Online VNF Scaling in Datacenters

Xiaoke Wang*, Chuan Wu*, Franck Le†, Alex Liu‡, Zongpeng Li§ and Francis Lau*
*The University of Hong Kong, Email: {xkwang, cwu, fcmlau}@cs.hku.hk
†IBM T. J. Watson, Email: fle@us.ibm.com
‡Michigan State University, Email: alexliu@cse.msu.edu
§University of Calgary, Email: zongpeng@ucalgary.ca

Abstract—**Network Function Virtualization (NFV) is a promising technology that promises to significantly reduce the operational costs of network services by deploying virtualized network functions (VNFs) to commodity servers in place of dedicated hardware middleboxes. The VNFs are typically running on virtual machine instances in a cloud infrastructure, where the virtualization technology enables dynamic provisioning of VNF instances, to process the fluctuating traffic that needs to go through the network functions in a network service. In this paper, we target dynamic provisioning of enterprise network services - expressed as one or multiple service chains - in cloud datacenters, and design efficient online algorithms without requiring any information on future traffic rates. The key is to decide the number of instances of each VNF type to provision at each time, taking into consideration the server resource capacities and traffic rates between adjacent VNFs in a service chain. In the case of a single service chain, we discover an elegant structure of the problem and design an efficient randomized algorithm achieving a e/(e-1) competitive ratio. For multiple concurrent service chains, an online heuristic algorithm is proposed, which is O(1)-competitive. We demonstrate the effectiveness of our algorithms using solid theoretical analysis and trace-driven simulations.**

I. Introduction

The newly emerged paradigm of Network Function Virtualization (NFV) aims to consolidate network functions onto industry-standard high-volume servers, switches and storage using standard IT virtualization technology, in order to enable rapid network service composition/innovation, energy reduction and cost minimization for network operators [1]. A network function deployed in the NFV environment is termed virtualized network function (VNF) [2].

Virtualization of enterprise network functions has been among the initially proposed important use cases of NFV. Many enterprises require a significant number of network functions (comparable to the number of access routers), which typically constitute one or multiple network function service chains, to support their services [3]. For example, a common service chain for access service may contain an intrusion detection device, a firewall, and a load balancer that distributes incoming traffic to a pool of servers.

A common NFV proposal is to implement an enterprise's network service chains using VNF instances running on virtual machines (VMs) in a cloud computing platform, *i.e.*, the enterprise's datacenter or a public cloud, where the VNFs are made available to the enterprise in the fashion of Software as a Service [4]. No matter whether the datacenter is owned by the enterprise or operated by a 3rd-party cloud provider, cost minimization is among the top priorities. The main provisioning cost of VNF instances is due to power consumption to operate servers and cooling facilities, which is largely decided by the numbers and the types of VMs running different VNFs. On the other hand, launching new VNF instances on servers involves transferral of VM images, booting and attaching them to devices. This leads to a deployment cost, which is typically considered on the order of the cost to run a server for a number of seconds or minutes [5]. Such a deployment cost should be minimized as well, by avoiding frequent deployment and removal of VNF instances.

This paper addresses the following key problem in dynamic VNF provisioning in a cloud datacenter:

Given one or multiple service chains that constitute an enterprise's network service(s), how can we design online solutions that dynamically scale VNF instances and pack them onto servers, to adequately serve fluctuating input traffic, and to achieve close-to-offline-minimum provisioning cost over the long run of the system?

The key technical challenge in designing an efficient online algorithm to fulfil the above goal lies in the unknown nature of future traffic rates arriving at each service chain, which can vary significantly over time. We have to strategically make the deployment decisions on the go, avoiding undesirable situations of launching a VNF instance immediately following destroying the previous one.

Our contributions in efficient online VNF provisioning algorithm design are summarized as follows:

First, in the case of a single service chain, we present a randomized online algorithm which returns a solution of total cost at most $e/(e-1)$ times the offline optimal solution. The algorithm proceeds in two stages. The first stage is called "preplanning" which prevents VNF migration across servers over time. The second stage is an adaptation of the classic ski-rental algorithm which leads to an optimal competitive ratio.

Next, in the case of multiple concurrent service chains, we show the algorithm used in the single service chain cannot

The project was supported in part by grants from Hong Kong RGC under the contracts HKU 718513, 17204715, and C7036-15G (CRF). Franck's research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defense and was accomplished under Agreement Number W911NF- 06-3-0001. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defense or the U.K.

be directly applied to this scenario. Instead, we present a heuristic algorithm which relies on a minimal weight matching algorithm to minimize the deployment cost and achieves a competitive ratio of O(1).

Furthermore, our algorithms are simple and easy to implement in practice in an online manner. We demonstrate the effectiveness of our algorithms using both solid theoretical analysis as well as extensive trace-driven evaluation.

II. Related Work

A. Enabling Technologies for NFV

There have been recent work on building efficient software middleboxes approaching performance of hardware middleboxes, *e.g.*, ClickOS [6] and CoMb [3]. ClickOS is able to saturate a 10Gb pipe on a commodity server and CoMb can reduce the network provisioning cost by 60%.

Some other work focus on management of VNF instances. SIMPLE [7] implements an SDN-based policy enforcement layer for efficient middlebox-specific traffic steering. APLOMB [8] facilitates outsourcing of enterprise middleboxes to the cloud. Clayman *et al.* [9] design an orchestrator-based architecture for automatic placement of virtual nodes and allocation of network services. These systems greatly facilitate the deployment of VNF instances.

To support VNF scaling, Split/Merge [10] enables efficient, load-balanced elasticity for scaling virtual middleboxes. OpenNF [11] supports loss-free and order-preserving flow state migration by leveraging events triggered by VNFs, buffering the relevant packets, and enabling a two-phase forwarding state update. We assume exploiting such a system for implementing scaling instructed by our online algorithms.

