Paper presented at the *2017 IEEE International Conference on Cloud Computing Technology and Science* (CloudCom)
https://doi.org/10.1109/CloudCom.2017.22

Payless Monitoring Service for Tenants in Cloud with Traffic and Energy-aware Function Deployment

Alireza Shameli-Sendi*, Habib Louafi†, and Mohamed Cheriet§
*Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran
† Ericsson Security Research, Ericsson, Montreal, Canada
§ Computer Engineering Department, University of Quebec (ETS), Montreal, Canada
Email: *a_shameli@sbu.ac.ir, †habib.louafi@ericsson.com, §mohamed.cheriet@etsmtl.ca

***Abstract*—Many applications are distributed in cloud as they consist of different modules and tiers. Cloud tenant must be able to monitor the deployed applications to ensure that it is operating correctly, meeting its SLAs, and fulfilling business requirements. Activating all type of monitoring functions for all tenant's applications at the same time is costly. The more required monitoring functions the more allocated cloud resources. Moreover, it incurs monetary cost for tenants. In this paper, the problem of virtual monitoring function (vMF) placement for service chains is modeled by maximizing both the network and computing resource utilization. Beside that, we propose a set of tenant policies, which are either monetary-driven or performance-driven, translated as constrains in optimal placement algorithm. Simulation results show that the proposed model can reduce the total incurred cost by up to 10% compared to the best non-optimal heuristic. Moreover, the proposed model can decrease the execution time about 84% compared to the basic solution.**

***Index Terms*—Network Function Virtualization, Virtual Monitoring Function, Optimal Placement, Network Cost, Computing Cost**

I. Introduction

Given the sheer size and complexity of virtual network and applications in cloud, there is an increasing demand for dynamic and efficient performance monitoring [5], [10]. Because of resource limitation, it is not possible to activate monitoring functions every where. Current solutions rely on either placing limited monitoring functions on some fixed locations and re-routing the traffic over these monitoring functions, or setting some routing paths, where the monitoring functions should be deployed [6]. However, both solutions suffer limitations. The fixed place of monitoring functions limit to the traffic characteristics (e.g., load, speed). On the other hand, placing monitoring functions for each flow or link is CPU and memory intensive, and thus not practical.

Therefore, we need to find the optimal places where the monitoring functions should be deployed and re-route some flows to be monitored by those functions, in such a way to minimize the total cost of resource consumption of the monitoring functions as well as the cost of links re-routing [2], [9]. This is very challenging, as some nodes may not have sufficient resources (CPU and memory) to run any monitoring function or process all re-routed flows [7], [14]. Furthermore, the impact of deployed solution (monitoring function placement and flows re-routing) on the performance (e.g., response time) of the network and applications should be minimized by all means.

In this paper, we propose to place vMF in cloud with respect to the network and computing utilization maximization objectives. Network objective refers to minimize traffic delay for flows needed to be monitored and computing objective expresses balancing nodes computing resources. Those objectives are executed due to the tenant policies which are either monetary-driven or performance-driven. The policies are co-location, merge, distribution, and zoning.

Co-location and merge can save monetary cost for tenants when there are many applications in the cloud. Co-location policy forces different type of monitoring functions to be placed on the same node. Thus, it reduces the pre-processing cost (e.g., serialization) as it has to be done only once per node. Merge policy eliminates the same type, as much as possible, to save the cost of VM creation of the same monitoring job functionality.

Distribution policy forces the placement algorithm to place vMFs in different nodes. It improves the performance of monitoring when (a) the load of the majority of the nodes are high and they cannot accept to host several new vMFs or (b) the application may expect receiving high traffic and co-locating many vMFs on a node may affect performance. Zoning policy helps to reduce the execution time of optimal placement algorithm, since the search space is restricted. It helps to speed-up the execution of the solver, initiate VMs and vMFs creations, capture the source of the problem, and deactivate the monitoring functions. Thus, zoning helps both saving monetary cost and improving monitoring performance. Note that the performance may be negatively affected when the majority of the nodes in a zone are busy.

In short, the contributions of this paper are three-fold:

1) We identify two categories of monitoring policy placement in cloud: monetary-driven and performance-driven. The tenants may choose one or both (switching between them over time) of them, with respect to the type of deployed applications in cloud or budget limitation.
2) We formulate the virtual monitoring function placement problem using integer linear programming (ILP), where the objective is to maximize both networking and computing resources utilization while satisfying the tenant monitoring policy or limitation.

3) Note that to monitor a three-tier application, there are two requests, since two communications exist between tiers. The majority of works split the requests into different sets (same source and destination) and then run them sequentially. Off course they lose optimality but they reduce the execution time. In this paper, we propose a set of constraints which help us to reduce the execution time when all requests are run simultaneously.

The rest of the 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 our optimization framework. Section VII is dedicated to the presentation and discussion of the results obtained via different use cases. Section VIII concludes the paper.

II. Related Work

Network monitoring using OpenFlow has been explored in recent years. Adrichem et al. [13] proposed an open-source software implementation to monitor per-flow metrics, especially throughput, delay, and packet loss, in OpenFlow networks. Chowdhury et al. [3] proposed a flexible and extendable monitoring framework for SDN, called PayLess. They focused on trade-off between monitoring accuracy, timeliness and network overhead. The presented model supports different types of monitoring: performance, security, fault-tolerance, etc. For each selected monitoring type, a set of metrics is proposed. Performance metrics may include delay, latency, jitter, throughput, etc. As seen, the aforementioned work explored different virtual monitoring functions (delay, latency, jitter, throughput, packet loss) relying on OpenFlow or other programmable routing platform which are needed in the cloud. Virtual monitoring functions are part of Network Function Visualization (NFV) and the optimal placement of VNF became a hot topic in cloud recently.

