Optimizing NFV Chain Deployment Through Minimizing the Cost of Virtual Switching*

Marcelo Caggiani Luizelli†
UNIPAMPA and *UFRGS*, *Brazil*
marceloluizelli@unipampa.edu.br

Danny Raz†
Technion, *Israel*
danny@cs.technion.ac.il

Yaniv Sa'ar
Nokia, *Bell Labs*
yaniv.saar@nokia-bell-labs.com

***Abstract*—Network Function Virtualization (NFV) is a novel paradigm that enables flexible and scalable implementation of network services on cloud infrastructure. A key factor in the success of NFV is the ability to dynamically allocate physical resources according to the demand. This is particularly important when dealing with the data plane since additional resources are required in order to support the virtual switching of the packets between the *Virtual Network Functions* (VNFs). The exact amount of these resources depends on the way service chains are deployed and the amount of network traffic being handled.**

Thus, orchestrating service chains that require high traffic throughput is a very complex task and most existing solutions either concentrate on handcrafted tuning of the servers to achieve the needed performance level, or present theoretical placement functions that assume that the switching cost is part of the input. In this work, we bridge this gap by presenting a deployment algorithm for service chains that optimizes performance by minimizing the actual cost of virtual switching. The results are based on extensive measurements of the actual switching cost and the performance of service chains in a realistic NFV environment. Our evaluation indicates that this new algorithm significantly reduces virtual switching resource utilization when compared to the de-facto standard placement in *OpenStack/Nova* – allowing a much higher acceptance ratio of network services.

I. Introduction

Despite the ever-growing popularity of *Network Function Virtualization* (NFV), we are still far away from having large scale fully operational NFV networks. One of the main obstacles on this path is the performance of the network functions in the virtual environment. The hardware middleboxes that are in use by network operators today are specifically designed to provide the needed high performance (and high reliability), but getting the same level of performance from commercially off-the-shelf hardware is much more challenging. Hardware accelerators (such as DPDK and SR-IOV) were developed specifically for this purpose, yet the deployment of high performance service chains in a virtual environment still remains a complex handcrafted process (e.g., as indicated by the reports in [1] and [2]).

* This paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 671566 ("Superfluidity"). This paper reflects only the author's views and the European Commission is not responsible for any use that may be made of the information it contains.

† Work done while in Nokia, Bell Labs.

As a result, service chain placement in such scenarios is mostly static and operators lose one of the main attractive features of NFV– the ability to dynamically allocate resources according to the current need. Such a dynamic mechanism would allow a much more efficient utilization of resources, since the same physical resource can be used by different *Virtual Network Functions* (VNFs) when needed. Thus, achieving both high performance and agility, by being able to dynamically change the resource allocation of service chains, remains a great challenge.

Identifying near optimal deployment mechanisms for NFV service chains has recently received significant attention from both academia and industry ([3]–[15]). However, to the best of our knowledge, existing studies do not take into account the cost of resources required to steer the traffic within a chain, which is non-negligible for deployment of packet intensive chains such as in the domain of NFV. Therefore, typical models (e.g., in NFV orchestrators) might either lead to infeasible solutions (e.g., in terms of CPU requirements) or suffer high penalties on the expected performance.

One noticeable exception is the very recent paper [16] that focuses on evaluating and modeling the virtual switching cost in NFV-based infrastructure. Virtual switching is an essential building block that enables flexible communication between VNFs but it also comes with an extra cost in terms of computing resources that are allocated specifically to software switching in order to steer the traffic through running services (in addition to computing resources required by the VNFs). This cost depends primarily on the way the VNFs are internally chained, packet processing requirements, and accelerating technologies (such as DPDK [1]).

Figure 1 illustrates a possible deployment of four service chains on three identical physical servers (*A*, *B* and *C*). As one can see, service chain φ^1 is composed of three VNFs – $\varphi^1 = \langle \varphi^1_1, \varphi^1_2, \varphi^1_3 \rangle$, φ^2 is composed of four VNFs, φ^3 is composed of five, and φ^4 is composed of two VNFs. In the depicted deployment (shown in Figure 1), servers *A* and *C* have the same number of deployed VNFs and thus may have the same computing resource requirement for processing. However, determining the amount of processing resources (CPU)

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

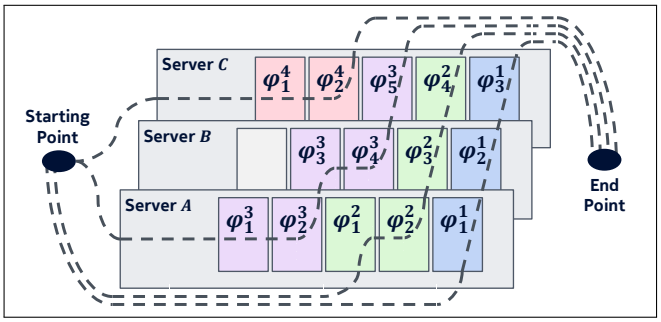


Fig. 1: Example of possible deployment of four service chains on top of three physical NFV servers

needed for switching inside these server is far from being straightforward. In some cases, the deployment can be infeasible due to lack of sufficient computing resources for the switching task. The amount of computing resources needed for the internal switching depends on the structure of the chaining in the server, and the amount of traffic associated with each chain.

Existing work that measure and evaluate the performance of the internal switching (e.g. [2], [16]–[18]) mainly focus on two types of deployment strategies: a *distribute* strategy where each VNF in the chain resides on a different server (for example, service chain φ^1 in Figure 1), and a *gather* strategy where the entire chain is grouped on the same server (for example, service chain φ^4 in Figure 1). In [2] these two deployment strategies are respectively referred to as "north/south" and "east-/west" deployments. The work in [16] analyzes these two placement strategies, and develops a cost function that predicts the amount of computing resources needed for the internal switching. Interestingly, OpenStack/Nova ([19]) global policies also follows these two deployment strategies. Namely, after a basic filtering step that clears resources according to local-server constraints (e.g., CPU cores, memory, affinity, etc.), it implements two types of global decisions that boil down to: *load balancing* of a certain metric (i.e., spread VNFs across servers), and; *energy saving* of a certain metric (i.e., stack VNFs up to the limit).

