Online Learning-Assisted VNF Service Chain Scaling with Network Uncertainties

Xiaoke Wang*, Chuan Wu*, Franck Le†, Francis C.M. Lau*
*The University of Hong Kong, Email: {xkwang, cwu, fcmlau}@cs.hku.hk
†IBM T. J. Watson Research Center, Email: fle@us.ibm.com

Abstract—**Network function virtualization has emerged as a promising technology to enable rapid network service composition/innovation, energy conservation and cost minimization for network operators. To optimally operate a virtualized network service, it is of key importance to optimally deploy a VNF (virtualized network function) service chain within the provisioning infrastructure (e.g., servers and the network within a cloud datacenter), and dynamically scale it in response to flow traffic changes. Most of the existing work on VNF scaling assume access to precise network bandwidth information for placement decisions, while in reality, network bandwidth typically fluctuates following an unknown pattern and an effective way to adapt to it is to do trials. In this paper, we address dynamic VNF service chain deployment and scaling by a novel combination of an online provisioning algorithm and a multi-armed bandit optimization framework, which exploits online learning of the available bandwidths to enable optimal deployment of a scaled service chain. Specifically, we adopt the online algorithm to minimize the cost for provisioning VNF instances on the go, and a bandit-based online learning algorithm to place the VNF instances which minimizes the congestion in a datacenter network. We demonstrate effectiveness of our algorithms using solid theoretical analysis and trace-driven evaluation.**

I. Introduction

Network Function Virtualization (NFV) is an emerging technology that consolidates 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 conservation and cost minimization for network operators [1]. Example network functions include firewalls, intrusion detection systems (IDSs), and load balancers. A network function deployed following the NFV paradigm is termed a virtualized network function (VNF).

An important use case of NFV is virtualization of enterprise network functions. A significant number of network functions (comparable to the number of access routers) are typically in need in enterprise networks. The network functions constitute one or multiple network function service chains, to support their services [2]. For example, the access service is commonly provisioned by a service chain consisting of an intrusion detection device, a firewall, and a load balancer that distributes incoming traffic to a pool of servers.

A common proposal to implement an enterprise's service chains is to run the VNF instances on virtual machines (VMs) in a cloud computing platform, *i.e.*, the enterprise's datacenter or a public cloud datacenter [3]. 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. Provisioning cost of VNF instances is mainly due to power consumption to operate servers and cooling facilities, largely decided by the numbers and the types of VMs running different VNFs on the servers. On the other hand, launching new VNF instances 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 [4]. Such a deployment cost should be minimized as well, by avoiding frequent deployment and removal of VNF instances when flow traffic fluctuates along a service chain.

In the meantime, due to the sharing nature of datacenters, there are always other workloads which coexist with the VNF service chain in a datacenter, which share link bandwidths in the datacenter network. The background traffic caused by the other workloads is commonly time varying, following unknown patterns [5]. Being irrespective of the background traffic when deploying/scaling the VNF service chain may well lead to unacceptable performance degradation due to network congestion [6]. It is important to come up with a deployment solution that guarantees the best performance even in case of unknown, varying background traffic.

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

Given a VNF service chain, how can we design online solutions that dynamically scale VNF instances and pack them onto servers, to achieve close-to-offline-minimum provisioning cost over the long run of the system, as well as minimize link congestion levels, in face of unknown time-varying background traffic?

The key challenges lie in two folds: (i) the unknown nature of future rates of flows arriving at the service chain, which can vary significantly over time; (ii) the uncertainty of background traffic along links in the datacenter. To handle (i), we need to strategically make online deployment decisions on the go, avoiding undesirable situations of destroying a VNF instance immediately followed by launching the same instance. To address (ii), we seek to exploit knowledge of available network bandwidths acquired from past experience, while carrying out some exploration to acquire more information about background traffic on different links. Such exploration and exploitation are core components in a multi-armed bandit (MAB) optimization framework [7], which we advocate in our algorithm design. We novelly combine an online VNF provisioning algorithm and the bandit-based online learning framework to come up with our online VNF deployment and scaling solutions. Our contributions are summarized as follows:

▷ We decouple the problem into two parts. The first part computes the numbers of VNF instances to deploy according to the varying traffic rates to the service chain. We propose a randomized online algorithm to compute the numbers and guarantee overall cost at most $e/(e-1)$ times the offline optimal cost. The algorithm is an adaptation of the classic ski-rental algorithm [8] which leads to an optimal competitive ratio.