The optimal placement of VNFs is known to be an NP-hard problem. Particularly, optimizing both network and computing resources at the same time may make the problem difficult to solve in a reasonable period of time. Thus, several research initiatives either propose to optimize these resources separately or to tackle the more general problem by proposing heuristics to address the pure optimization problem. In [4], an ILP formulation is proposed to optimally steer flows through static middleboxes in the network. Sekar et al. [12] propose an optimization approach that assumes predefined paths between sources and destinations and then tries to find the optimal placement of the virtual appliances on these paths. Zhang et al. [15] and Addis et al. [1] propose to prioritize the objective functions and then run them sequentially. The first step is to dynamically find the optimal computing node to initiate a service function, and then traffic is routed through the initiated service. In contrast, Addis et al. [1] minimize first the maximal link utilization; and then, after setting the routes, as a second objective, they minimize the network core utilization.

By considering both objectives dynamically, link and node core utilizations, optimal placement of a chain becomes more challenging. That is, when the number of nodes in the network increases, the running time enhances exponentially. To tackle this issue, Mohammadkhan et al. [11] propose to partition the problem into smaller pieces such that the final result remains close to the optimal solution. In contrast, Jarraya et al. [8] present some heuristics to make the result scalable. For example, they restrict the resource availability in the network with respect to the type of VNF, or they partition the graph vertically, from source to destination, into some several sub-graphs. Compared to the existing work, we propose a set of monitoring policies considered as constraints in finding the optimal solution. These policies enable tenant to optimize the monetary costs or performance based on their needs.

III. Problem Statement

Let us have a virtual network comprised of a set of nodes $N = \{n_i\}$, and a set of links $L = \{l_{n_i,n_j}\}$. To each link l_{n_i,n_j}, we associate a cost of sending a packet on that link, denoted as $C_l(l_{n_i,n_j})$, n_i and n_j being source and destination. Let $F = \{f_i\}$ be a set of network flows, to each f_i of which is associated a bandwidth capacity needed by that flow, denoted as $B_f(f_i)$.

We want to deploy a set of k vMFs, denoted by $M = \{m_i\}_{i=1}^{k}$, on the network. Deploying each vMF incurs a cost (vCPUs and memory), which depends on the network node on which it will be deployed. Let $C_n(m_i, n_j)$ be the cost of deploying the vMF m_i on the network node n_j.

Our objective is find the optimal placements where the vMFs can be deployed. These optimal placements should minimize, simultaneously, the cost of deploying the set of vMFs, the cost of re-routing the other flows to be monitored by the deployed vMFs, while satisfying the tenant policies.

Let $M^* = \{m_i^*\}_{i=1}^{k}$ be the optimal placements of $M = \{m_i\}_{i=1}^{k}$, which are given by:

$$M^* = \underset{n_i \in N, l_i \in L, f_i \in F, m_i \in M}{\arg\min} Cost(n_i, m_i, l_{n_i,n_j}, f_i) \quad (1)$$

where $Cost(n_i, m_i, l_{n_i,n_j}, f_i)$ is the total cost of deploying the k vMFs and re-routing the flows that are not monitored to be monitored by these vMFs.

IV. Proposed Optimal Placement Framework

In this section, we present our solution to equation (1), in order to identify the optimal placements of the set of k vMFs. We first present an overview of our framework, then we detail our methodology.

A. Overview

Our framework takes as input (i) the network topology, (ii) the cost of link between nodes, (iii) the cost of nodes, (iv) number of different vMF instances that can be instantiated at each node, (v) maximum capacity in units of bandwidth supported by each link, (vi) the number of flows, and (vii) the number of vMFs that will be monitoring each flow. On the other side, the framework identifies the set of nodes, where the k vMFs can be deployed optimally. The core of our framework consists in solving (1), which is detailed in the following

TABLE I: ILP parameters and variables

$N = \{n_i\}$	set of network nodes
$L = \{l_{n_i,n_j}\}$	set of links of the network
$F = \{f_i\}$	set of (unidirectional) flows between pairs of nodes
$M = \{m_i\}_{i=1}^{k}$	set of possible vMFs
$C_l(l_{n_i,n_j})$	cost of sending a packet on the link l_{n_i,n_j}
$B_f(f_i)$	bandwidth capacity needed by the flow f_i
$C_n(m_i, n_j)$	cost of deploying the vMF m_i on the network n_j
$M^* = \{m_i^*\}_{i=1}^{k}$	optimal placements of $M = \{m_i\}_{i=1}^{k}$
$x(l_{n_i,n_j}, f_i)$	decision of re-routing flow f_i on the link l_{n_i,n_j}
$y(n_i, m_i, f_i)$	decision of placing vMF m_i on the node n_i to monitor flow f_i
s_f	node source of the flow f, $s_f \in N$
d_f	node destination of the flow f, $d_f \in N$
L_{n_i,n_j}^{max}	maximum capacity in units of bandwidth supported by link l_{n_i,n_j}
$d_{l,f}$	number of instances vMF of type l required by the flow f
$q_{n_i,l}$	number of instances vMF of type l that can be instantiated at node n_i
$K_f \subseteq M$	set of vMFs that must be traversed by the flow f
$v_{n_i,n_j,f}$	positive integer variable representing the visiting order of the successor links along the path formed by the solution from d_f to d_f
H_{min}	the minimum number of hops for the optimal routing solutions
H_{max}	the maximum number of hops for the optimal routing solutions
Lcolocation	list of vMFs that need to be co-located
$CFlow_m$	list of flows that need to co-locate vMF m
Lmerge	list of vMFs that need to be merged
$MFlow_m$	list of flows that need to merge vMF m
Ldistribution	list of vMFs that need to be distributed
$DFlow_m$	list of flows that need to distribute the vMF m

subsection. Table I summarizes the mathematical symbols used in this paper.