In this paper, we consider arbitrary deployments (like the one described in Figure 1) and develop a general service chain deployment strategy that considers both the actual performance of the service chains as well as the needed internal switching resource. This is done by decomposing service chains into sub-chains and deploying each such sub-chain on a (possibly different) physical server, in a way that minimizes the total switching overhead cost. Of course, there are exponentially many ways to map the sub-chains to servers. We introduce a novel algorithm based on an extension of the well-known reduction from the weighted matching to the min-cost flow problem, and show that it gives a near optimal solution with a better running time than exhaustive search.

We evaluate the performance of our algorithm against the fully distribute or fully gather solutions, which are very similar to the placement of the *de-facto* standard mechanism commonly utilized on cloud schedulers (e.g., OpenStack/Nova with load balancing or energy conserving weights) and show that our algorithm significantly outperforms these heuristics (up to a factor of 4 in some cases) with respect to operational cost and the ability to support additional network functions.

The main contributions of this paper are:

(i) **NFV deployment cost model.** We develop a general switching cost model that predicts the switching related CPU cost for arbitrary deployments and evaluate its accuracy over a real NFV environment.

(ii) **Optimal deployment mechanism.** We develop an efficient online placement algorithm that uses this new cost model to minimize the switching cost of service chain requests. We evaluate the expected performance of this novel algorithm and show that it can significantly increase utilization by allowing more network functions to run on the same NFV infrastructure in a more efficient way.

The rest of the paper is organized as follows. In Section II we define our model and problem. In Section III we develop the cost function for the virtual switching. In Section IV we describe the proposed algorithm. In Section V we evaluated the performance and the quality of our algorithm, and in Section VI we discuss related works. Finally, in Section VII we conclude our work and provide future directions.

II. Model and Problem Definition

The recommended best practice for virtualization intense environment requires each server to allocate two disjoint sets of CPU core, one set for the hypervisor to provision resources and another set for the VNF to operate ([17]). Given a server S we denote by S^h the number of CPU cores that are allocated and reserved solely for the hypervisor to operate, by S^v the number of CPU cores reserved for the VNFs to operate and by S^{h+v} the overall number of cores in the server.

We define a *service chain* φ to be an ordered set of VNFs $\varphi = \langle \varphi_1, \varphi_2, \ldots, \varphi_n \rangle$, and denote by $|\varphi| = n$ its length. We denote by $\varphi|^p$ (and $\varphi|^s$) the number of packets per second (and average packet size, respectively) that service chain φ is required to process. For a VNF $\varphi_i \in \varphi$ we denote by $\varphi_i|^c$ the CPU required for its operation. Throughout this paper, unless explicitly saying otherwise, we assume that we are given a set of k servers $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ and a sequence of m service chains $\Phi = \langle \varphi^1, \varphi^2, \ldots \varphi^m \rangle$[1].

[1]We always use subscript φ_i to refer to the i-th VNF in service chain φ, and upperscript φ^j to refer to the j-th service chain in Φ.

Given a service chain $\varphi = \langle \varphi_1, \varphi_2 \ldots, \varphi_n \rangle$ we define a *sub-chain* of VNFs to be a sub-sequence $\varphi_{s \twoheadrightarrow e} = \langle \varphi_s, \ldots, \varphi_e \rangle$ where $1 \leq s \leq e \leq n = |\varphi|$. We denote by $\Rightarrow$ the operator that *concatenates* VNFs. An ordered set of r sub-chains $\langle \varphi_{s_1 \twoheadrightarrow e_1}, \ldots, \varphi_{s_r \twoheadrightarrow e_r} \rangle$ is a *decomposition* of φ if the concatenation of all sub-chains recomposes φ, i.e., $\varphi = \langle \varphi_{s_1 \twoheadrightarrow e_1} \Rightarrow \cdots \Rightarrow \varphi_{s_r \twoheadrightarrow e_r} \rangle$. We say that a set of VNFs is an *affinity group* (and *anti-affinity group*) if all VNFs are mapped to the same server (and respectively mapped to different servers).

We define $\mathcal{P} : \Phi \rightarrow S^k$ to be a *placement function* that for every service chain $\varphi \in \Phi$, maps every VNF $\varphi_i \in \varphi$ to a server $S_j \in \mathcal{S}$. We denote two particularly interesting placement functions:

(i) $\mathcal{P}_g$ – called *gather* placement, where every service chain φ is an affinity group, namely:

$$\forall_{\varphi \in \Phi} \forall_{\varphi_i, \varphi_j \in \varphi} : \quad \mathcal{P}_g(\varphi_i) = \mathcal{P}_g(\varphi_j)$$

(ii) $\mathcal{P}_d$ – called *distribute* placement, where every service chain φ is an anti-affinity group, namely:

$$\forall_{\varphi \in \Phi} \forall_{\varphi_i, \varphi_j \in \varphi} : \quad i \neq j \rightarrow \mathcal{P}_d(\varphi_i) \neq \mathcal{P}_d(\varphi_j)$$

Figure 2 illustrates a few possible deployment strategies. Figure 2(a) depicts deployment of VNFs that follow gather placement function $\mathcal{P}_g$. Figure 2(b) depicts deployment of VNFs that follow distribute placement function $\mathcal{P}_d$. Figure 2(c) depicts deployment of VNFs that follow arbitrary mixed placement function $\mathcal{P}$. Intuitively, our goal is to develop a placement function $\mathcal{P}$ that decomposes each service chain into arbitrary decomposition of the service chain, while minimizing the operational cost of the network traffic switching. The decomposition groups together sub-chain into an affinity group, while exactly one (arbitrary) VNF from each sub-chain is in the same anti-affinity group.

For a given placement function $\mathcal{P}$ that deploys all service chains in $\Phi = \langle \varphi^1, \varphi^2, \ldots \varphi^m \rangle$ on the set of servers $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$, we define two cpu-cost functions:

(i) $\mathcal{C}^v : \mathcal{P} \rightarrow (\mathbb{R}^+)^k$ is the *guest cpu-cost function* that assigns per server the CPU required to operate the VNFs allocated on the server.