▷ Based on the numbers derived by the online algorithm, the second part is to determine placement of the VNF instances on servers, according to the learned congestion levels on the network links inter-connecting the servers, to load balance traffic on different links. We adapt an ϵ_t-greedy multi-armed bandwidth algorithm, and show that the accumulated gap of performance achieved by our algorithm from that of a sensible baseline, grows sublinearly with time, a desirable performance target in online learning [7].

▷ Our algorithms are simple and easy to implement in practice in an online fashion. We demonstrate the effectiveness of our algorithms using both theoretical analysis as well as trace-driven evaluation.

In the rest of the paper, previous literature is reviewed in Sec. II. We formulate the system model in Sec. III, and design online algorithms for VNF scaling and placement in Sec. IV. We discuss several extensions in Sec. V, and evaluate the performance of our algorithms in Sec. VI. Sec. VII concludes the paper.

II. Related Work

Before the emergence of NFV, VM placement has been addressed by a rich body of work. See [9] for a survey of VM allocation policies available in federated clouds. Different from VM placement, VNF instances are often chained together such that bandwidth required for interconnecting VNFs should be taken into consideration. A number of work have recently appeared on VNF placement. Stratos [10] is a system that provides an abstraction for placement and scaling of VNF instances, as well as involves a three-stage heuristic algorithm to reduce network congestion. VNF-P [11] presents an optimization model for VNF resource allocation to minimize the number of servers in use, while respecting end-to-end network latency constraints. Bari *et al.* [12] study a similar problem to optimize network operational costs and utilization, without violating service level agreements. Cohen *et al.* [13] investigate the problem of optimal VNF placement over geo-distributed datacenters, and design several algorithms achieving proven approximation ratios. Except [13], the other studies propose heuristics to solve the respective VNF placement problems, without providing any theoretical performance guarantee. All the above work focus on offline/one-time VNF placement and resource allocation, instead of dynamic VNF provisioning, which calls for efficient online algorithms. In our previous work [14], we propose an online algorithm for VNF deployment, which is $e/(e-1)$-competitive in case of unknown future input traffic rate to the service chains. In that work, we assume that inter-server bandwidths are always sufficient, and do not take background traffic and bandwidth uncertainty into consideration. In real-world data centers, inter-server bandwidths may not always be sufficient and any VNF service chain deployment scheme needs to take bandwidth uncertainty into considerations, which is the main motivation of the current work. In addition, Palkar *et al.* [15] also assume that the network bandwidth information is known in their NFV scheduling framework.

Multi-armed bandit optimization is an effective online learning and optimization framework [16]. In a multi-armed bandit problem [17] [18], an agent faces a set of arms (actions) and needs to select one arm or a set of arms repeatedly in a sequence of rounds. After pulling an arm, the agent incurs a loss (or receives a reward) on the pulled arm, which is a realization of the unknown, underlying loss (or reward) distribution associated with that arm. The goal is to incur the minimal cumulative loss (or obtain the maximal cumulative reward) over the whole time horizon. The focus of bandit algorithm design is to achieve a good tradeoff between exploration and exploitation, i.e., to try some less attempted arms that might provide a better return (exploration), or stick to the arm that has brought low loss (or high reward) so far (exploitation). The performance of a bandit algorithm is evaluated by regret, which is commonly computed as the gap between the overall loss incurred (or reward obtained) by an offline solution and the expected loss (or reward) of the bandit algorithm. To our knowledge, we are the first in applying bandit optimization to make optimal VNF placement decisions in the face of network uncertainties between VNF instances.