B. Optimal Placement of VNFs

There already have some work about the placement of VMs before the emergence of NFV. For example, Meng *et al.* [12] investigate how to optimize the placement of VMs on host machines considering the traffic patterns among VMs. Also, frameworks like OpenStack [13] have schedulers which could consolidate VMs for physical memory and power consumption savings. Such work does not take network service chaining into consideration, which has to be taken into account when we place VNFs. To optimize orchestration and placement of VNFs, Stratos [14] orchestrates the VNFs to optimize resource usage in three steps. VNF-P [15] presents an optimization model for VNF resource allocation and designs a fast heuristic algorithm. Bari *et al.* [16] study a similar problem, formulate the problem into an integer linear program (ILP) and solve it by a standard solver. Cohen *et al.* [17] investigate a NFV placement problem to optimize the distance cost between clients and the virtual functions that they need. Except [17], the other studies propose heuristic algorithms to solve the respective VNF placement problems, without providing any theoretical performance guarantee. In addition, all of these work focus on offline/one-time VNF placement and resource allocation, instead of dynamic VNF provisioning, which calls for efficient online algorithm design.

III. Problem Model

A. System Model

We consider an enterprise deploying S service chains in a cloud datacenter. Each service chain s is an ordered sequence of VNFs. The set of links interconnecting VNFs in s is denoted by $\mathbb{L}^s$, where $(i, j) \in \mathbb{L}^s$ if VNF i is the predecessor of VNF j in service chain s. For ease of problem formulation, special VNF types named VNF 0 and $0'$ are added to the head and the tail of a service chain to indicate source and destination of the traffic flow, respectively, and the corresponding links $(0, i)$ and $(j, 0')$ are added to $\mathbb{L}^s$ as well, assuming i and j are the first and the last VNF in s. The system works in a time slotted fashion, spanning time slots $1, 2, \ldots, T$. The input traffic rate to each service chain s varies from time to time, denoted by $\alpha^s(t)$ (*e.g.*, in Mbps) at time t.

There are in total I different types of VNFs in the system. Multiple VM instances can be provisioned in the cloud datacenter to run the same type of VNF, which we refer to as *instances of the VNF*. These VMs have specific configurations and require fixed resources. An instance of VNF i consumes c_{ir} of resource $r \in R$, where R is the number of resource types including CPU, memory, storage and network bandwidth. Based on its resource composition, an instance of VNF i can maximally process traffic at the rate of b_i (in Mbps) at each time, without incurring prolonged packet queueing delays that violate a preset performance threshold.

There are U servers in the cloud datacenter. Without loss of generality, we assume that the servers are homogeneous, each having a capacity C_r of resource $r \in R$.

B. Problem Formulation

Our dynamic VNF provisioning problem is to find the optimal numbers and server placement of VNF instances for the service chains in each time slot, in order to minimize the provisioning cost over the entire system span T. We define the following decision variables: (i) VNF placement variable $x_{ui}(t)$, which denotes the number of instances of VNF i running on server u in t, $\forall u \in U, i \in I, t \in T$; (ii) routing variable $y^s_{uivj}(t)$, which represents the amount of traffic (in Mbps) in service chain s, forwarded from instances of VNF i on server u to instances of VNF j on server v, $\forall u, v \in U, i, j \in I, s \in S, t \in T$.

The VNF provisioning cost contains two parts:

I. VNF operational cost. Let ϕ_i denote the cost of running an instance of VNF i per time slot, mainly attributed to the power consumption of the hosting server. The overall operational cost is

$$\mathbb{O} = \sum_{t \in [T]} \sum_{u \in [U]} \sum_{i \in [I]} \phi_i x_{ui}(t) \tag{1}$$

II. VNF deployment cost. Deploying a new instance of VNF i requires transferring a VM image containing the network function, booting it and attaching it to devices on the server. We associate a deployment cost φ_i with the process. The overall VNF deployment cost can be expressed as

$$\mathbb{D} = \sum_{t\in[T]} \sum_{u\in[U]} \sum_{i\in[I]} \varphi_i [x_{ui}(t) - x_{ui}(t-1)]^+ \qquad (2)$$

where$[x_{ui}(t) - x_{ui}(t-1)]^+ = \max\{x_{ui}(t) - x_{ui}(t-1), 0\}$

indicates the number of newly added instances of VNF i on server u in t.

Our objective is to minimize the total cost $\mathbb{O} + \mathbb{D}$. The constraints that the decision variables should respect include the following.

First, all incoming traffic to each service chain at each time should be served. This can be guaranteed by the following constraints.

(i) Taking in all the incoming traffic to instances of the first VNF in a service chain (possibly deployed in different servers):

$$\sum_{v\in[U]} \sum_{j:(0,j)\in\mathbb{L}^s} y^s_{00vj}(t) \geq \alpha^s(t), \forall s \in [S], t \in [T] \qquad (3)$$

where $y^s_{00vj}(t)$ denotes the incoming traffic rate (from the dummy VNF 0 on an imaginary server 0) directed to instances of VNF j on server v, which is summed up in the left-hand side of (3) only if VNF j is the first VNF in the chain, *i.e.*, $(0, j) \in \mathbb{L}^s$.

(ii) Flow conservation at instances of each VNF i in service chain s on each server u:

$$\sum_{v\in[U]} \sum_{(i,j)\in\mathbb{L}^s} y^s_{uivj}(t) = \lambda^s_i \sum_{v\in[U]} \sum_{(j,i)\in\mathbb{L}^s} y^s_{vjui}(t), \qquad (4)$$
$$\forall s \in [S], u \in [U], i \in [I], t \in [T]$$

where the left-hand-side is the overall outgoing traffic from instances of VNF i on server u to instances of next-hop VNF j of service chain s on different servers, and the summation in the right-hand side computes the overall incoming traffic to instances of VNF i on u from instances of the previous-hop VNF deployed on different servers. It is worth noting that the flow rate of traffic may change after being processed by a network function [14]. Hence we define a gain/drop factor λ^s_i for each VNF i in each service chain s.