(ii) $\mathcal{C}^h : \mathcal{P} \rightarrow (\mathbb{R}^+)^k$ is the *hypervisor cpu-cost function* that assigns per server the CPU required to operate the network traffic switching on the server.

For simplicity, we use $\mathcal{C}^v_i$ (and $\mathcal{C}^h_i$) to refer to the i'th value associated with server S_i, and $\mathcal{C}^v$ (and $\mathcal{C}^h$) to denote the sum of all k servers, i.e., $\mathcal{C}^v = \sum_{i=1}^k \mathcal{C}^v_i$ (and respectively $\mathcal{C}^h = \sum_{i=1}^k \mathcal{C}^h_i$). We say that a placement function $\mathcal{P}$ is *feasible* with respect to the set of servers $\mathcal{S}$, if for every server, the sum of the hypervisor cpu-cost function and the guest cpu-cost function do not exceed the number of CPU cores that are available on this server, i.e.,

$$\forall_{S_j \in \mathcal{S}} : \mathcal{C}^h_j(\mathcal{P}) + \mathcal{C}^v_j(\mathcal{P}) \leq S^{h+v}_j \qquad (1)$$

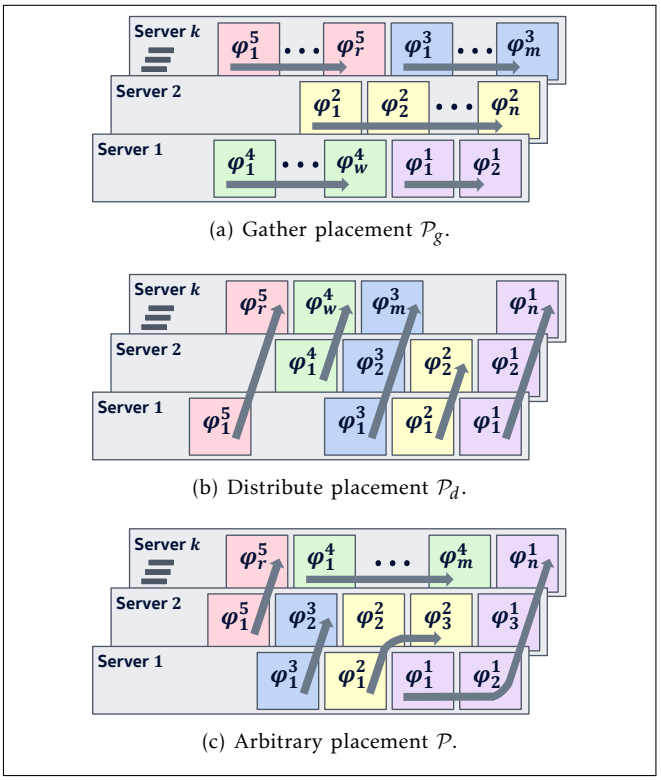


Fig. 2: Given a set of 5 service chains Φ and a set of k servers $\mathcal{S}$, illustrating different deployment strategies

Note that given a service chain φ, the distribute placement function requires to have at least $|\varphi|$ servers in order for the deployment to be feasible. On the other hand the gather placement function requires to have at least one server that can host all VNFs in φ in order for the deployment to be feasible. Thus, there could be situations where neither of these placements is feasible while a feasible placement exists.

We can now formally define the algorithmic problem of interest. We are given a set of k servers $\mathcal{S}$, and an online sequence of m service chains Φ. For each service chain in the sequence we need to find a feasible placement to all the VNFs in the chain such that the total cpu-cost increase is minimized. Finding a feasible placement for a given service chain is a zero-one acceptance problem since we are not allowed to place only part of the service chain, i.e., we either have to place the entire chain or nothing. Minimizing the total cpu-cost releases additional resources that can potentially be used to allocated additional network functions, and improves the acceptance rate of service chains in the sequence. Our evaluation of acceptance rate (see Section V) shows that the online cpu-cost minimization algorithm outperforms OpenStack/Nova scheduler, and brings us to a near optimal offline minimization.

III. The Operational Cost of Switching

In this section we define a model that captures the operational cost of virtual switching for a given server. Namely, given a placement function $\mathcal{P}$ that entails a set of sub-chains (originating from possibly different service chains) to deploy on server S_j, our goal is to develop a function that predicts the CPU cost of server S_j.

A. Virtual Switching

In virtualization-intense environments, virtual switching is an essential functionality that provides isolation, scalability, and mainly flexibility. However, the functionality provided by software switching also introduces a non-negligible operational cost making it much harder to guarantee a reasonable level of network performance, which is a key requirement for the success of the NFV paradigm. Assessing and understanding this operational cost and particularly the cost associated with virtual switching is a crucial step towards driving cost-efficient service deployments.

Several recent publications addressed the performance of intensive traffic applications in NFV chaining settings (e.g., [2], [16]–[18]). In most industry related work the goal is to define the setting that provides the best performance on a specific hardware, and not on the switching cost of a given service chain under a certain setting. As we mentioned, [16] is different as it does try to evaluate the switching cost but it does so only for the special cases of gather and distribute.

Yet the results of these recent studies indicate that the operational cost of deploying service chains depends on the installed OvS (either kernel OvS or DPDK-OvS), the required amount of traffic to process, the length of the service chain, and the placement strategy. Moreover, it appears that there is no single strategy that is always superior, with respect to the operational cost, and that the best strategy depends on the system parameters and the characterization of the deployed service chains.

B. The Cost of Virtual Switching

Let $\{\varphi^1_{s_1 \twoheadrightarrow e_1}, \ldots, \varphi^r_{s_r \twoheadrightarrow e_r}\}$ be a set of r sub-chains, each is part of a decomposition from possibly different service chain. Each sub-chain $\varphi^w_{s_w \twoheadrightarrow e_w}$ (for $1 \le w \le r$) might carry different traffic requirements, that are defined by service chain $\varphi^w \in \Phi$. Figure 3 illustrates such a deployment on server S_j. The total guest cpu-cost $\mathcal{C}^v_j(\mathcal{P})$ is just the sum of the required CPU for each VNF, but calculating the hypervisor switching cpu-cost $\mathcal{C}^h_j(\mathcal{P})$ is much more involved.