B. Methodology

To solve (1), we model the total cost $Cost$ using integer linear programing. Therefore, we propose minimizing the following cost function:

$$\begin{aligned} &Cost(n_i, m_i, l_{n_i,n_j}, f_i) = \\ &\sum_{f\in F}\sum_{n_i\in N}\sum_{n_j\in N}\Big(C_l(l_{n_i,n_j})\cdot B_f(f_i)\cdot x(l_{n_i,n_j}, f_i)\Big) + \\ &\sum_{f\in F}\sum_{n_i\in N}\sum_{m_i\in M}\Big(C_n(m_i, n_j)\cdot B_f(f_i)\cdot y(n_i, m_i, f_i)\Big) \end{aligned} \tag{2}$$

where, $x(l_{n_i,n_j}, f_i) \in \{0, 1\}$ indicates whether the link l_{n_i,n_j} is selected for re-routing the flow f_i or not, while $y(n_i, m_i, f_i) \in \{0, 1\}$ indicates whether the node n_i is selected to host the vMF m_i and be traversed by the flow f_i or not. The two costs $C_l(l_{n_i,n_j})$, and $C_n(m_i, n_j)$ are bounded between 0 and 1.

At a high level, we want to decide, for each flow f_i generated by a source s_f and consumed by a destination d_f which nodes have to be visited and where the vMFs have to be placed. These are captured, in our formulation, by the two binary variables $x(l_{n_i,n_j}, f_i)$ and $y(n_i, m_i, f_i)$. The solution to our optimization problem is the set of $x(l_{n_i,n_j}, f_i)$ and $y(n_i, m_i, f_i)$, which together denote the optimal placements of the k vMFs and the optimal path to traverse them.

Our proposed formulation is a load-balancing function that tries to assign the same workload to all potential vMFs locations, which leads to minimize the maximum workload among them. On the other hand, it attempts to balance the link utilization for all routing paths.

C. Basic Constraints

In the following, we define a set of constraints that represent the baseline for our optimization framework, where both computing and networking resources are optimized simultaneously.

Each flow produced by a source node is consumed only by the destination node and no part is consumed by intermediary nodes. In other words, for any node n_i, the sum of its in-degree edges is equal to the sum of its out-degree edges, which should be one. That is:

$$\begin{aligned} &\forall n_i \in N, \forall f \in F \\ &\sum_{n_j\in N} x(l_{n_j,n_i}, f) + (\text{if } n_i = s_f \text{ then } 1) = \\ &\sum_{n_r\in N} x(l_{n_i,n_r}, f) + (\text{if } n_i = d_f \text{ then } 1) \end{aligned} \tag{3}$$

Note that, the exact quantity of vMFs requested should be instantiated. Let k be the number of vMFs requested by flow f, thus, we have:

$$\sum_{n_i\in N} y(n_i, m, f) = k \qquad \forall m \in M, \forall f \in F \tag{4}$$

vMFs can only be deployed on nodes where required resources (vCPUs and memory) are available. Let $card(MFlow_m)$ be the number of flows that need to merge the vMFs m. In some scenarios, for some reasons, we may merge the same type of vMFs of two communications. The advantage of merging vMFs is, first, less network resources are consumed, second, the more merging the same type of vMFs, the effective the execution time. Formally, we have:

$$\sum_{f\in F} y(n_i, m, f) - q_{n_i,m} * card(MFlow_m) \leq 0, \ \forall n_i \in N, m \in M \tag{5}$$

A node is added to the solution set, if a vMF is to be deployed at that node. That is:

$$\begin{aligned} &\forall n_i \in N \setminus \{s_f\}, m \in M, f \in F \\ &\sum_{n_j\in N} x(l_{n_i,n_j}, f) - y(n_i, m, f) \geq 0 \end{aligned} \tag{6}$$

The bandwidth allocated to flows traversing a link does not exceed the maximum capacity of that link. That is:

$$\sum_{f\in F} B_f(f)\cdot x(l_{n_i,n_j}, f) \leq L_{n_i,n_j}^{max} \qquad \forall n_i \in N, n_j \in N \tag{7}$$

No cycle is permited in the obtained solution paths. That is:

$$\begin{aligned} &\sum_{n_i\in N} x(l_{n_i,n_j}, f) \leq 1 \qquad \forall n_j \in N, f \in F \\ &\sum_{n_j\in N} x(l_{n_i,n_j}, f) \leq 1 \qquad \forall n_i \in N, f \in F \end{aligned} \tag{8}$$

The obtained solution should have low impact on SLA. To do so, the number of hops between a source node and a destination node is bounded between H_{min} and H_{max}, which represent, respectively, the minimum and maximum number of hops in the optimal solution. These two boundaries can be adjusted by: (i) the position of s_f and d_f in the network, (ii) the status of resource availability in nodes, and (iii) the number of vMFs in the chain. Note that, the number of vMFs in the chain solution may increase the routing path, when some of them are co-located or merged on the same node. For example, if s_f and d_f are located in two different pods in fat-tree, at least six hops are needed to traverse the tree between s_f and d_f. Since a path solution with less than six hops is not realistic, they should be ignored by the framework. In this case, H_{min} is sets to 6.

