z_d := FW-L3,WAF} and Seg_O ={ Z_s:= FW-L3,IDS; Z_c := IDS; Z_d:=FW-L3,WAF}. Note also that FW-L3 in Z_s and Z_d are considered as different FW instances. For the sake of space, we will not discuss this choice here though we believe that this use case can be extended to compare any two segmentations.

B. Results and Discussion

We propose to compare the ILP formulation of our problem (no decomposition and no segmentation) to our multistage approach. In the following, we also investigate the impact of each mechanism on the algorithm performance. We finally investigate the penalty on optimality induced by the segmentation.

1) Decomposition Benefits: In order to show how our decomposition approach, described in Section V-A, impacts the execution time and the cost, we ran our algorithm, with and without decomposition, for all k values. For the experiments without decomposition, we could not run the case of $k \geq 16$ due to the limitations of GLPK solver that cannot handle large number of nodes. Table II summarizes the execution time results. As one can note, the execution time for decomposed solution is 2.8 times better for $k = 4$ and 32.1 times better for $k = 8$. Even with decomposition, we could not run our experiment for $k = 128$. As for the total cost of the solution (we don't show the cost results here), it is identical for both cases (i.e. with or without decomposition) for $k = 4$ and $k = 8$. This could not be verified by experiments for $k \geq 16$. One can notice that the decomposition significantly improves the execution time without hurting the optimality and correctness of the solutions. As explained in the section V-A, decomposition reduces the search space for the algorithm, which reduces the execution time but does not impact the optimality of the algorithm results.

Even though the improvements in our algorithm execution time are significant with decomposition, we realized that even with medium k values (the execution time is already at 14.4 seconds for $k = 32$) the large execution times make our optimization algorithm usage difficult in challenging deployments where the decisions to place SMs in the cloud infrastructure need to be made in few seconds Therefore, segmentation of SMs availability, as presented next, is needed to improve on the execution time after decomposition.

TABLE II. OCDO-2 EXECUTION TIME IN SECONDS AS A FUNCTION OF K, FOR 2 FLOWS AND 4 SMS, WITH AND WITHOUT DECOMPOSITION.

	No Segmentation	
k	No Decomposition	Decomposition
k=4	1.4	0.5
k=8	44.9	1.4
k=16	-	1.6
k=32	-	14.4
k=64	-	61.2
k=128	-	-

2) Improvements using Segmentation : As illustrated in Table III, the segmentation not only results in better performances, from 5 times better for $k = 4$ up to 14 times better for $k = 64$, it further extends the applicability of our algorithm to larger data center (e.g. $k = 128$). For $k = 128$ and no-segmentation, OCDO-2 did not converge. We will present the impact of segmentation on the costs next.

TABLE III. OCDO-2 EXECUTION TIME IN SECONDS AS A FUNCTION OF K FOR 2 PURCHASERS AND 4 SMS, PER BLOCK, FOR THE CASES OF NO SEGMENTATION AND WITH SEGMENTATION Seg_N.

	Decomposition	
k	No Segmentation	Segmentation Seg_N
k=4	0.5	0.1
k=8	1.4	0.1
k=16	1.6	0.1
k=32	14.4	0.7
k=64	61.2	4.6
k=128	-	49.2

3) Comparing different segmentations: As discussed in V-B, the segmentation is chosen by the tenant security experts. In this section, we illustrate how different segmentations can impact the execution time for OCDO-2. As the table IV shows the segmentation Seg_N performs always better than the segmentation Seg_O. This can be explained by a smaller search space in the case of the former compared to the latter. We conclude that by reducing the overlap between different SMs, and by making segments with small number of SMs, we reduce the search space and therefore contribute to decrease the execution time and make the approach more scalable.

Fig 4 illustrates the penalty in percentage of costs for each segmentation scenario (Seg_O and Seg_N). The penalty is computed based on the percentage of deviation of the cost of the considered segmentation (Seg_O or Seg_N) from the cost for the no-segmentation case. One can note that Seg_N comes with higher cost gap than Seg_O even though, Seg_N performs better from execution point of view (c.f. Table IV). Therefore, there is a tradeoff between time execution and optimality; Higher execution time comes with better optimality. However, for all measured use cases of workloads, the largest gap is

[6]GNU Linear Programming Kit: kam.mff.cuni.cz/~elias/glpk.pdf

always less than 15 % for both type of segmentations for all $k \leq 64$, compared to the case of the optimal values with no segmentation.