(iii) Provisioning a sufficient number of instances of each VNF on each server, whose overall processing capacity can serve all the received flows (possibly) from different previous-hop VNFs in different service chains:

$$\sum_{s\in[S]} \sum_{v\in[U]} \sum_{(j,i)\in\mathbb{L}^s} y^s_{vjui}(t) \leq x_{ui}(t) b_i, \forall i \in [I], u \in [U], t \in [T] \qquad (5)$$

In addition, resource capacities on each server should not be over-committed by the deployed VNF instances at any time, as expressed by the following constraints:

$$\sum_{i\in[I]} x_{ui}(t) c_{ir} \leq C_r, \forall u \in [U], r \in [R], t \in [T] \qquad (6)$$

In summary, the offline VNF provisioning problem is formulated as follows:

$$\min \sum_{t\in[T]} \sum_{u\in[U]} \sum_{i\in[I]} (\phi_i x_{ui}(t) + \varphi_i [x_{ui}(t) - x_{ui}(t-1)]^+) \qquad (7)$$

s.t. constraints (3)(4)(5)(6)

$$x_{ui}(t) \in \{0, 1, ...\}, \forall u \in [U], i \in [I], t \in [T] \qquad (8)$$

$[X]$	integer set {1,2,...,X}
U	# of servers in the system
T	# of time slots
I	# of VNF types
S	# of service chains
R	# of resource types in a server
$\mathbb{L}^s$	the set of links between VNFs in service chain s
$x_{ui}(t)$	# of VNF i on server u in time slot t
$y^s_{uivj}(t)$	traffic from VNF i on server u to VNF j on server v in t
$\alpha^s(t)$	the input traffic to service chain s at time t
λ^s_i	gain/drop factor of VNF i in service chain s
c_{ir}	VNF i's consumption of resource r
C_r	capacity of resource r on a server
b_i	processing capacity of an instance VNF i
ϕ_i	per-time-slot operational cost of an instance of VNF i
φ_i	deployment cost of an instance of VNF i

TABLE I
NOTATION

$$y^s_{uivj}(t) \geq 0, \forall s \in [S], u, v \in [U], i, j \in [I], t \in [T] \quad (9)$$

We list important notation in this paper in Table I, for ease of reference.

From the model, we can see that as the input traffic rate to the service chains varies, the minimal number of VNF instances required also changes. This necessitates new routing solutions and new placement solutions. Also, due to the existence of deployment cost, optimizing operational cost in each time slot will lead to a suboptimal solution.

C. Simplifying the Offline Optimization Problem

The offline optimization problem in (7) gives the optimal VNF deployment and traffic routing decisions at each time, using complete information of traffic rates during the entire system span. Towards designing an efficient online algorithm, we study the structure of the offline problem carefully, and identify that the problem formulation can actually be simplified by removing routing decisions $y^s_{uivj}(t)$'s, leaving only VNF placement decisions $x_{ui}(t)$'s. The key observation is that given specific input traffic rates to the service chains at t, $\alpha^s(t), \forall s \in [S]$, the minimal number of instances of each VNF required in the entire system at the time can be obtained based on constraints (3), (4) and (5), as stated in the following theorem.

Theorem 1. *The minimal number of instances of VNF i required to serve input traffic rates $\alpha^s(t), \forall s \in [S]$, in time slot t is $n_i(t) = \lceil \frac{\sum_{s\in[S]} \bar{\lambda}^s_i \alpha^s(t)}{b_i} \rceil$, where $\bar{\lambda}^s_i = \prod_{j=1}^{N^s(j+1)=i} \lambda^s_{N^s(j)}$ is the cumulative gain/drop factor in service chain s before the flow enters VNF i in the chain and $N^s(j)$ denotes the type of j-th VNF in the service chain s.*

The detailed proof is given in our technical report [18].

Then we can convert the offline problem (7) into the following one:

$$\min \sum_{t\in[T]} \sum_{u\in[U]} \sum_{i\in[I]} (\phi_i x_{ui}(t) + \varphi_i [x_{ui}(t) - x_{ui}(t-1)]^+) \qquad (10)$$

s.t.

$$\sum_{u\in[U]} x_{ui}(t) \ge n_i(t), \forall i \in [I], t \in [T] \tag{11}$$

$$\sum_{i\in[I]} x_{ui}(t) c_{ir} \le C_r, \forall u \in [U], r \in [R], t \in [T] \tag{12}$$

$$x_{ui}(t) \in \{0, 1, ...\}, \forall u \in [U], i \in [I], t \in [T] \tag{13}$$

In fact, we are able to prove the following theorem:

Theorem 2. *The offline VNF provisioning problem in (10) is equivalent to the offline problem in (7).*

The detailed proof is given in our technical report [18].

In what follows, we design online algorithms to make VNF placement and traffic routing decisions on the go, based only on current information and past history. We divide our design into two cases, for deployment of a single service chain and concurrent deployment of multiple service chains.

IV. Online Algorithm for a Single Service Chain

In this section, we focus on a single service chain containing VNFs $1 \to 2 \to \cdots \to L$. We remove s (that indicates a service chain) from all relevant notation, for simplicity.

A. Insights from the Offline Problem

The offline VNF provisioning problem in a system with only one service chain takes the same form as problem (10), except that the minimal number of instances of each VNF i in constraint (11) is computed as $n_i(t) = \lceil \frac{\bar{\lambda}_i \alpha(t)}{b_i} \rceil$.