For a single sub-chain the switching cost is exactly the gather cost of a chain deployed on server S_j and can be obtained directly from function $\mathcal{F}$ defined in [16]. However, when we deploy more than one sub-chain, the overall cost is not the sum of the separate deployments. Figure 4 depicts this CPU switching cost as a function of

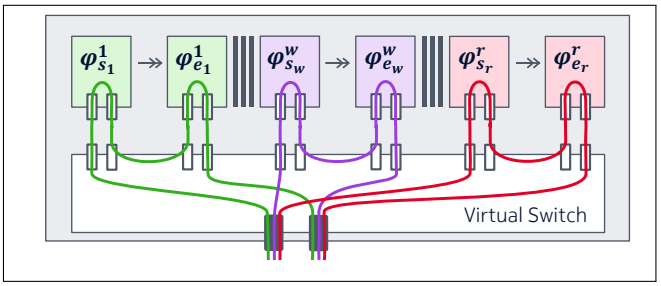


Fig. 3: Server S_j deployed with r sub-chains from possibly different service chains $\{\varphi^1_{s_1 \twoheadrightarrow e_1}, \ldots, \varphi^w_{s_w \twoheadrightarrow e_w}, \ldots, \varphi^r_{s_r \twoheadrightarrow e_r}\}$

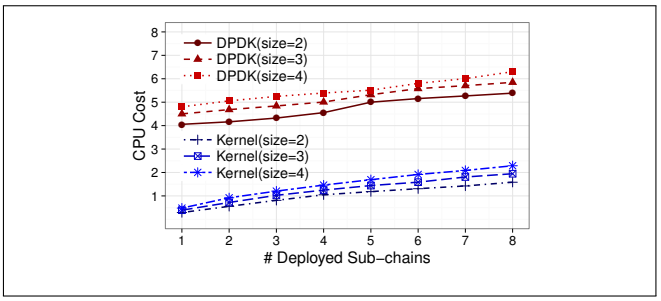


Fig. 4: The cpu-cost of concurrent deployment as a function of r, for several selected sub-chain sizes

the number of concurrent deployments (for various sub-chain lengths and different switching configurations). One can see that the cost of deploying two sub-chains is much smaller than twice the cost of deploying one sub-chain (this is true for each of the sub-chains shown in this figure). Thus in concurrent chain deployments the switching cost is amortized and the total cost is smaller than the sum of two separate costs.

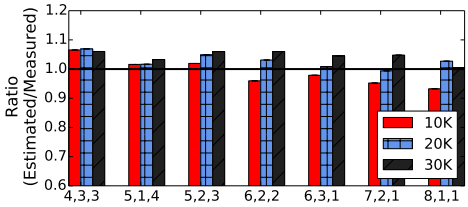
To quantify this we define $\|\varphi_{s \twoheadrightarrow e}\|^r$ to be the *concurrent deployment* of r copies of subchain $\varphi_{s \twoheadrightarrow e}$, and $\Delta(\varphi_{s \twoheadrightarrow e}, r)$ to be the *switching cost delta* between r times deploying a single sub-chain $\varphi_{s \twoheadrightarrow e}$ and the concurrent deployment of r copies of $\varphi_{s \twoheadrightarrow e}$, i.e.,

$$\Delta(\varphi_{s \twoheadrightarrow e}, r) = \left(r \cdot \mathcal{F}(\varphi_{s \twoheadrightarrow e}) - \mathcal{F}(\|\varphi_{s \twoheadrightarrow e}\|^r)\right) \tag{2}$$

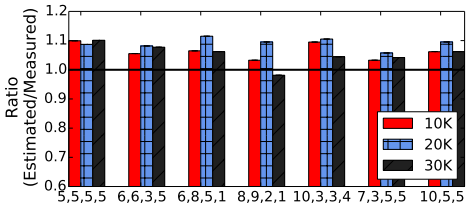
We performed extensive evaluations separately for environment installed with kernel OvS and DPDK-OvS and for different sub-chain lengths. Part of the results are shown in Figure 4. In order to be compliant with previous work, we used in this evaluation an environment setup that is similar to the one described in [16].

Given a set of r sub-chains $\{\varphi^1_{s_1 \twoheadrightarrow e_1}, \ldots, \varphi^r_{s_r \twoheadrightarrow e_r}\}$ that are deployed on server S_j (as illustrated in Figure 3), we can now compute the cpu-cost as follows:

$$\mathcal{C}^h_j = \sum_{w=1}^{r} \mathcal{F}(\varphi^w_{s_w \twoheadrightarrow e_w}) - (r-1) \cdot \left(\sum_{w=1}^{r} \Delta(\varphi^w_{s_w \twoheadrightarrow e_w}, r)\right) \Big/ r \tag{3}$$



(a) Decompositions of 10 VNFs.



(b) Decompositions of 20 VNFs.

Fig. 5: The ratio between the computed values of $\mathcal{C}_j^h$ compared to measurements of the corresponding executions

Where $\Delta(\varphi^w_{s_w \twoheadrightarrow e_w}, r)$ is computed as shown in Equation 2, with the traffic requirements defined for $\varphi^w_{s_w \twoheadrightarrow e_w}$. The left-hand side of Equation 3 is the sum of the cpu-cost associated with the gather deployment of each sub-chain, and the right-hand side reflects the saving due to the concurrent deployment of r chains which is $(r-1)$ times the average switching cost delta. Note that for $r = 1$ we do not have any savings and the cost is just the gather cost of deploying a single sub-chain.

C. Empirical Validation

To validate our formulation, we evaluate function $\mathcal{C}_j^h$ for various sub-chain deployment configurations. Figure 5 presents the ratio between the cpu-cost estimated by function $\mathcal{C}_j^h$ compared to measurements of the corresponding executions, for several traffic requirements. Figure 5(a) depicts the ratio for 10 VNFs, which are grouped into 3 different sub-chains, and Figure 5(b) depicts the ratio for 20 VNFs, which are grouped into 4 different sub-chains. As shown, our estimated cpu-cost is not very far from the measured executions (on average, less than 5%).