III. Problem Model

A. System Model

We consider an enterprise deploying a service chain in a cloud datacenter. The service chain consists of an ordered sequence of I VNFs, i.e., VNF 1, VNF 2, ..., VNF I. For ease of problem formulation, a dummy VNF 0 is added to the head of the chain to indicate the aggregate source of the traffic flows passing through the chain. Instances of the VNFs are to be deployed on VMs of U servers in the cloud datacenter. An instance of VNF i consumes a c_i amount of resource and can maximally process flow traffic at the rate of b_i (in Mbps), without incurring prolonged packet queueing delays that violate preset performance thresholds. It is worth noting that the flow rate of traffic may change after being processed by a network function [10]. Hence we define a change ratio λ_i for VNF i, denoting the ratio of the rates of the flow after and before passing the VNF. We assume that proportional flow routing is used to route traffic among VNF instances, i.e., flow from VNF $i-1$ is split equally among instances of the next-stage VNF i, as a practical approach. The available amount of resource to run the VNF instances may differ among different servers since these servers may run other workloads concurrently. We denote the available amount of resource in server u by C_u, decided by deducting allocated resources to VMs running other workloads from the server capacity. Though we only model one type of resource on a server, we note that the model is readily extensible to the case of multiple types of resources on each server.

The servers are inter-connected by a set of routers, e.g., via a fat-tree switching network. We assume that the logical link between two servers, which consists of multiple physical links interconnected by switches, does not share common physical links with other logical inter-server links. This assumption is only made to simplify our problem formulation. We will see that this assumption can be dropped, as long as the network topology is known to us, in Sec. V.

Practically, we are unaware of the rate of background traffic on the logical link between any pair of servers beforehand, as produced by other workloads running on the servers. We assume that the rate of background traffic from server u to server v, denoted by $l_{uv}(t)$, varies following an unknown distribution, with a hidden mean value of $\bar{l}_{uv}$. We can infer the background traffic rate from server u to server v if and only if our deployment solution places instances of two consecutive VNFs, VNF i and VNF $i+1$, in the service chain on server u and server v, respectively: we observe achieved transmission bandwidth between the two servers, e.g., by keeping track of the number of packets transmitted in each time slot, and estimate background traffic rate by deducting the achieved bandwidth from assumed capacity of the inter-server connection.[1] To quantify the performance of our NFV deployment, we seek to achieve balanced bandwidth consumption on inter-server links (due to both background traffic and our service chain traffic), to minimize congestion on the links.

The system works in a time-slotted fashion, spanning time slots $1, 2, \ldots, T$. The input traffic rate to the service chain varies from time to time, denoted by $\alpha(t)$ (*e.g.*, in Mbps) at time t. Similarly, we denote the total rate of traffic to instances of VNF i by $\alpha_i(t)$. We have $\alpha_i(t) = \alpha(t)\prod_{j\in[i-1]}\lambda_j$. We use $[X]$ to indicate the set $\{1, 2, \ldots, X\}$ in this paper.

B. Problem Formulation

We aim to design online solutions that dynamically scale VNF instances and pack them onto servers, to achieve close-to-offline-minimum provisioning cost over the entire running span of the system, while in the meantime, minimizing the congestion in face of unknown varying background traffic. There is typically a trade-off between cost minimization and congestion minimization: by provisioning more VNF instances, the cost is increased but the congestion is likely to decrease, and vice versa. In this paper, we seek to minimize the cost first, and then minimize congestion, since cost is still a major concern for network service operators.

[1]We do not need to know the exact physical capacity of the inter-server connection, but only use a reasonable, large bandwidth capacity when running our algorithm, e.g., 40Gbps.

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. We further define $x_i(t) = \sum_{u\in[U]} x_{ui}(t)$ as the total number of instances of VNF i in t. (ii) Routing variable $y_{uvi}(t)$, which represents the rate of traffic (in Mbps) forwarded from instances of VNF $i-1$ on server u to instances of VNF i on server v. Since we adopt proportional routing, we have $y_{uvi}(t) = \alpha_i(t)x_{u(i-1)}(t)x_{vi}(t)/x_{i-1}(t)x_i(t)$.