We observe that the objective function is equivalent to $\sum_{t\in[T]} \sum_{i\in[I]} (\sum_{u\in[U]} \phi_i x_{ui}(t) + \sum_{u\in[U]} \varphi_i [x_{ui}(t) - x_{ui}(t-1)]^+)$. In the second term, we have $\sum_{u\in[U]} [x_{ui}(t) - x_{ui}(t-1)]^+ \ge [\sum_{u\in[U]} x_{ui}(t) - \sum_{u\in[U]} x_{ui}(t-1)]^+$. The equality holds if and only if the signs of $x_{ui}(t) - x_{ui}(t-1), \forall u \in [U]$, are the same: from $t-1$ to t, the numbers of instances of VNF i deployed on all the servers either all increase ($x_{ui}(t) \ge x_{ui}(t-1), \forall u \in [U]$) or all decrease ($x_{ui}(t) \le x_{ui}(t-1), \forall u \in [U]$); there does not exist a pair of servers u_1 and u_2, such that $x_{u_1 i}(t) > x_{u_1 i}(t-1)$ and $x_{u_2 i}(t) < x_{u_2 i}(t-1)$. The later can be considered as a *VNF instance migration* case, *i.e.*, at least one instance of VNF i is moved from one server to another.

The following is an example where VNF instance migration has to be done due to limited capacity. Suppose there are 2 servers, each of which has 1 unit resource. The service chain is VNF $1 \to 2$. An instance of VNF 1 requires 0.5 unit of resource and an instance of VNF 2 requires 0.2 unit. At first, there is 1 instance of VNF 1 and 1 instance of VNF 2 on each server. Then, due to the increase of input traffic rate, we have to add 1 instance for each VNF. However, none of the two servers has enough resource to accommodate an additional instance of VNF 1. We have to migrate one instance of VNF 2 from one server to the other to make room for the additional instance of VNF 1.

Nevertheless, if we can somehow ensure that no migration is needed (we will design an algorithm for it), the objective function is equivalent to $\sum_{t\in[T]} \sum_{i\in[I]} (\sum_{u\in[U]} \phi_i x_{ui}(t) + \varphi_i [\sum_{u\in[U]} x_{ui}(t) - \sum_{u\in[U]} x_{ui}(t-1)]^+)$. Furthermore, if the server capacity constraint (12) can be ignored temporarily, then we can create a new decision variable $x_i(t) = \sum_{u\in[U]} x_{ui}(t)$, denoting the total number of instances of VNF i deployed in the system in t, and convert the offline problem (10) into the following:

$$\min \sum_{t\in[T]} \sum_{i\in[I]} (\phi_i x_i(t) + \varphi_i [x_i(t) - x_i(t-1)]^+) \tag{14}$$

$$\text{s.t.} \quad x_i(t) \ge n_i(t), \forall i \in [I], t \in [T] \tag{15}$$

$$x_i(t) \in \{0, 1, ...\}, \forall i \in [I], t \in [T] \tag{16}$$

The optimization problem (14) deals with the total number of instances to deploy for each VNF type to fulfil the traffic demand at each time, without taking care of the detailed placement of the VNF instances on servers, *i.e.*, how many of $x_i(t)$ to place on a server u. In addition, there is no coupling of decisions among different types of VNFs, such that we can separately optimize the total number of instances to deploy for each VNF, by solving the subproblem:

$$\min \sum_{t\in[T]} (\phi_i x_i(t) + \varphi_i [x_i(t) - x_i(t-1)]^+) \tag{17}$$

$$\text{s.t.} \quad x_i(t) \ge n_i(t), x_i(t) \in \{0, 1, ...\}, \forall t \in [T]$$

We observe that the offline optimization problem (17) can be solved by doing some minor modifications to classic ski-rental algorithms [19] [20]. It can be verified that if in the next $\lfloor \varphi_i / \phi_i \rfloor$ time slots, the number of VNF i instances required is smaller or equal to current number of VNF i instances, then the most economical way is to remove the extra VNF i instances and only keep the maximum number of VNF i instances required by the next $\lfloor \varphi_i / \phi_i \rfloor$ time slots. This is exactly the idea behind the offline ski-rental algorithms. Therefore, to design an online algorithm for problem (17), we can apply the classic ski-rental algorithms, such as that in [19]. We distinguish idle VNF instances from running VNF instances. For idle instances, they would not be removed from servers until the accumulated operational cost is larger than their deployment cost. If an idle instance is required during that time duration, we simply turn its state into running without causing any deployment cost. The detailed algorithm is to be discussed in Alg. 2.

The pending issue for designing an online algorithm that solves (10), is to come up with an efficient VNF placement scheme, which places VNF instances at the total number of $x_i(t)$ from (14) on the servers, ensuring that server capacity constraint in (12) is respected, and no VNF instance migration would happen, *i.e.*, from $t-1$ to t, the servers are all hosting more instances of VNF i or are all hosting less instances of VNF i. In the next subsection, we propose an efficient pre-planning step, which produces a feasible VNF placement solution, assuming that the input traffic rate to the service chain is the largest possible that the system can support. We then design our online algorithm to solve (10), combining $x_i(t)$'s computed from (14) and the VNF placement scheme derived from the pre-planning step, which satisfies the above conditions.

B. VNF Deployment for Maximum Traffic Rate

We now investigate the maximum input traffic rate α^{max} that the system can support in a single time slot, as well as obtain a feasible VNF deployment solution to serve a flow at this maximum rate. Given an input traffic rate α, we can obtain the corresponding minimum number of instances of each VNF needed, n_i, according to Theorem 1, for all i in the service chain. Then whether U servers are enough to host these VNFs can be determined by finding a feasibility solution following optimization problem, where binary variable z_u denotes whether server u is used or not:

$$\sum_{u \in [U]} x_{ui} \geq n_i, \forall i \in [L] \tag{18}$$

$$\sum_{i \in [l]} x_{ui} c_{ir} \leq C_r z_u, \forall u \in [U], r \in [R] \tag{19}$$

$$x_{ui} \in \{0, 1, ...\}, z_{ui} \in \{0, 1\}, \forall u \in [U], i \in [L] \tag{20}$$

To decide the maximum supportable input traffic rate, we can apply a bi-section algorithm. The complete algorithm is given in Alg. 1.