$$\sum_{i \in N} \sum_{j \in N} x(l_{n_i,n_j}, f) \leq H_{max} \tag{9}$$

$$\sum_{i \in N} \sum_{j \in N} x(l_{n_i,n_j}, f) \geq H_{min} \tag{10}$$

Finally, since this problem at hands is NP-hard, any constraint which helps to limit the search space should be considered. Such constraints help the framework eliminating the nodes that are not in the solutions space. For instance, in the fat-tree topology, never a solution meets the host in the routing path, except source and destination nodes. Therefore, we define a region of nodes in the graph which should not be visited, called $\bar{R}$. Nodes in $\bar{R}$ satisfy the following property:

$$\sum_{n_j \in \bar{R}} x(l_{n_i,n_j}, f) \leq 0 \quad \forall n_i \in N, f \in F \tag{11}$$

D. Monetary or Performance Constraints

1) Merge: It is possible to place several vMFs, of the same type, on the same node, as one vMF. For each two different flows f_1 and f_2, such that f_1 and f_2 are in MFlow_m, then both flows should meet the same vMF at that node. That is:

$$\begin{aligned} \forall n_i \in N, m \in \mathit{Lmerge}, f_1, f_2 \in \mathit{MFlow}_m, f_1 \neq f_2, \\ y(n_i, m, f_1) = y(n_i, m, f_2) \end{aligned} \tag{12}$$

where, Flow_m is the set of flows that traverse the vMF m, and *Lmerge* is the set of vMFs that should be merged on the same node.

2) Co-location: In some cases, we may need to co-locate different type of vMFs of a flow on the same node. This is given by:

$$\begin{aligned} &\forall f_1 \in \mathit{CFlow}_m, f_2 \in \mathit{CFlow}_{m'}, f_1 = f_2, \\ &\forall n_i \in N, m, m' \in \mathit{Lcolocation}, m \neq m' \\ &y(n_i, m, f_1) = y(n_i, m', f_2) \end{aligned} \tag{13}$$

where, m and m' are two different type of vMFs.

3) Distribution: In some cases, we need to place some vMFs at the different locations, i.e. distributed on different nodes. First, we define a set of vMFs that should be distributed, denoted by *Ldistribution*, and the set of flows that should pass through each of these vMFs, denoted by Flow_d. This is formulated as follows:

$$\begin{aligned} &\forall n_i \in N, m, m' \in \mathit{Ldistribution}, \\ &\forall f_1, f_2 \in \mathit{DFlow}_m, f_1 \neq f_2 \\ &y(n_i, m, f_1) = 1 - y(n_i, m', f_2) \end{aligned} \tag{14}$$

4) Zoning: Some regions of nodes are defined in the network, which represent the monitoring zones (denoted MZ). A vMF that should be deployed in ZM should satisfy the following equation:

$$\sum_{n_i \in MZ} y(n_i, m, f) = 1 \qquad \forall m \in K_f, f \in F \tag{15}$$

where $K_f \subseteq M$ is the subset of vMFs that should be traversed by the flow f.

V. Experimental Results

A. Simulation Setup

The experiments run for fat-tree $k = 8$, consists of 200 nodes and 386 links. In order to evaluate the presented monitoring placement algorithm, we assume that a tenant has 50 three-tier applications are distributed randomly in different racks. Moreover, we assume that the tiers of each application are not placed in the same rack. The tenant does need these 50 three-tier applications to be monitored in the beginning.

Assume that the tenant target is monitoring these 50 applications in 25 different times, each time two applications with respect to two polices both together: merge and distribution. To monitor each flow, three monitoring functions are needed to be deployed to measure delay, packet-loss, and jitter. Therefore, in each placement, 12 vMFs are needed in total for 6 tiers of two three-tier applications (see Figure 1). The merge policy, which saves the cost of initiating VMs to place vMFs and the same job functionality, reduces the total number from 12 to 3. The distribution policy enforces the solver algorithm to place these three vMFs on three different nodes.

To initialize b_f, the number of units of bandwidth needed by a flow f, a random number is generated. To have this flexibility that all flows can pass all links, b_f of each flow should be less than the available capacity in units of bandwidth in link in the network, divided by the number of flows in each execution, which is four in our paper. For the first placement, we set the network such that the load of links and nodes vary. 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 vary dramatically according to time, random values are used for $c_{i,j}$ and $b_{i,l}$. When the vMFs are placed on the optimized nodes and the flows are set to be steered through vMFs from sources to destinations, the load of related nodes and links are increased. Then, the network with proper costs is fed to the inputs of the next placement execution. For performance testing and finding the optimal solution, the

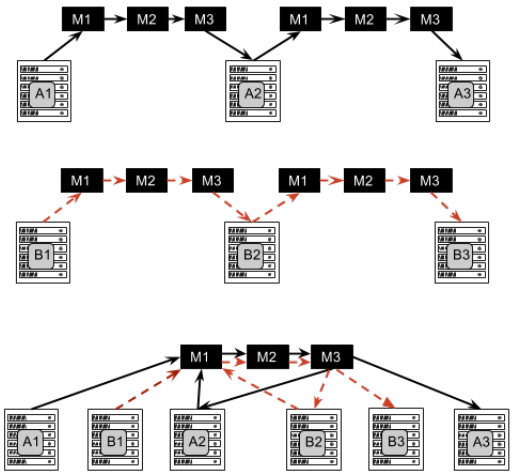


Fig. 1: Two distributed applications and three monitoring functions needed to monitor different measurements between tiers.