IV. Optimized Operational Cost Placement

In this section we introduce our placement algorithm for Operational (switching) Cost Minimization (OCM). Given a sequence of service chains Φ and a set of physical servers $\mathcal{S}$, our algorithm entails a strategy to deploy the service chains onto the set of physical servers. Per each service chain in the sequence, we find a partition of the chain into sub-chains and an allocation of a server to each sub-chain in a way that minimizes the total switching cost. This on-line handling does not guarantee global minimum switching cost but as we show in this section, it does reduce the switching CPU cost and allow deploying significantly more services.

A. Service Chain Partitioning

Given a service chain φ, our goal is to assign a server for each VNF $\varphi_i \in \varphi$, such that the overall placement is feasible and has the minimal network switching cost. OpenStack/Nova implements a naive approach which goes over all VNFs in the service chain, for each VNF filter the feasible servers, and then follows one of its global policies to assign a server. There are two problems with this approach. The first problem is that the placement is done per VNF and thus after placing several VNFs, it might realize that there is no feasible server for the next VNF and revoke the placement of the entire service chain. The second drawback is the complexity of this algorithm (even when succeeding to find a feasible allocation). When the number of servers is $K = |\mathcal{S}|$ and the maximum number of VNFs in service chain is bounded by N, the runtime complexity of this approach is $O(K^N)$. Since in practice the number of servers can reach hundreds or more, this approach easily becomes impractical.

To address this scalability issue we take a different approach, which iterates over all set partitions of the service chain into subsets of VNFs, relying on the fact that the size of service chains is practically bounded by a small constant. For each partition in the set, we use the switching cost model, described in the previous section, to determine the cost of $\mathcal{C}_j^h$ for each server. Then we find an optimal placement of this partition by finding the maximal matching between sub-chains and servers. To analyze the complexity of this approach, we need to bound the number of different partitions, which entails the required number of iterations.

The number of all possible partitioning of a set is known to be the *Bell number* of N (denoted by $B(N)$), and a recent study ([20]) established an upper bound of $(\frac{0.792N}{ln(N+1)})^N$ for it. Thus, in order to find an optimal placement, the *optimal OCM* algorithm needs to iterate over $O(N^N)$ different partitions of the chain. Note, however, that this complete enumeration of set partitioning also includes placements that allow the traffic to enter and leave a server more than once (e.g., partitioning service chain $\varphi = \langle \varphi_1, \varphi_2, \varphi_3 \rangle$ into subsets $\{\varphi_1, \varphi_3\}$ and $\{\varphi_2\}$). Since in practice this is not a reasonable placement, we consider a heuristically relaxed version of the problem that allows much fewer iteration (however, does not guarantee to find the optimum).

When assuming that in a reasonable deployment every service chain goes through a server at most once, finding all set partitioning (without repetitions) corresponds to enumerating all possible decompositions of φ. Thus we need to solve the combinatorial problem known as *integer composition*, which lists all possible ordered

lists of positive integers $n_1,\ldots,n_r$, such that their sum is equal to N (i.e., $N = \sum_{i=1}^{r} n_i$). Integer composition has been widely studied in the literature, including fast algorithms for its solution (e.g., [21], [22]), and the number of all possible integer compositions of N is exactly 2^{N-1}. Namely in order to find an optimal placement that goes through a server at most once, the *OCM* algorithm requires to iterate over $O(2^N)$ different partitions.

B. Using Matching to Find Optimal Cost Placement

For each partition, we want to find an optimal placement that maps each sub-chain in it to a server in $\mathcal{S}$. Note that we can also restrict the assignment such that each server is assigned with at most one sub-chain. This is true since assigning two consecutive sub-chains is equal to assigning a bigger sub-chain and this is covered by a different partition. We construct a bipartite graph where the nodes on one side are the sub-chains in the partition and the nodes on the other side are the servers in $\mathcal{S}$. The cost of an edge that connects sub-chain to a server is the extra $\mathcal{C}_j^h$ cost according to the switching cost model if an assignment of this sub-chain to this server is feasible, and ∞ otherwise (equivalent to removing this edge).

It is not too difficult to verify that a minimal cost placement of this set partitioning corresponds exactly to a minimum-weight perfect matching in the bipartite graph. Solving the minimum-weight perfect matching in the bipartite graph problem can be reduced to the problem of finding a minimum cost flow in a graph ([23]). The complexity is polynomial, and for a given service chain φ of length $N = |\varphi|$ and K servers, can be solved in time $O((N+K)\cdot N^2 \cdot K^2)$.

C. Operational Cost Minimization Algorithm

We are now ready to present our placement algorithm. The *Operational Cost Minimization* (OCM) algorithm receives as an input a set of servers $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ and a sequence of service chains $\Phi = \langle \varphi^1, \varphi^2, \ldots \varphi^m \rangle$. For each service chains $\varphi \in \Phi$, the OCM invokes the optimal service chain placement step shown in Algorithm 1 that is made of three building blocks: *(i)* list all set partitions of the VNFs in φ, or all decompositions in the relaxed version of the algorithm (Subsection IV-A); *(ii)* Given a set partition build the objective function, namely a cost function that predicts the operational cost of network traffic switching per each server (Subsection III-B); *(iii)* Given an objective function build a reduction to minimum-weight matching in bipartite graphs between partitions and servers, where the weights are given by the objective function (Subsection IV-B). We denote by $\mathcal{P}_o$ (and $\mathcal{P}_r$) the *optimal* placement function (and the *relaxed* placement function) that implements the optimal OCM algorithm (and respectively the relaxed version of the OCM algorithm).