Algorithm 1: Pre-planning Step

```
Input: $U, L, R, \lambda, \mathbf{b}, \mathbf{c}, \mathbf{C}$
Output: $\alpha^{max}$, $\mathbf{x}^{max}$

$l_b = 0, u_b = MAXRATE$;
while $u_b - l_b > 1$ do
    $m := (u_b + l_b)/2$;
    Solve problem (18) by setting $n_i = \lceil \frac{\bar{\lambda}_i m}{b_i} \rceil$, using
    bin-packing algorithm in [21]
    if Exists a feasible solution $\mathbf{x}$ to problem (18) then
        $l_b := m$
    else
        $u_b := m$
$\alpha^{max} := l_b$, $\mathbf{x}^{max} := \mathbf{x}$
```

In Alg. 1, we set the initial search range for α^{max} to $[0, MAXRATE]$, where $MAXRATE$ can be a very large estimated rate. To solve problem (18) at specific n_i's, we note that the problem is the high multiplicity variant of a multi-dimensional bin packing problem [22], which is NP hard if the length L of the service chain is a large number. However, the length of a service chain is usually short - the length of most of the representative NFV service chains is no larger than 4. Hence we take L as a fixed small constant. In this case, we show that there exists a polynomial-time algorithm to solve the problem. In particular, Goemans and Rothvoss [21] show that the 1-dimensional bin packing problem with a fixed number of item types (L in our model) and high multiplicity can be solved in polynomial time. We prove that the same algorithm from [21] can be applied to solve bin packing problems of any fixed dimensions, *i.e.*, the number of resource types R in our model, in polynomial time as Theorem 3 shows.

Theorem 3. *Alg. 1 runs in polynomial time.*

The detailed proof is given in our technical report [18].

C. VNF Placement Scheme

Combining the pre-planning step in Alg. 1 with our insights in Sec. IV-A, we are able to design an efficient VNF placement scheme which can guarantee server capacity constraint (12) is respected and no migration of VNF instances occurs. The main idea is to guarantee the set of server placement of VNF instances to serve the input traffic rate $\alpha(t)$ for any t is always a subset of the set of server placement of VNF instances to serve the maximum input traffic rate α^{max}.[1] For example, suppose we have 3 instances of VNF 1 on server 1 and 1 instance of VNF 1 on server 2 to serve α^{max}; then to serve $\alpha(t)$, we could deploy 2 instances of VNF 1 on server 1 but never deploy 2 instances of VNF 1 on server 2. Such a placement scheme always respects server capacity constraint (12) since the VNF placement to serve α^{max} is governed by the constraint already.

To accomplish the above, all we need is a multiset, S_i for VNF i, which includes the IDs of servers where all the instances of VNF i are placed according to the solution to serve the maximum traffic rate. For example, if we need 2 instances of VNF i on server 1, 3 instances of VNF i on server 2 and 1 instance of VNF i on server 3 to serve the maximum rate, then $S_i = \{1, 1, 2, 2, 2, 3\}$. Using S_i, when we need to deploy more VNF i instances, we simply eject sufficient server IDs from S_i and deploy VNF i instances on the respective servers accordingly; when some instances of VNF i are removed from the servers, we insert the respective server IDs into S_i.

Theorem 4. *This VNF placement scheme can guarantee server capacity constraint (12) is always respected and no migration of VNF instances occurs.*

The detailed proof is given in our technical report [18].

D. Online Algorithm

Our proposed online algorithm to solve problem (10) combines the classic ski-rental algorithm from [19] with our proposed VNF placement scheme. More specifically, the VNF instances on the servers are either marked as running or idle. The total number of running VNF i instances in time slot t is $n_i(t)$ and the total number of all VNF i instances on the servers in time slot t is $x_i(t)$. In each time slot, if $n_i(t) \geq x_i(t-1)$, then switch all the idle VNF i instances to running, eject $n_i(t) - x_i(t-1)$ elements from S_i and deploy $n_i(t) - x_i(t-1)$ VNF i instances on the respective servers; if $n_i(t-1) \leq n_i(t) < x_i(t-1)$, switch $n_i(t) - n_i(t-1)$ VNF i instances, which become idle most recently, into running state; otherwise, switch $n_i(t-1) - n_i(t)$ running VNF i instances to idle. For each idle VNF i instance, we generate a "deadline" following a given distribution: P{"deadline" $= j$} = $(\frac{\Delta_i - 1}{\Delta_i})^{\Delta_i - j} \frac{1}{\Delta_i (1 - (1 - 1/\Delta_i)^{\Delta_i})}$, where $\Delta_i = \lfloor \varphi_i / \phi_i \rfloor$. Remove the idle instance from its server completely after it has been

[1] We reasonably assume the input traffic rate $\alpha(t)$ in $[0, T]$ is within $[0, \alpha^{max}]$, *i.e.*, the datacenter's capacity is sufficient to serve the input traffic.