The service chain 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

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

II. VNF deployment cost. Deploying a new instance of VNF i requires transferring a VM image containing the network function into the datacenter, booting it and attaching it to devices on the server. We associate a cost φ_i for deploying a newly added instance of VNF i. We ignore cost for migrating existing instances from one server to another, since technology for live VM migration has been more and more mature with less impact on system performance as compared to deploying a new instance [19].[2] The overall VNF deployment cost can be expressed as

$$\mathbb{D} = \sum_{t\in[T]}\sum_{i\in[I]} \varphi_i [x_i(t) - x_i(t-1)]^+ \tag{2}$$

where $[x_i(t) - x_i(t-1)]^+ = \max\{x_i(t) - x_i(t-1), 0\}$, indicating the number of newly added instances of VNF i in t.

The total cost thus can be expressed as $\mathbb{O} + \mathbb{D}$.

On the other hand, the congestion level of the system is determined by the most congested inter-server connection, i.e., that transmits the largest volume of traffic, as follows:

$$\max_{u\in[U], v\in[U]} \sum_{i\in[I]} y_{uvi}(t) + l_{uv}(t) \tag{3}$$

The set of constraints that the decision variables should respect include the following.

First, all incoming traffic to the 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):

$$\sum_{v\in[U]} y_{0v1} \geq \alpha(t), \forall t \in [T] \tag{4}$$

[2]We also ignore the bandwidth consumption due to VM migration for simplicity, but can readily model the increase in link bandwidth usage since we know the size of the VM.

where $y_{0v1}(t)$ denotes the incoming traffic rate (from the dummy VNF 0 on an imaginary server 0) directed to instances of VNF 1 on server v.

(ii) Flow conservation at instances of VNF i on each server u:

$$\lambda_i \sum_{v\in[U]} y_{vui}(t) = \sum_{v\in[U]} y_{uv(i+1)}(t), \forall u \in [U], i \in [I], t \in [T] \tag{5}$$

where the left-hand side is the overall input traffic rate to instances of VNF i on server u multiplied by the change ratio, and the right-hand side computes the overall output traffic rate from instances of VNF i on server u to next-stage VNF deployed on different servers.

(iii) Provisioning a sufficient number of instances of each VNF on each server, whose overall processing capacity can serve the overall rate of received traffic from instances of the previous-stage VNF:

$$\sum_{v\in[U]} y_{vui}(t) \le x_{ui}(t) b_i, \forall i \in [I], u \in [U], t \in [T] \tag{6}$$

Next, resource capacity on each server should not be over-committed by the deployed VNF instances at any time:

$$\sum_{i\in[I]} x_{ui}(t) c_i \le C_u, \forall u \in [U], t \in [T] \tag{7}$$

Finally, we would like to avoid frequent migration of VNF instances from one server to another. When background traffic changes over time, it is possible that the deployment solution achieving the lowest congestion level on inter-server connections is different from the deployment solution in the previous time slot. In that case, we may allow some VNF instances to migrate from one server to another. However, it is not realistic to perform excessive migration due to its impact on service continuity. Therefore, we limit the times of VNF migration in each time slot by a parameter M:

$$\sum_{i\in[I]} \Big(\sum_{u\in[U]} [x_{ui}(t) - x_{ui}(t-1)]^+ - [x_i(t) - x_i(t-1)]^+ \Big) \le M \tag{8}$$

where the first part of the left-hand side is the sum of newly deployed instances and migrated instances, and the second part is the number of newly deployed instances.

We list key notation in Table I for ease of reference.

$[X]$	integer set {1,2,...,X}
U	# of servers in the system
I	# of VNFs in the service chain
M	times of VNF instance migration allowed per time slot
T	total # of time slots
$l_{uv}(t)$	rate of background traffic from server u to server v in t
$x_{ui}(t)$	# of VNF i instances on server u in t
$y_{uvi}(t)$	rate of service chain traffic from u to VNF i instances on v in t
C_u	capacity of server u
b_i	processing capacity of an instance of VNF i
λ_i	flow rate change ratio when passing VNF i
α_i	total traffic rate to VNF i instances
ϕ_i	per-time-slot operational cost of an instance of VNF i