GLPK (GNU Linear Programming Kit) is used on a machine with 6 cores Intel Westmere (XEON L5638) clocked at 2.30 GHz with 24 GB RAM.

B. Result

In this section, we compare our heuristic, called *Min-NCU* (Minimizing the maximum Network and Computing Utilization), in terms of resource utilization, cost, scalability, with four different heuristics: *Min-NU*, *Min-CU*, *Min-NU->CU*, and *Min-CU->NU*. In Min-NCU, both objectives, network and computing resources utilization are considered simultaneously to tackle the optimal placement. Min-NU only optimizes the network resource utilization, routing cost, from a source (a tier) to a destination (another tier). In contrast, Min-CU only optimizes the computing resource utilization. It finds the best nodes (less costly) to place vMFs without taking into account the routing cost. In Min-NU->CU and Min-CU->NU, the placement objectives are prioritized and then run sequentially. In Min-NU->CU, first the best routing from a source to a destination is identified and then the best nodes within the best routing path are discovered. In contrast, Min-CU->NU first finds the best nodes and then attempts to find the best path to traverse from a source to a destination by visiting all selected best nodes.

1) Utilization: Figure 2 illustrates computing, network, and total resources utilization behavior for five different heuristics for 25 requests of optimal placements over time. Figures 2a and 2b show the result of two opposite heuristics, one is seeking to minimize only the maximum network utilization while the other one is seeking to minimize the maximum computing utilization. In Min-NU, the harmony is observed in the network utilization. In contrast this harmony for the Min-CU heuristic is seen in the computing utilization. Due to the high number of hops in routing path, compared to the number of nodes to place monitoring functions, Min-CU shows a non-harmony behavior for the second objective.

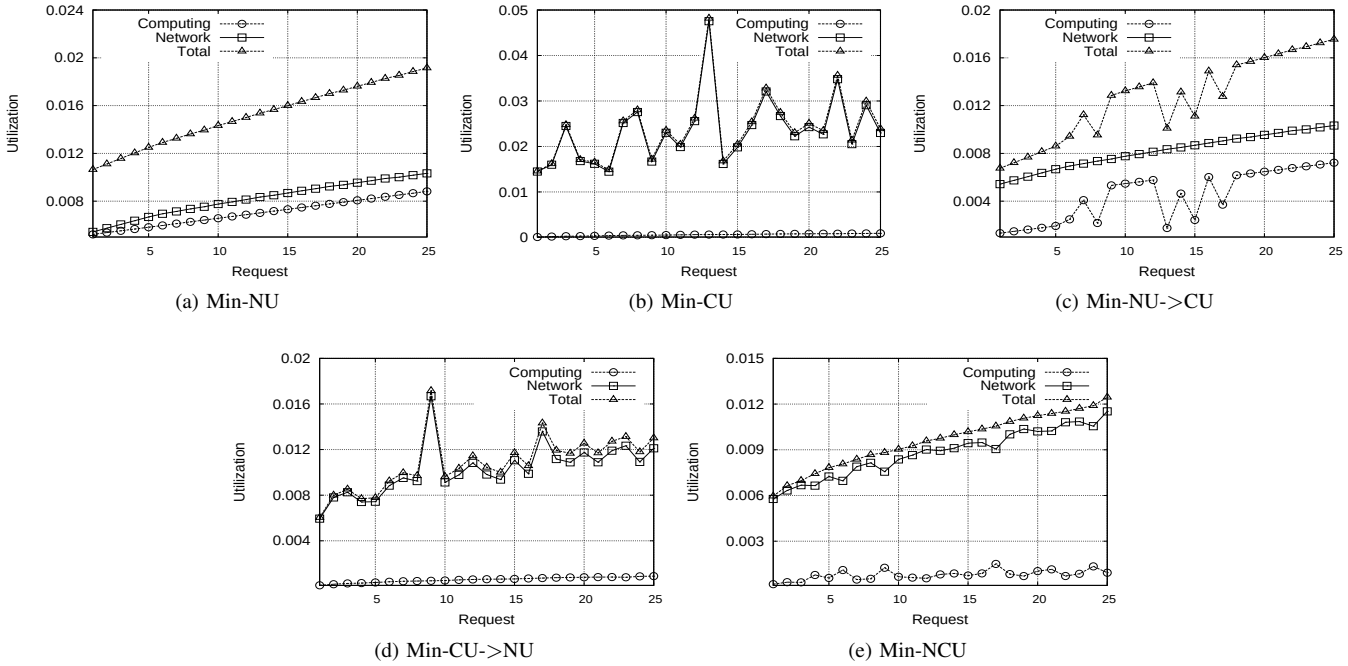
As seen in Figure 2b, selecting the less costly nodes to place vMFs does not guarantee traversing the cheap routing path from a source to a destination. The prioritized heuristics, Min-NU->CU and Min-CU->NU, attempt to address this limitation as shown in Figures 2c and 2d. We compare the result of these prioritized heuristics in Figure 3. The objective function in Min-NCU is to minimize the maximum network and computing utilization simultaneously. Thus, as seen in Figure 2e the harmony is seen in sum of network and computing utilization. For example, compare the request four and five. In request five, the heuristic finds the less costly nodes to place vMFs compared to request four, but the routing cost of the solution for request five is greater than request four.