Algorithm 2: Online Algorithm for Single Service Chain

```
Input: $\mathbf{n}(t)$, $\mathbf{n}(t-1)$, $\mathbf{S}$, $\mathbf{x}(t-1)$
Output: $\mathbf{x}(t)$
for $i \in [L]$ do
    if $n_i(t) \geq x_i(t-1)$ then
        Switch all the idle VNF $i$ instances to running;
        Eject $n_i(t) - x_i(t-1)$ elements from $S_i$;
        Place $n_i(t) - x_i(t-1)$ instances on respective servers;
    else if $n_i(t-1) \leq n_i(t) < x_i(t-1)$ then
        Switch $n_i(t) - n_i(t-1)$ idle VNF $i$ instances to running;
    else
        Switch $n_i(t-1) - n_i(t)$ running VNF $i$ instances to idle;
    forall the idle VNF $i$ instances do
        if marked as running in the previous time slot then
            counter := 0;
            deadline := $j$ with probability $(\frac{\Delta_i - 1}{\Delta_i})^{\Delta_i - j}\frac{1}{\Delta_i(1-(1-1/\Delta_i)^{\Delta_i})}$;
        if counter $\geq$ deadline then
            Remove it from the server;
            Insert the server ID into $S_i$;
    $x_{ui}(t)$ :=total # of running and idle VNF $i$ instances on server $u$ at time $t$
```

idle for the "deadline"number of time slots and insert the corresponding server ID into S_i.

It can be proved that our Alg. 2 runs very fast.

Theorem 5. *The randomized online algorithm in Alg. 2 has a worst-case computation complexity of $O(K)$ per time slot, where K is the maximum number of VNF instances the datacenter can serve.*

The competitive ratio of our Alg. 2 to solve problem (10) can be shown to be equal to the competitive ratio achieved by the randomized ski-rental algorithm to solve problem (17).

Theorem 6. *The randomized online algorithm in Alg. 2 produces a feasible solution of (10) and achieves a competitive ratio of $e/(e-1)$.*

The detailed proof is given in our technical report [18].

V. Online Algorithm for Multiple Service Chains

We next focus on the general scenario where there are more than one service chains in the system.

The algorithm we designed for a single service chain cannot be extended to the case of multiple service chains, due mainly to the non-existence of the maximum input flow rate vector which uniquely decides a MAX VNF deployment solution for each service chain, *i.e.*, that to serve the maximal flow rates. When multiple chains coexist and may share the same VNF instances, we only have *pareto* maximal input flow rate vectors among the flows. Since the MAX VNF deployment solution no longer exists for each chain, we are not able to design VNF placement solution over time as subsets of the MAX placement solution, without incurring any VNF migration. Hence, we adopt a new approach.

Algorithm 3: Online Algorithm for Multiple Service Chains

```
Input: $\mathbf{n}(t)$, $\mathbf{x}(t-1)$
Output: $\mathbf{x}(t)$
$\mathbf{P}$ = patterns returned by Bin_Packing($\mathbf{n}(t)$);
$p$ = |$\mathbf{P}$|;
$\mathbf{K}$ = pattern multiplicities returned by Bin-Packing($\mathbf{n}(t)$);
Let $x_i$ denote the pattern in server $i$ in time $t-1$;
$S_1 = \{x_1, x_2 ..., x_U\}$;
$S_2 = \{\underbrace{P(1), ..., P(1)}_{K(1)}, ..., \underbrace{P(p), ..., P(p)}_{K(p)}\}$;
$w_{ij} = [S_2(j) - S_1(i)]^+$;
$\mathbf{x}(t)$ = Minimum_Weight_Matching($S_1$,$S_2$,$W$);
```

Our analysis has revealed that any online algorithm which uses the minimal number of VNF instances at each time slot is $(1 + \max_{i \in [I]} \frac{\varphi_i}{\phi_i})$-competitive, as Theorem 7 shows. That is, if the online algorithm finds a solution which makes $\sum_{u \in [U]} x_{ui}(t) = n_i(t)$ in (11) and further satisfies constraint (12), it is $(1 + \max_{i \in [I]} \frac{\varphi_i}{\phi_i})$-competitive.

Theorem 7. *Any online algorithm which uses the minimal number of VNF instances in each time slot and respects the server capacity constraint is $(1 + \max_{i \in [I]} \frac{\varphi_i}{\phi_i})$-competitive.*

The detailed proof is given in our technical report [18].

One natural idea is to design an algorithm that packs this minimal number of VNF instances into servers, without violating capacities. Again, this can be done using the bin-packing algorithm in [21]. The bin-packing algorithm returns a set of *patterns* and their corresponding *multiplicities* [21]. A pattern is a feasible placement solution of instances of all VNFs in a server, which can be described by an I-dimensional vector V. The i-th component, $V(i)$ in V, denotes the number of VNF i instances to be deployed on the server. A pattern can be applied to several servers and the number of times it is applied is called its multiplicity. However, the bin-packing algorithm does not tell us which actual server(s) is(are) to be used for deployment of each pattern.

We map the result patterns to individual servers, trying to minimize the deployment cost from one time slot to the next. For example, suppose in the previous time slot $t-1$, we have 2 instances of VNF 1 on server 1 and 2 instances of VNF 2 on server 2. In t, the bin-packing algorithm returns two patterns, (2,0) and (0,3), each of which is used once. Then we should map (2,0) to server 1 (i.e., run 2 instances of VNF 1 on server 1 in t) and (0,3) to server 2 (i.e., run 3 instances of VNF 2 on server 2 in t) .

Formally, the deployment cost incurred is $\sum_{i \in [I]} \varphi_i [V(i) - x_{ui}(t-1)]$, if we map a pattern V to a server u. To minimize deployment cost through mapping, we have a minimum weight matching problem, where the weight on a link connecting a pattern to a server is the corresponding deployment cost. The minimum weight matching problem can be solved efficiently, *e.g.*, using the Kuhn-Munkres algorithm [23].

Our online algorithm for VNF deployment of multiple service chains is given in Alg. 3. Note that the mapping component of the algorithm in general reduces the overall cost in (11) achieved by the algorithm, as compared to one that does no mapping but only packs $n_i(t)$ instances of VNF i into servers. Therefore, the competitive ratio that Alg. 3 achieves is at most $(1 + \max_{i \in [I]} \frac{\varphi_i}{\phi_i})$ based on Theorem 7.