TABLE I
NOTATION

IV. Online Algorithms for VNF Provisioning and Placement

A. Online VNF Provisioning

We observe from the above formulation that in order to minimize total cost, we only need to consider the value of $x_i(t)$ instead of $x_{ui}(t)$. This is because constraints (4), (5) and (6) together determine the minimal number of instances of each VNF required. Let $n_i(t)$ denote the minimal number of instances of VNF i needed in t. Then $n_i(t) = \lceil \alpha_i(t)/b_i \rceil$. Hence, to minimize total cost, we only need to determine how many VNF instances to provision in each time slot. We reformulate the cost minimization problem as follows.

$$\min \sum_{t\in[T]} \sum_{i\in[I]} (\phi_i x_i(t) + \varphi_i [x_i(t) - x_i(t-1)]^+) \tag{9}$$

$$\text{s.t.} \quad x_i(t) \ge n_i(t), \forall i \in [I], t \in [T] \tag{10}$$

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

This problem is then a variant of the classic ski-rental problem [8]. In the classic ski-rental problem, one is going skiing for an unknown number of days (depending on how long the snow will last). Everyday he decides whether to buy a ski, paying a one-time cost, or rent a ski, paying a much lower price for the day. A ski-rental algorithm is an online buy/rent decision making algorithm, aiming to minimize the ratio between what one pays using the algorithm and what one would pay optimally if he knew the number of days to ski. In this problem, we decide whether to remove an instance from a server, potentially paying a deployment cost for running it anew, or keep the instance running, paying an operational cost. We design the online algorithm in Alg. 1 based on classical ski-rental algorithms [8] [20] to solve the problem, striking a balance between operational cost and deployment cost in face of unknown future input traffic rates to the service chain.

It can be proved that the randomized online algorithm in Alg. 1 produces a feasible solution of problem (9) and achieves a competitive ratio of $e/(e-1)$, which is the same as that by a classical online ski-rental algorithm [8].

B. Online VNF Placement

Once we obtain the solution of problem (9), we can minimize the congestion level in the system by carefully placing these VNF instances. The congestion minimization problem is as follows:

$$\min \max_{u,v\in[U]} \sum_{i\in[I]} y_{uvi}(t) + l_{uv}(t) \tag{11}$$

$$\text{s.t.} \quad \text{constraints (4)(5)(6)(7)(8)}$$

$$\sum_{u\in[U]} x_{ui}(t) = x_i(t), \forall i \in [I], t \in [T] \tag{12}$$

Algorithm 1: Online VNF Provisioning Algorithm in t

```
Input: $\alpha_i, x_i(t-1), n_i(t-1)$
Output: $x_i(t), n_i(t)$
$n_i(t) := \alpha_i(t)/b_i$;
for $i \in [L]$ do
    if $n_i(t) \geq x_i(t-1)$ then
        Switch all the idle VNF $i$ instances to running;
        Deploy additional $n_i(t) - x_i(t-1)$ instances on 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 $(\frac{\Delta_i-1}{\Delta_i})^{\Delta_i-j}\frac{1}{\Delta_i(1-(1-1/\Delta_i)^{\Delta_i})}$, where $\Delta_i = \lfloor\varphi_i/\phi_i\rfloor$;
        if counter $\geq$ deadline then
            Remove it from the server;
    $x_i(t)$ :=total # of running and idle VNF $i$ instances on servers
```

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

Note that $x_i(t)$ is already determined by Alg. 1, and now we only need to decide $x_{ui}(t)$. Without constraint (8), we can prove this problem can be reduced to the bottleneck travelling salesman problem (TSP) [21]. Therefore, it is NP-hard to obtain the exact solution to the problem.

Theorem 1. *Problem (11) is NP-hard and can be reduced to the bottleneck TSP problem when we do not consider constraint (8).*

Proof: The goal of the bottleneck TSP problem is to find a Hamiltonian path on a set of points in order to minimize the edge with the largest weight in the tour or path. We consider a special case of problem (11) where the service chain has a length of U and each server only has room for one VNF instance. In addition, the input traffic rate to the service chain is sufficiently small compared with the background traffic. Then, the feasible solution of problem (11) is a Hamiltonian path on the set of servers. Our goal is equivalent to find a path whose most congested link has the lightest background traffic among all the paths. This exactly defines the bottleneck TSP problem. Since the bottleneck TSP problem is NP-hard to solve, problem (11) is also NP-hard to solve. ■

However, when we include constraint (8), the solution space of problem (11) is limited, as the following theorem shows.