Figure 3a compares five different heuristics in terms of network utilization. Min-NU and Min-NU->CU has the best load-balancing between links compared to other heuristics since they target only network cost objective. As seen, the result of Min-NCU (our heuristic) is very close to the best result. Since Min-CU does not take into account the network cost, it has the highest network utilization. As mentioned, Min-CU->NU attempts to have better network utilization compared to the Min-CU, since it finds the best routing among the source, all best nodes to place vMFs, and the destination. As seen, Min-CU->NU has better result compared to the Min-CU.

Figure 3b compares five different heuristics in terms of computing utilization. As expected Min-NU and Min-NU->CU have the highest value compared to others. The result of Min-NCU is very close to the best result which are Min-CU and Min-CU->NU. As seen, for requests 12, 19, and 22 the result of Min-NCU is better than Min-CU and Min-CU->NU. The reason is that Min-NCU does not chose the best nodes and there is always this opportunity to select the best nodes for the coming requests.

Figure 3c illustrates the total utilization, network and computing together, for five heuristics. Ignoring the cost of the network makes Min-CU the worst among the heuristics. Since the contribution of network cost is more than the computing cost for each request, Min-NU has better result compared to the Min-CU. As mentioned, Min-NU->CU and Min-CU->NU attempt to improve the result of Min-NU and Min-CU, respectively. Although Min-NU performs better than Min-CU, but the result of Min-CU->NU is better than Min-NU->CU. Regarding the network utilization, in Figure 3a we observe that the result of Min-CU->NU is very close to the Min-NU->CU, but the big difference for computing utilization, as seen in Figure 3b, makes the Min-CU->NU to have better result compared to the Min-NU->CU. It refers to this fact that when the best nodes are selected first, we have better flexibility to chose the best routing among the source, selected nodes, and destination, compared to selecting the best routing first. This is because of less flexibility among nodes in the selected routing, compared to the first approach. In contrast, Min-NCU has the best total utilization since it seeks to minimize the maximum utilization of network and computing simultaneously.

Figure 4 shows the improvement percentage of Min-NCU



(a) Min-NU (b) Min-CU (c) Min-NU->CU

(d) Min-CU->NU (e) Min-NCU

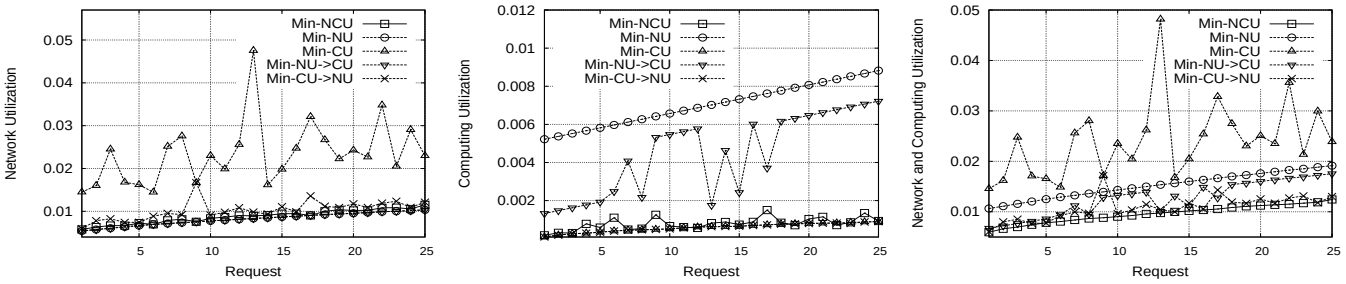
Fig. 2: Computing and network utilization behavior for five different heuristics for 25 placements over time. The placement scenario for each request consists of two three-tier applications and two policies, merge and distribution. It was performed for a fat-tree of size k=8.

compared to other heuristics in terms of total utilization. The improvements of Min-NCU compared to the Min-NU and Min-CU, which seek to optimize only one objective, are 54% to 79% and 41% to 80%, respectively. The improvement compared to the Min-NU->CU is between 3% and 32%. As mentioned in the Figure 3c, the best heuristic close to the Min-NCU is Min-CU->NU. The improvement compared to the Min-CU->NU is between -0.8% and 49%. For two requests, 5 and 24, Min-CU->NU has better result (which is difficult to be seen in Figure 3c). The nodes which are selected now by Min-CU->NU are partially available for Min-NCU, since Min-NCU used them to minimize another objective function, network cost. As a conclusion, for the explained scenario which involves 25 requests, we see 10.4% improvement in the total utilization of network and computing resources, compared to Min-CU->NU (the best heuristic close to our solution).

2) Cost: Figure 5 compares five different heuristics in terms of cost for a fat-tree of size k=8. The cost value is the total of all costs for 25 requests. As can be seen in Figure 5, when only one objective is optimized, computing or network resource utilization, the cost of the non-considered objective in the respective heuristic remains high. For example, Min-NU, has the lowest cost for network, but has the worst one for the computing. Similarly, Min-CU has the lowest cost for computing, but has the worst one for network. As seen, the total costs of Min-NU->CU and Min-CU->NU heuristics are close (near-optimal) to Min-NCU. For example, Min-NU->CU has better network cost, compared to Mac-NCU since in the first iteration it finds the best routing, while the best nodes may not exist in the best routing. In contrast, Min-CU->NU suffers the same problem for the non-best routing path through the best nodes of placing monitoring functions. As can be seen, by considering both objectives, Min-NCU is able to attune both costs simultaneously. Thus, our approach has the minimum total cost for each placement request since the network and computing costs are increased together.