Theorem 8. *The online algorithm in Alg. 3 is* $(1 + \max_{i \in [I]} \frac{\varphi_i}{\phi_i})$*-competitive.*

VI. Performance Evaluation

We evaluate the performance of our online algorithms using trace-driven simulations. We compare their performance with the offline optimum, as well the RHC(0) algorithm in [24], which exactly solves the ILP (10) in each time slot. Our algorithms can be built on existing frameworks like OpenStack [13] to automatically perform VNF instance scaling as the input traffic rate to the service chains varies.

A. Settings

Workload traces: Like in [20] and [5], we use a week of I/O traces from 1 RAID volume, CAMRESHMSA01-lvm0, at MSR Cambridge [25], representing activities in a service used by hundreds of users. We use the trace data as time-varying input traffic rates to the service chains. The trace data is normalized such that the peak load is 400Gbps, and the peak-to-mean ratio (PMR) is 4.27.

Cost benchmark: We use the cost incurred in static provisioning of VNF instances in a datacenter as a benchmark, for computing cost savings of different algorithms in comparison. In the static provisioning, we assume that the datacenter has the complete workload information ahead of time and provisions a constant number of VNF instances over time according to the peak load.

Servers and VNFs: For simplicity, we assume CPU capacity is the only bottleneck in our experiments. There are 1000 servers and each of them has 16 physical CPU cores. There are four types of VNFs to compose service chains, namely firewall, load balancer, NAT and IDS. Following configurations in [16], a firewall instance requires 4 CPU cores and can handle an incoming traffic rate of 900Mbps. A load balancer or NAT instance requires 2 CPU cores and can handle 900Mbps incoming traffic rate. An IDS instance requires 8 CPU cores and can handle 600Mbps incoming traffic rate.

Service chains: We consider the following service chains: firewall→NAT, firewall→IDS and firewall→IDS→load balancer. We assume that a firewall filters 10% of the input traffic and an IDS filters 20% of the input traffic on average. Load balancers and NATs in the service chains do not change the flow rates.

VNF operational cost: We set ϕ_i proportionally to C_i, and $\varphi_i/\phi_i \in [1, 10]$. It is generally reasonable since deployment cost is on the order of the cost to run a VNF instance for several minutes [5].

B. Performance of Algorithms for a Single Service Chain

The service chain is firewall→IDS→load balancer. According to Alg. 1 and our setup, with 1000 servers, the system can handle at most 886Gbps input traffic and in total 985 firewall instances, 1329 IDS instances and 709 load balancer instances are required to serve this traffic. One feasible MAX placement solution is to have 664 servers with 2 IDS instances each, 246 servers with 4 firewall instances each, 88 servers with 8 load balancer instances each, 1 server with 1 IDS instance, 1 firewall instances and 2 load balancer instances and 1 server with 3 load balancer instances.

1) Impact of Unit Deployment Cost to Operational Cost: Under different values of φ_i/ϕ_i, Fig. 1 shows the cost saving by comparing three algorithms to the benchmark, our algorithm in Alg. 2, offline optimum derived by solving problem (10) exactly using MOSEK [26], and the RHC(0) algorithm, respectively. We can see that our proposed randomized online algorithm can save up to 70% of the cost and it performs better than RHC(0) when φ_i/ϕ_i is greater than 4. Although RHC(0) and Alg. 2 have similar performance when φ_i/ϕ_i is less than 4, it is worth noting that Alg. 2 guarantees better performance in the worst case and is computationally tractable (RHC(0) involves solving ILPs exactly). Also, the cost reduction is decreasing as φ_i/ϕ_i increases, *i.e.*, the smaller the deployment cost is, the more cost saving we can obtain. This is because a static provisioning approach will only pay a deployment cost at the very beginning but a right-sizing approach will pay an increasing deployment costs each time it decides to vary its deployment. Therefore, we could save more cost as technology continues to advance, resulting in the reduction of deployment cost.

We also investigate the impact of φ_i/ϕ_i on the competitive ratio of different algorithms. The results are shown in Fig. 2. We can see the ratio achieved by our online algorithm is quite stable as φ_i/ϕ_i changes, while RHC(0) performs worse as φ_i/ϕ_i increases. This also confirms a previous result from [24], which proves RHC(0) is $(1 + \varphi_i/\phi_i)$-competitive.

2) Impact of Peak-to-Mean Ratio (PMR): Intuitively, comparing to static provisioning, dynamic provisioning can achieve more cost saving when the input workload varies more significantly. Our following evaluation confirms this intuition. Similar to [5], we generate the workload based on the MSR traces by scaling $\alpha(t)$ as $\overline{\alpha(t)} = K\alpha(t)^{\gamma}$, and adjust γ and K to keep the mean constant. We let PMR of the input traffic rates at different times vary from 2 to 10. Notice when PMR = 10, the maximal input traffic rate is 879Gbps, which is quite close to 886Gps. Fig. 3 shows that the cost saving increases from about 30% at PMR=2 (which is common for workload in large datacenters) to about 67% for higher PMRs (that are common in small to medium sized datacenters [20]).

C. Performance of Algorithms for Multiple Service Chains

We experiment with three service chains, namely firewall→NAT, firewall→IDS and firewall→IDS→load balancer. We compare the competitive ratios achieved by our proposed online algorithm in Alg. 3 and RHC(0), computed against the offline optimum obtained by solving the offline ILP in (10) exactly.