Theorem 2. *Let N denote the number of newly added or removed instances in a time slot. There are at most $I^M U^{2M+N}$ feasible VNF placement solutions in the time slot.*

Proof: Given the placement solution in the previous time slot, when we add or remove N instances, there are at most U^N different solutions. Then we migrate M instances. Each time we perform a VNF instance migration, there are at most UI types of source VNF instances and U destination servers to be chosen. Therefore, we consider at most $I^M U^{2M+N}$ feasible VNF placement solutions. ■

Since we would like to react promptly to the time-varying traffic, the duration of each time slot should be kept small. Within a small interval, the number of instances to be removed or added, N, is typically small (less significant traffic rate variation over a small amount of time), and the allowed number of migration to perform, M, should be small as well, to prevent large impact on system performance. Therefore, enumerating the solution space is computationally tractable.

However, we are unaware of the time-varying background traffic rate, $l_{uv}(t)$. We seek to design a multi-armed bandit based online learning algorithm to estimate the background traffic rate over time, and strategically make VNF placement decisions accordingly. In our problem, the arm is an inter-server connection (logical link). In each time slot, we decide the deployment solution, which further decides the inter-server links for sending the service chain traffic, i.e., a set of arms to pull. The background traffic on the selected connections can be estimated, as a realization of the unknown, underlying background traffic distribution on that connection.

In existing multi-armed bandit problems, typically only one arm or the same number of arms are pulled in each round; the corresponding bandit algorithms, e.g., UCB [7], EXP3 [22], achieve a good tradeoff between exploiting the arm (set) that has provided the best result so far and exploring a new arm (set), for overall performance optimization. In our problem, the number of arms to pull may differ from one time slot to the next, and we cannot directly decide the arm set, but rather infer it based on the deployment decisions made. We need to carefully design a strategy to explore different arms. The rationale is as follows: Over time, we observe the background traffic in the inter-server links based on our service chain deployment. Due to the stochastic nature of the background traffic, our observation is noisy. Instead of always sticking to the connections with high available bandwidth so far, we should make some explorations to use again links which have exhibited large background traffic, with some probability. In the meanwhile, we should not explore too much either, as each time we choose a suboptimal link to route the service chain traffic, we may introduce some unnecessary congestion.

To this end, we adapt an ϵ_t-greedy algorithm [7] to deal with the hardness of arm selection in our problem. The idea of the algorithm is that we choose an optimal deployment solution with a high probability and pick a random feasible solution with a small probability, based on current estimation of background traffic rates. By doing so, we exploit good links frequently while in the meantime, doing some explorations by picking random links. Also, the probability of doing exploration decreases over time, so that the algorithm will converge to the optimal solution. In our problem, sometimes picking a random deployment solution does not provide more

Algorithm 2: Online Learning Algorithm for VNF Placement in t

```
Input: $\alpha_i, x_i(t)$, $0 < c < 1$
Output: $x_{ui}(t)$
$\epsilon_t = c/t \in (0, 1]$;
Compute all the feasible solutions of problem (11) based on $x_{ui}(t-1), \tilde{l}_{uv}$ and $T_{uv}$;
Pick a random number between 0 and 1;
if the random number < $1 - \epsilon_t$ then
    Choose the optimal solution of problem (11);
else
    Randomly choose a solution;
    while the solution is dominated by optimal solution do
        Randomly choose a solution of problem (11);
if $\sum_{i\in[I]} y_{uvi}(t) > 0$ then
    estimate $l_{uv}(t)$ as capacity of link $uv$ minus $\sum_{i\in[I]} y_{uvi}(t)$;
    $\tilde{l}_{uv} = (\tilde{l}_{uv}T_{uv} + l_{uv}(t))/(T_{uv} + 1)$;
    $T_{uv} = T_{uv} + 1$;
```

information than choosing the optimal solution, because the connections chosen by the random solution might be a subset of the connections chosen by the optimal solution. We say such a solution is *dominated* by the optimal solution. In this case, we should discard this random solution and pick another random solution.