TABLE IV. OCDO-2 EXECUTION TIME IN SECONDS AS A FUNCTION OF K FOR 2 PURCHASERS AND 4 SMS, FOR TWO SEGMENTATION Seg_O AND Seg_N. FOR $k = 128$ AND Seg_O, OCDO-2 DID NOT CONVERGE.

k	Segmentation Seg_N	Segmentation Seg_O
k=4	0.1	0.1
k=8	0.1	0.1
k=16	0.1	0.1
k=32	0.7	2.1
k=64	4.6	17.1
k=128	49.2	-

It is also worthy to note that the difference between the two penalties in both segmentations is between 0 and less than 5 percents. Thus, for large data centers, one can use Seg_N instead of Seg_O to have better execution time without loosing so much from the optimality point of view. Finally, for the case of a fat-tree of size k=128, Seg_N provides the only way for OCDO-2 to converge.

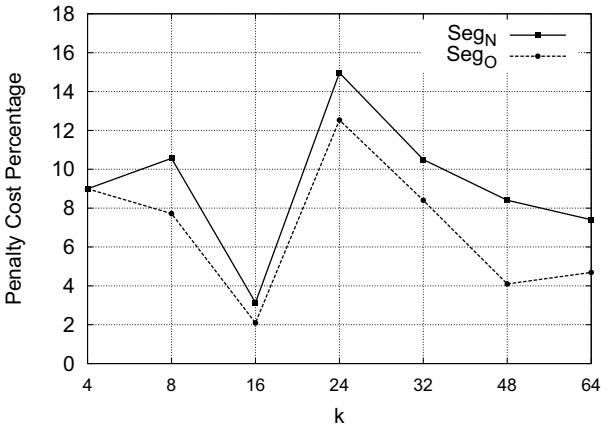

Fig. 4. Penalty Cost Percentage for Seg_O and Seg_N as a Function of k

VIII. CONCLUSION

Network functions virtualization together with software-defined networking have the potential to significantly improve network security in the cloud. Optimal placement of virtual middleboxes is challenging when dealing with large cloud data centers, multiple flows, and increased number of potential NF locations. Adding policy-awareness has been recognized to increase the difficulty of solving this problem. In this paper, we presented a novel approach to address the aforementioned challenges. Mainly, we propose an online per request placement and path optimization formulation, that is based on our extension to the well-established traveling purchaser problem. We also come up with two techniques, namely decomposition and segmentation, that composed together enable scaling to quite large data centers (128-ary fat-tree). We plan as future work to explore how our approach can address other emerging topologies.

ACKNOWLEDGMENT

This work is partly funded by NSERC Chair on Sustainable Smart Eco-Cloud, NSERC and by the NSERC/ERICSSON CRDPJ: ECOLOTIC Sustainable and Green Telco-Cloud.

REFERENCES

[1] NFV Working Group, "Network Functions Virtualisation (NFV); Use Cases V1.1.1," http://www.etsi.org/deliver/etsi_gs/nfv/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf, Oct. 2013.

[2] 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.

[3] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *NSDI*, 2012, pp. 323–336.

[4] 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.

[5] P. Quinn, S. Kumar, P. Agarwal, R. Manur, A.Chauhan, N. Leymann, M. Boucadair, C. Jacquenet, M. Smith, N. Yadav, T. Nadeau, K. Gray, B. McConnell, and K. Glavin, "Network service chaining problem statement," Working Draft, IETF Secretariat, Internet-Draft draft-quinn-nsc-problem-statement-03, August 2013. [Online]. Available: https://tools.ietf.org/html/draft-quinn-nsc-problem-statement-03

[6] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 51–62.

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

[8] 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.

[9] 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.

[10] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "Flowtags: enforcing network-wide policies in the presence of dynamic middlebox actions," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 19–24.

[11] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 413–424.

[12] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of virtualized deep packet inspection functions in sdn," in *Military Communications Conference, MILCOM 2013-2013 IEEE*. IEEE, 2013, pp. 992–997.

[13] 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

[14] 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

[15] T. Ramesh, "Traveling purchaser problem," *Opsearch*, vol. 18, pp. 78–91, 1981.

[16] A. Shameli-Sendi, Y. Jarraya, M. F. Ahmed, M. Pourzandi, C. Talhi, and M. Cheriet, "Optimal placement of sequentially ordered virtual security appliances in the cloud," in *the Proceedings of the 14th IFIP/IEEE Inter. Symposium on Integrated Network Management (to appear)*, 2015.