Figure 6 compares the result of Min-NCU with other heuristics for each objective separately. We compare the computing cost to the best heuristic, Min-CU and the network cost to Min-NU. We observe that for network resource cost, Min-NCU is at most 11.5% more costly compared to the best case while Min-NCU is at most 172.8% more costly for computing cost. As seen, four times the computing cost passes 100% and three times Min-NCU has better cost. The reason for the worst computing cost is that the best nodes does not guarantee to minimize the routing cost. The reason for the best computing cost is that Min-NCU does not chose the best nodes and there is always this opportunity to select the best nodes for the coming requests.

3) Scalability: As explained, the optimal placement of vMFs is known to be an NP-hard problem. Thus, we need some heuristics to obtain the result in the acceptable time. Figure 7 shows the benefit of some important constraints to improve the execution time for the case of a fat-tree of size k=8 (it consists of 200 nodes and 384 links). For this result, we consider a three tier application. These three tiers, $A1$, $A2$, and $A3$, are placed on racks 0, 40, and 127, respectively (Note that there are 128 racks in fat-tree k=8). Our placement goal is placing three vMFs in the network such that



(a) Network utilization (b) Computing utilization (c) Total utilization

Fig. 3: Comparing utilization behavior for five different heuristics for 25 placements over time. The placement scenario consists of two three-tier applications and two policies, merge and distribution. It was performed for a fat-tree of size k=8.

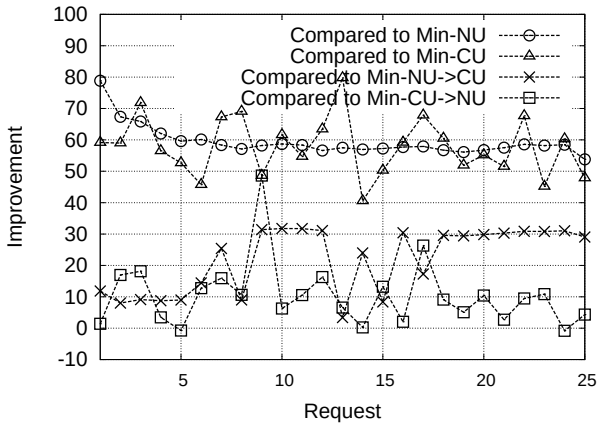


Fig. 4: Min-NCU improvement in the total utilization compared to other heuristics.

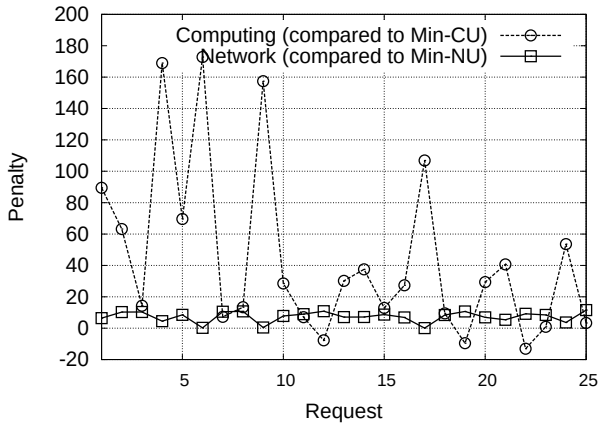


Fig. 6: Min-NCU penalty cost compared to Min-NU and Min-CU.

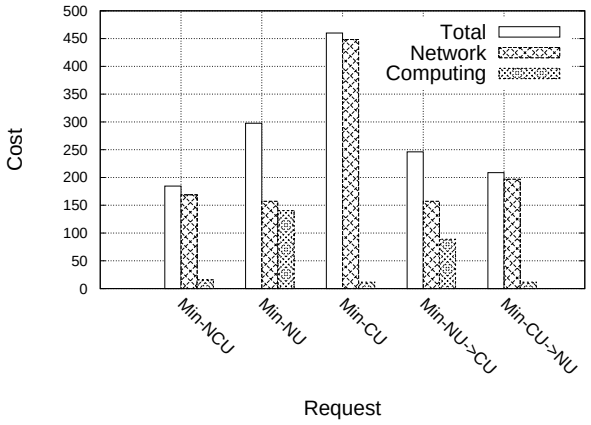


Fig. 5: Comparing five different optimal placement heuristics in term of placement cost for the case of a fat-tree of size k=8.

the communication between $A1$ and $A2$ and communication between $A2$ and $A3$ meet the same monitoring functions (the six vMFs are merged as three).