Let T_{uv} denote the times that link uv is used to route service chain traffic and $\tilde{l}_{uv}$ denote the estimated mean value of the background traffic rate of link uv. The algorithm is given in Alg. 2.

Finally, we combine Alg. 1 and Alg. 2 to produce the complete online algorithm in Alg. 3.

Algorithm 3: Complete Online Algorithm for t

```
Input: $\alpha_i(t), x_i(t-1), n_i(t-1)$
Output: $x_{ui}(t), n_i(t)$
Determine $x_i(t)$ and $n_i(t)$ by Alg. 1;
Determine $x_{ui}(t)$ by Alg. 2;
```

C. Analysis of Algorithm 2

Definition of Regret. Multi-armed bandit based online learning is commonly described as a repeated game between a player and an adversary. In the beginning of each time slot, we (the player) choose a placement solution X_t from a feasible solution set χ_t (of problem (11)). The adversary samples the background traffic rate $l_{uv}(t)$ for each inter-server connection from a respective distribution which is unknown to us. We use $f_t(X_t)$ to denote the congestion level we can obtain when choosing the placement solution X_t, i.e., value of the function in (3) evaluated at X_t. We can observe the background traffic rate $l_{uv}(t)$ if and only if the flow paths derived from X_t includes the link uv. Unlike the full-information setting, in bandit-based online learning, we do not observe the entire loss function f_t at all possible X_t's, but only its value at the chosen X_t. Our goal is to minimize the accumulated congestion level over all T time slots.

The expected accumulative congestion level achieved by our algorithm after T rounds is $\mathbb{E}[\sum_{t=1}^{T} f_t(X_t)]$. To evaluate how good this accumulative congestion level is, we compare it to that achieved by a baseline strategy. The baseline strategy is a sequence of deterministic decisions (on service chain placement) for T time slots. A common way to evaluate a bandit algorithm's performance is to measure its external pseudo-regret (which we abbreviate as the *regret*), using baseline decisions which achieve the largest gap from our online solutions, defined as follows [7] [22] [23] [24]:

$$\mathbb{E}[R(T)] = \max_{Y_t\in\chi_t} \mathbb{E}[\sum_{t=1}^{T}(f_t(X_1, ..., X_t) - f_t(X_1, ..., X_{t-1}, Y_t))] \quad (14)$$

In our problem, the domain of f_t only depends on the placement solution in the previous time slot. In other words, f_t can be a function of the immediately previous decision and the current decision, i.e.,

$$f_t(X_1, ..., X_t) = f_t(X_{t-1}, X_t)$$

Therefore, the regret definition in Eq. (14) becomes

$$\mathbb{E}[R(T)] = \mathbb{E}[\sum_{t=1}^{T} f_t(X_{t-1}, X_t)] - \max_{Y_t\in\chi_t} \sum_{t=1}^{T} f_t(X_{t-1}, Y_t) \quad (15)$$

Analysis of Regret. We show that if the set of feasible solutions does not change, i.e., $\chi_1 = \chi_2 = ... = \chi_T$, our bandit algorithm can guarantee a regret sublinear to T.

Theorem 3. *If $\chi_1 = \chi_2 = ... = \chi_T$, the expected regret of our bandit algorithm in Alg. 2 over T time slots is bounded as*

$$E(R(T)) \leq dk + cd\log(\frac{T-1}{k+1}), \quad (16)$$

where c is the input parameter in Alg. 2, $d = f_t(Y_t) - \max_{X\in\chi_t} f_t(X)$ is the gap between the best baseline placement solution and the worst placement solution, and $k = e^{K/c}$, where $K = |\chi_t|$ is the size of the solution space.

Proof: The regret can be divided into two parts: (1) regret incurred due to doing some exploration to discover the optimal solution; (2) regret due to deviating from the optimal solution by having certain probability to explore other suboptimal solutions.