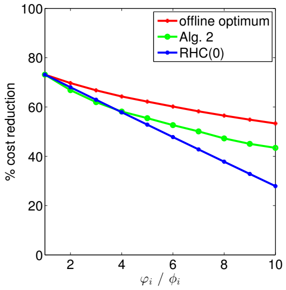

Fig. 1. Impact of φ_i/ϕ_i on cost reduction for single service chain

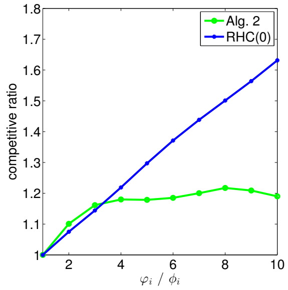

Fig. 2. Impact of φ_i/ϕ_i on competitiveness for single service chain

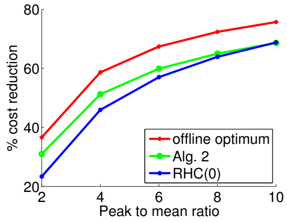

Fig. 3. Impact of PMR for single service chain

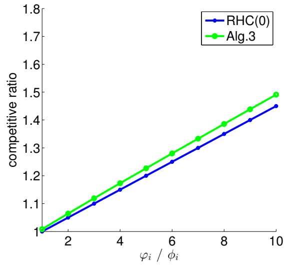

Fig. 4. Impact of φ_i/ϕ_i on competitiveness for multiple service chains

Fig. 4 shows that the competitive ratios of Alg. 3 are only slightly worse than those of RHC(0). This shows the matching step is indeed helpful. Note that Alg. 3 does not require solving an ILP in each time slot while RHC(0) does, Alg. 3 is more preferable. We also observe that the competitive ratios are increasing as φ_i/ϕ_i becomes larger, which is consistent with our theoretical competitive ratio analysis.

VII. Concluding Remarks

Network function virtualization provides a flexible way to deploy, operate and orchestrate network services with much less capital and operational expenses. Software middleboxes (e.g., ClickOS) are rapidly catching up with hardware middleboxes in performance. Network operators are already opting for NFV based solutions. We believe that our proposals for dynamic VNF provisioning and placement will have a significant impact on VNF management in the near future. Our model can be used to determine the optimal numbers of VNF instances and their optimal placement on servers, to optimize operational cost and resource utilization over the long run. Two solutions are designed: for a single service chain, we obtain a randomized online algorithm with competitive ratio $e/(e-1)$; for multiple service chains, we design a $(1+\max_{i\in[I]}\frac{\varphi_i}{\phi_i})$-competitive online algorithm, relevant to the ratio of deployment cost over operational cost. Our trace driven simulations demonstrate that the overall cost can be reduced significantly by our dynamic VNF provisioning algorithms.

References

[1] N. ISG, "White Paper Ver. 2," 2013.

[2] ETSI, "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV," http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.01.01_60/gs_NFV003v010101p.pdf.

[3] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and Implementation of a Consolidated Middlebox Architecture," in *Proc. of NSDI*, 2012.

[4] G. ETSI, "Network Functions Virtualisation (NFV); Use Cases," *V1*, vol. 1, 2013.

[5] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic Right-sizing for Power-proportional Data Centers," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 5, pp. 1378–1391, 2013.

[6] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the Art of Network Function Virtualization," in *Proc. of NSDI*, 2014.

[7] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement using SDN," in *Proc. of SIGCOMM*, 2013.

[8] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, 2012.

[9] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The Dynamic Placement of Virtual Network Functions," in *Proc. of Network Operations and Management Symposium (NOMS)*, 2014.

[10] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/Merge: System Support for Elastic Execution in Virtual Middleboxes," in *Proc. of NSDI*, 2013.

[11] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling Innovation in Network Function Control," in *Proc. of SIGCOMM*, 2014.

[12] X. Meng, V. Pappas, and L. Zhang, "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement," in *Proc. of INFOCOM*, 2010.

[13] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[14] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A Network-aware Orchestration Layer for Middleboxes in the Cloud," Technical Report, Tech. Rep., 2013.

[15] H. Moens and F. De Turck, "VNF-P: A Model for Efficient Placement of Virtualized Network Functions," in *Proc. of International Conference on Network and Service Management (CNSM)*, 2014.

[16] M. Faizul Bari, S. Rahman Chowdhury, R. Ahmed, and R. Boutaba, "On Orchestrating Virtual Network Functions in NFV," *ArXiv e-prints*, Mar. 2015.

[17] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near Optimal Placement of Virtual Network Functions," in *Proc. of INFOCOM*, 2015.

[18] "Online Scaling VNFs in Datacenters," Tech. Rep., 2015, https://www.dropbox.com/s/n1u6gang5nvoqoz/TechReport.pdf?dl=0.

[19] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive Randomized Algorithms for Nonuniform Problems," *Algorithmica*, vol. 11, no. 6, 1994.

[20] T. Lu, M. Chen, and L. L. Andrew, "Simple and Effective Dynamic Provisioning for Power-proportional Data Centers," *IEEE Transactions on Parallel and Distributed Systems,*, vol. 24, no. 6, 2013.

[21] M. X. Goemans and T. Rothvoß, "Polynomiality for Bin Packing with a Constant Number of Item Types," in *Proc. of SODA*, 2014.

[22] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, "Approximation Algorithms for Bin Packing: a Survey," in *Approximation algorithms for NP-hard problems*, 1996.

[23] J. Munkres, "Algorithms for the Assignment and Transportation Problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[24] M. Lin, Z. Liu, A. Wierman, and L. L. Andrew, "Online Algorithms for Geographical Load Balancing," in *Proc. of International Green Computing Conference (IGCC)*, 2012.

[25] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-loading: Practical Power Management for Enterprise Storage," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, p. 10, 2008.

[26] A. Mosek, "The MOSEK Optimization Software," *Online at http://www. mosek. com*, vol. 54, 2010.