Recall that in practice the number of servers K can reach hundreds or more, and since the runtime complexity of OpenStack/Nove approach is $O(K^N)$, it easily becomes impractical for chains that contain more than 2-3 VNFs. The OCM algorithm iterates over set partitioning, and for each set partition reduces the problem to minimum-weight matching in bipartite graphs, that can be solved in runtime of $O((N+K)\cdot N^2 \cdot K^2)$. The number of different set partitioning that the optimal OCM algorithm needs to iterate is bounded by $O(N^N)$, which entails a runtime complexity of $O(N^N \cdot (N+K)\cdot N^2 \cdot K^2)$ that can roughly scale to chains of size 5-6. On the other hand, the number of different decompositions that the relaxed version of the OCM algorithm needs to iterate is $O(2^N)$, which entails a runtime complexity of $O(2^N \cdot (N+K)\cdot N^2 \cdot K^2)$. Since typical chain length is roughly between 2 and 10 VNFs, the number of different decompositions is practically a constant. In Subsection V-A we evaluate the running times of all approaches, which reaffirms our formal analysis.

Algorithm 1 optimal service chain placement step

Input: $S \leftarrow \{S_1, S_2, \ldots, S_k\}$: set of servers
$\varphi \leftarrow \langle \varphi_1 \rightarrow \varphi_2 \ldots \rightarrow \varphi_n \rangle$: service chain
1: *min-cost* $\leftarrow \infty$: minimum cost found so far
2: *deploy-map* $\leftarrow$ NIL: maps all VNFs to servers in S
3: $\mathcal{A} \leftarrow$ all possible set partitioning (or decompositions)
4: **for** every set partition $a \in \mathcal{A}$ **do**
5: **for** every partition $p \in a$, and server $S_j \in \mathcal{S}$ **do**
6: $\mathcal{C}_j^h, \mathcal{C}_j^v \leftarrow$ compute the cost of deploying p on S_j
7: $G \leftarrow$ reduce to minimum cost flow in a graph
8: **if** $min(G) \leq$ *min-cost* **then**
9: *min-cost* $\leftarrow min(G)$
10: *deploy-map* $\leftarrow$ extract solution from G
11: **return** *deploy-map*

V. Evaluation

To assess the expected performance of the OCM algorithm as well as the quality of the results (i.e., the ability to increase utilization) compared to commonly used placement strategies, we implemented placements $\mathcal{P}_o$ and $\mathcal{P}_r$, and performed an extensive set of simulation based evaluations.

We consider a typical NFV-node (i.e., a small datacenter) that is composed of 100 servers. Each server is an high-end HP ProLiant DL380p Gen8 server with: two Intel Xeon E5-2697v2 processors, where each processor is made of 12 physical cores (24 cores in total) running at 2.7 Ghz; two NUMA nodes (Non-Uniform Memory Access), each has 192 GBytes RAM (total of 384 GBytes), and; an Intel 82599ES 10 Gbit/s network device with two network interfaces (physical ports). This is the type of servers that were used in the evaluation of the cost function $\mathcal{C}_j^h$. We measure the operational cost of switching using standard Open vSwitch (kernel OvS) as well as on DPDK accelerated Open vSwitch (DPDK-OvS).

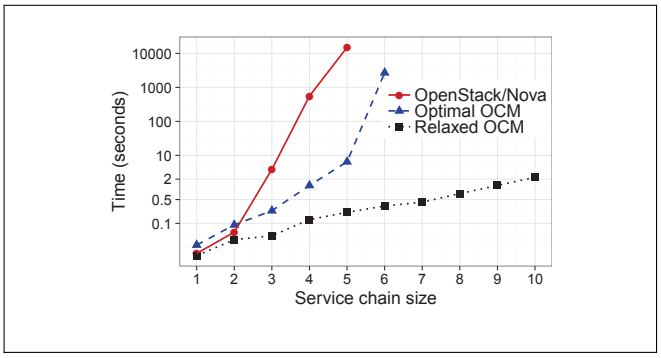


Fig. 6: Runtime analysis of different placements

For the purpose of this evaluation, we developed a generator of service chains requests (i.e., Φ). Service chain requests are handled one by one as described in our optimal service chain placement step (Subsection IV-C). The number of VNFs in a service chain is chosen uniformly at random in the range [2-10] (which is roughly the expected size of service chains in real deployments). Since our goal is to assess the operational cost of network switching, which is not directly affected by the amount of CPU needed by the VMs that implement the VNF, we assign to each VNF a single CPU. We vary the amount of packets processed in each service chain in the range of 10K to 1.5M packets per second. Unless otherwise specified, each experiment was performed 30 times, and considered 700 different service chain requests.

A. Evaluating the Runtime

Figure 6 depicts the runtime (in log scale) of placing a single service chains request, comparing our placements, $\mathcal{P}_o$ and $\mathcal{P}_r$ with OpenStack/Nova for different service chain sizes.

The evaluations reaffirm our formal analysis (see Subsection IV-C). Placement $\mathcal{P}_r$ results indeed show that the runtime aligns with the analyzed complexity which adds a factor of $O(2^N)$ that as expected can easily scale to service chains of length ~10 and more (executing for few seconds). The same analysis is also valid for placement $\mathcal{P}_o$ which adds a factor of $O(N^N)$ that as expected can scale to chains of length ~6 (up to a minute of execution). Finally, the results of OpenStack/Nova quickly surpasses 10K seconds for very few VNFs in the chain, which again aligns with our complexity analysis of $O(K^N)$ (i.e., the runtime is sensitive to the number of servers K).

When scaling our experiment to 1K servers, the relaxed OCM is still able to find solutions in a reasonable time (in the order of few seconds in the worst case). It is important to emphasize that we significantly improved the runtime while delivering quality-wise solutions as shown in the following.