Figure 7 illustrates nine executions of optimal placement algorithm by considering different constraints. *1) No-Limitation:* In the first execution, there is no limitation and we can visit any link and also the monitoring functions can be placed at any node. In this case, the placement algorithm is executed by the basic constraints (see Section IV-C Eq. (1)-(8)). As can be observed, the algorithm takes 90.4s to find the optimal solution. *2) Routing (R):* We added two new constraints to the first execution, Eq. (9)-(10), to limit the routing paths from source to destination. It improves 20% the execution time, 72s. *3) No-Visit (NV):* In the fat-tree topology, never a solution visits any node in the host level from a source to a destination. Of course never the solver ends up with such a solution but as seen in Figure 7, it speeds up the solution from 90.4s to 57.2s by considering Eq. (11). *4) NV+R:* We considered the constraints Eq. (9)-(10) and Eq. (11) together, limiting routing and no-visiting host nodes. As seen, it improves the execution time significantly compared to the first execution, 65% (from 90.4s to 31.4s). Moreover, it improves the time 56% and 45%, compared to the second and third execution, respectively, where those are executed independently. 5) No-Place (NP): In this execution we added three constraints to the first execution to prevent placing vMF in three regions, core (R_c), edge (R_e), and host (R_h) as follows:

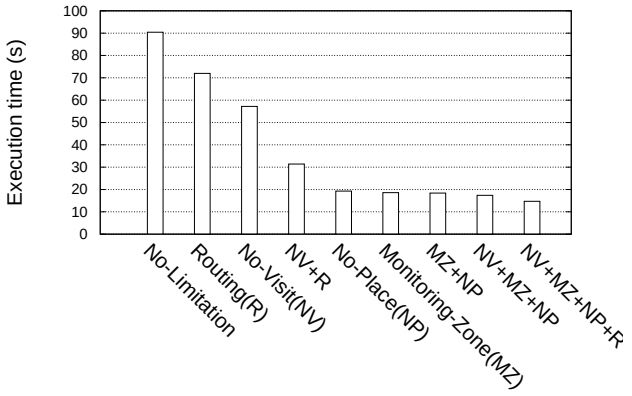


Fig. 7: Comparing the benefit of different constraints in the execution time for placing three vMFs for a three-tier application for the case of a fat-tree of size k=8.

$$\sum_{j \in R_c = \{0..127\}} y_{j,l,f} = 0 \qquad \forall l \in K_f, f \in F \tag{16}$$

$$\sum_{j \in R_e = \{128..159\}} y_{j,l,f} = 0 \qquad \forall l \in K_f, f \in F$$

$$\sum_{j \in R_h = \{192..199\}} y_{j,l,f} = 0 \qquad \forall l \in K_f, f \in F$$

6) *Monitoring Zone (MZ):* In the sixth execution, the monitoring zone constraint Eq. (15) was added to the first execution. This constraint enforces that the vMFs are placed in a region of nodes in the graph which represents the monitoring zone. Constraints added to the executions five and six are opposite to each other. As seen, the result of execution six is better than five (compare 18.6 to 19.3s). 7) *MZ+NP:* In the next execution, seven, we wanted to see the benefit of having the opposite constraints together, monitoring-zone and no-place. The improvement was about 0.2s (18.6s to 18.4s). 8) *NV+MZ+NP:* For the execution eight, no-visit constraints (those added to execution three) were added to this execution and the improvement, compared to the previous one, was 1s. 9) In the final execution, we considered all constraints together (NV, R, MZ, and NP), and as seen it had the best result among all executions. The final execution improves 84% the first execution time (from 90.4s to 14.7s).

VI. Conclusion

The emerging paradigm of Network Function Virtualization (NFV) aims to tackle the high operational costs of hardware-based dedicated boxes by replacing network functions with software counterparts that are referred to as Virtual Network Function (VNF). One of the benefit of using NFV is the simplicity in the implementation of network services (e.g., monitoring or security) by exploiting the important concept of service chaining. As cloud applications and networking grow more sophisticated, there is an increasing demand for practical and reliable monitoring systems. Current monitoring function deployments either place the monitoring functions on fixed locations and re-route some flows to be monitored by those monitoring functions, or place the monitoring functions for fix routing. However, both strategies are inefficient, as either computing resources cost is optimized or networking cost. In this paper, we propose an optimal deployment solution for monitoring functions in cloud, which takes into account a trade-off between the cost of deploying monitoring function on network nodes (vCPUs, memory) and the cost of re-routing some flows with respect to the tenant monitoring policies which is either monetary or performance driven. The policies are translated to constraints in ILP, which save the monitoring cost, operational cost, networking cost, or improve the monitoring performance.

References

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

[2] C.-W. Chang, G. Huang, B. Lin, and C.-N. Chuah. Leisure: Load-balanced network-wide traffic measurement and monitor placement. *IEEE Transactions on Parallel and Distributed Systems*, 26(4):1059–1070, 2015.

[3] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[4] 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.

[5] N. Grover, N. Agarwal, and K. Kataoka. liteflow: Lightweight and distributed flow monitoring platform for sdn. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–9. IEEE, 2015.

[6] G. Huang, C.-W. Chang, C.-N. Chuah, and B. Lin. Measurement-aware monitor placement and routing: a joint optimization approach for network-wide measurements. *IEEE Transactions on Network and Service Management*, 9(1):48–59, 2012.

[7] H. Huang, S. Guo, J. Wu, and J. Li. Service chaining for hybrid network function. *IEEE Transactions on Cloud Computing*, 2017.

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

[9] X. Li, H. Wu, D. Gruenbacher, C. Scoglio, and T. Anjali. Efficient routing for middlebox policy enforcement in software-defined networking. *Computer Networks*, 110:243–252, 2016.

[10] G. Liu and T. Wood. Cloud-scale application performance monitoring with sdn and nfv. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pages 440–445. IEEE, 2015.

[11] 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. *Mij*, 101:1.

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

[13] N. L. Van Adrichem, C. Doerr, F. Kuipers, et al. Opennetmon: Network monitoring in openflow software-defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–8. IEEE, 2014.

[14] Y. Yu, C. Qian, and X. Li. Distributed and collaborative traffic monitoring in software defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 85–90. ACM, 2014.

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