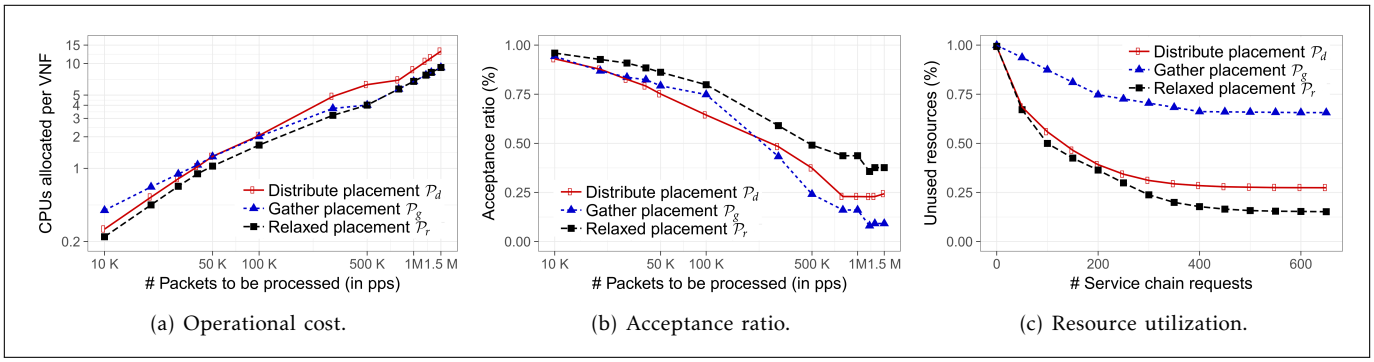
B. Evaluating the Quality of the Results

We compare the quality of the results of placement function $\mathcal{P}_r$ to the two commonly used strategies as discussed in Section II: *(i)* distribute strategy (load balancing policy in OpenStack/Nova, north/south in [2]) *(ii)* gather strategy (energy saving policy in OpenStack/Nova, east/west in [2]). We assume hard requirements (instead of soft constraints) and, therefore, requests are rejected whenever it is not possible to meet such requirements.

NFV operational cost. Figure 7(a) and Figure 8(a) depict the average additional CPU required to deploy a VNF, ranging over the traffic requirements of the service chain (in packet per seconds). As we observe, the additional number of cores needed for the virtual switching functionality is not negligible, and the cost tends to be higher as the traffic requirement increases. Note that for low network traffic requirements the operational cost of the distribute strategy is substantially higher in DPDK-OvS. This is since in DPDK-OvS there is a subset of poll-mode threads in the hypervisor, which consume resources regardless to the network traffic (doing busy-waiting). When comparing the amount of the additional resources required, placement function $\mathcal{P}_r$ requires less resources to achieve the same network performance guarantee. The improvement factor in the common case ranges between 20% and 40%, and in extreme cases can reach to as high as 400%.

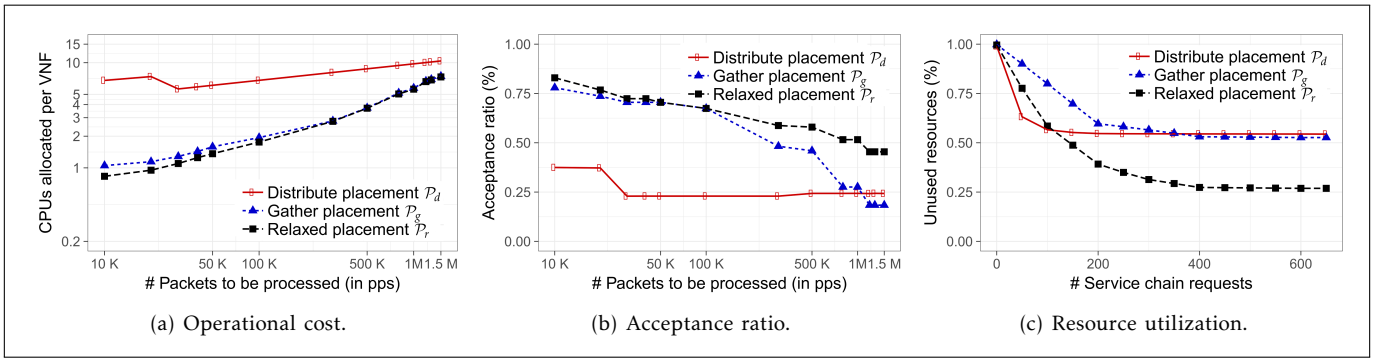
Service chain acceptance ratio. Figure 7(b) and Figure 8(b) depict the average acceptance ratio (namely, the ratio between the number of deployed chains and the number of the requested chains), ranging over different network traffic requirements. Placement function $\mathcal{P}_r$ is able to deploy a higher number of requests in most case, except for very few cases where its results are similar to the gather strategy $\mathcal{P}_g$. Since the distribute strategy $\mathcal{P}_d$ requires higher switching resources, the acceptance ratio is substantially affected in those cases (see Figure 8(b), where the acceptance is $\approx 25\%$). However, note that for higher network traffic requirements, the acceptance of the gather strategy degrades to be worst than the the distribute strategy. In turn, placement function $\mathcal{P}_r$ better takes advantage of the available resources, since it provides flexible solutions that better fit available physical resources and, therefore, ensures a higher acceptance rate. The improvement of placement function $\mathcal{P}_r$ over standard deployment policies is much more noticeable if we look at requests that carry high network traffic requirements.

Physical infrastructure resource utilization. Figure 7(c) and Figure 8(c) illustrate the average unused resource over time (i.e., the lower the average values the better). Observe that placement function $\mathcal{P}_r$ can utilize more resources in the physical infrastructure (up to 18% of unused resources) with comparatively



(a) Operational cost. (b) Acceptance ratio. (c) Resource utilization.

Fig. 7: Analysis of service chains deployment on NFV servers with kernel OvS



(a) Operational cost. (b) Acceptance ratio. (c) Resource utilization.

Fig. 8: Analysis of service chains deployment on NFV servers with DPDK-OvS

lower operational cost. The average of unused resources may be lower depending on the workload (for longer service chains, placement function $\mathcal{P}_r$ can better take advantage of chain decomposition). In DPDK-OvS, the average percentage of unused resources is slightly higher in comparison to kernel OvS. This is since DPDK-OvS requires a fixed amount of resources to operate (the threads required by the poll-mode drivers) which reflects high operational costs and less flexibility on placing service chains.

Overall one can see that reducing the switching CPU cost locally at each deployment of a new request, proves itself to be a very good strategy as it indeed improves the acceptance rate and allows a much better utilization of the resources.

VI. Related Work

Various studies considered the problems of virtual network function placement ([4], [5], [14]) and chaining ([3], [6]–[9], [13], [15]). However, none of these works take into account the cost of resources required to steer the traffic within a chain, which is non-negligible for deployment of packet intensive chains such as in the domain of NFV. Therefore, these solutions might either lead to infeasible solutions or suffer high penalties on the expected performance.

In [5] the authors addressed VNF placement, and used an optimization model along with approximation algorithms to solve the problem. They focus on where to deploy (VNFs) and how to assign traffic flows. Their work established a theoretical background for NFV placement, building on two classical optimization problems, namely facility location and generalized assignment. In [4] the authors focused on cost-oriented placement of elastic demands. They provide a dynamic mechanism to place, migrate and/or reassign network traffic mainly to cope with traffic fluctuations. However, their objective function do not take into account the internal switching cost of the server.

The authors of [3], [6], [15] and [7] studied joint optimization problems for placement and chaining of VNFs. In [3] they focused on formally specifying service chains and on analyzing their ILP model under different objective functions. In turn, [6] and [15] provide a general ILP model which takes into account end-to-end delay and resources constraints. The work in [7] provides a similar model that focus on reducing the general operational expenditures of datacenters over time (e.g., energy consumption of deployed services). [10] explored the relation between resource consumption on physical servers and links. [11] and [12] were the first to introduce approximation algorithms to the joint NFV optimization problem. The work in [11] presented the first polynomial

time service chain approximation for request admission control. Their solution is based on classical rounding techniques. Finally, [12] provides a deterministic approximation algorithm based on submodular functions covering incremental deployment.

Most of these work focused on arbitrary cost functions (e.g., reducing the amount of deployed VNFs). However, the real operational cost depends on many factors including the server settings and the way VNFs are deployed on physical servers (intra- and inter-server). Several recent studies (mainly industry-driven) address this issue of "configuration for optimal performance" [2], [16]–[18], but in these works this is done by a careful manual process. For the best of out knowledge this is the first paper that addresses this gap and presents an automated algorithmic approach for this important problem.

VII. Conclusion and Future Work

In this work we introduced the Operational Cost Minimization (OCM) placement algorithm, a performance-oriented deployment mechanism that minimizes internal switching CPU overhead and improves network utilization. This algorithm uses a novel cost model that captures the operational cost of the internal virtual switching for a given server. We provided empirical evidence, using a real NFV-based environment, indicating that our cost model is accurate comparing to actual deployment measurements (lower than 5%). Using this cost model, we introduced an efficient online placement algorithm that minimizes the switching cost of service chain requests. We show that OCM significantly reduces the operational costs and increases utilization, when compared to commonly used deployment strategies (up to a factor of 4 in the extreme case and 20% – 40% in typical cases).

We plan to integrate OCM in OpenStack/Nova and measure its performance in real NFV-based deployments. We also intend to explore extensions of our algorithm/model to support non-linear service chains and to cope with others network requirements (e.g., network monitoring). Finally, our work opens opportunities to design new pricing models for service chaining by NFV providers.

References

[1] "Intel dpdk," http://dpdk.org/, accessed: 07-20-2017.

[2] P. Kutch and B. Johnson, "Sr-iov for nfv solutions practical considerations and thoughts," http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/sr-iov-nfv-tech-brief.pdf, Intel Networking Division (ND), Feb 2017.

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

[4] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, Oct 2015, pp. 255–260.

[5] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 1346–1354.

[6] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 98–106.

[7] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)*, ser. CNSM '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 50–56.

[8] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 89–97.

[9] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of vdpi functions in nfv infrastructures," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, April 2015, pp. 1–9.

[10] T. Kuo, B. Liou, J. Lin, and M. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *IEEE International Conference on Computer Communications (INFOCOM 2016)*, San Francisco, USA, April 2016.

[11] M. Rost and S. Schmid, "Service chain and virtual network embeddings: Approximations using randomized rounding," *CoRR*, vol. abs/1604.02180, 2016. [Online]. Available: http://arxiv.org/abs/1604.02180

[12] T. Lukovszki, M. Rost, and S. Schmid, "It's a match!: Near-optimal and incremental middlebox deployment," *SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, pp. 30–36, Jan. 2016.

[13] J. J. Kuo, S. Shen, H. Kang, D. Yang, M. Tsai, and W. Chen, "Service chain embedding with maximum flow in software defined network and application to the next generation cellular network architecture," in *IEEE International Conference on Computer Communications (INFOCOM 2017)*, Atlanta, USA, April 2017.

[14] M. Wenrui, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent nfv middleboxes," in *IEEE International Conference on Computer Communications (INFOCOM 2017)*, Atlanta, USA, April 2017.

[15] M. C. Luizelli, W. L. da Costa Cordeiro, L. S. Buriol, and L. P. Gaspary, "A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining," *Computer Communications*, vol. 102, pp. 67 – 77, 2017.

[16] M. C. Luizelli, D. Raz, Y. Sa'ar, and J. Yallouz, "The actual cost of software switching for nfv chaining," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 335–343.

[17] Intel, "Intel open network platform release 2.1: Performance test report," Internet Engineering Task Force, Mar. 2016, Available: https://01.org/packet-processing/intel®-onp.

[18] B. O. Maryam Tahhan and A. Morton, "Benchmarking virtual switches in opnfv," https://tools.ietf.org/html/draft-ietf-bmwg-vswitch-opnfv-04, Work in progress, accessed: 07-20-2017.

[19] "Openstack," http://www.openstack.org/, accessed: 07-20-2017.

[20] D. Berend and T. Tassa, "Improved bounds on bell numbers and on moments of sums of random variables," *Probability and Mathematical Statistics*, vol. 30, no. 2, pp. 185–205, 2010.

[21] M. Merca, "Fast algorithm for generating ascending compositions," *Journal of Mathematical Modeling and Algorithms*, vol. 11, no. 1, pp. 89–104, 2012.

[22] J. Kelleher and B. O'Sullivan, "Generating all partitions: A comparison of two encodings," *CoRR*, vol. abs/0909.2331, 2009. [Online]. Available: http://arxiv.org/abs/0909.2331

[23] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.