For the first part, in every time slot t, there is an exploration probability $\epsilon_t = c/Kt$, where $0 < c < 1$ to choose the optimal solution. In expectation, it takes k time slots to choose the optimal solution, where k is derived from the following inequality:

$$\sum_{t=1}^{k} \frac{c}{Kt} \geq 1.$$

Solving this inequality gives us $k = e^{K/c}$. That is, in the first k time slots, we are exploring suboptimal solutions, which leads to a regret of $de^{K/c}$.

For the second part, the expected regret accumulated up till time t is bounded:

$$E(R^2(T)) \leq d \sum_{t=k+1}^{T} \frac{c}{t}.$$

This is because the optimal solution is to choose placement solution Y_t, but due to exploration, we have ϵ_t chance to choose another placement solution, which at most differs d from the optimal solution. Bounding a discrete sum by an integral, we have

$$E(R^2(T)) \leq cd \int_{t=k+1}^{T-1} \frac{1}{x} dx = cd \log(\frac{T-1}{k+1}).$$

■

$\chi_1 = \chi_2 = ... = \chi_T$ holds when the input traffic rate to the service chain is stable over time. Nevertheless, our algorithm can guarantee a sublinear regret too when the input traffic rate varies over time. The main rationale is that the uncertainties we would like to estimate are the background traffic rates traversing the links, and even though for different traffic rates we have different feasible solution spaces, all these solutions are dealing with the same set of uncertainties.

Theorem 4. *(Chernoff-Hoeffding bound). Let $X_1, ..., X_n$ be random variables with common range [0, 1] such that $E[X_t] = \mu$. Let $S_n = X_1 + \cdots + X_n$. Then for all $a > 0$,*

$$\mathbb{P}\{S_n > n\mu + a\} \leq e^{-2a^2/n}, \mathbb{P}\{S_n < n\mu - a\} \leq e^{-2a^2/n} \quad (17)$$

Lemma 5. *In T time slots, each inter-server link is explored for at least $U(U-1)\log(T)/c$ times in expectation.*

Proof: In time slot t, due to exploration in our bandit algorithm, for any link, there is at least $\frac{c}{U(U-1)t}$ chance that the link will be chosen to route traffic. Adding up these probabilities, each link is at least explored $U(U-1)\log(T)/c$ times in expectation. ■

Theorem 6. *The expected regret of our bandit algorithm in Alg. 2 grows sublinearly with T.*

Proof: The regret consists of two parts: (1) regret due to making random trials to explore the links which are not in the routing paths of the optimal solution; (2) regret due to insufficient information which misleads us to choose the suboptimal solution. The regret analysis of the first part is similar to what we did for part (2) in proof of Theorem 3, and the regret is bounded by $cd \log(T-1)$.

To analyze the regret of the second part, based on the Chernoff-Hoeffding bound in Theorem 4 and Lemma 5, the probability that our estimation on the background traffic differs from the true expected value by a small constant number is smaller than g/T, where g is a parameter. The chance that we overestimate in a suboptimal solution and underestimate in an optimal solution simultaneously is even smaller. Therefore, the regret in the second part also grows sublinearly with time.

Summing up the regret in these two parts, the expected regret accumulated due to our bandit algorithm grows sublinearly with time.

■

V. Discussions

We next discuss a few directions to further improve the performance of our bandit-based online algorithm in Alg. 2.

A. Solution Space Reduction

Although the size of the solution space χ_t can be as large as $I^M U^{2M+N}$ initially, when we learn more and more of the available bandwidth on each inter-server link, the solution space can be reduced. The reason is that after gaining more information of each link, we get to know which link is likely to be most congested for a given placement solution. Then in a time slot, for each link, there is one placement solution which routes the minimal amount of service chain traffic in that link while the link is the most congested link among all links exploited in that solution. Such a solution is a candidate best solution, and we only need to compare $U(U-1)$ such candidate solutions to obtain the optimal placement solution for that time slot. When we randomly choose a solution in Alg. 2, we only need to consider these $U(U-1)$ candidate solutions as well.

B. Handling Correlation among Inter-server Links

In our model, we assume that there is no correlation among server-to-server links. Servers in a datacenter are often connected through a special network topology such as a fat-tree. The logical link between a pair of servers is composed by a series of physical links, and different logical links can share the same physical links. If we send some traffic from server u to server v, we may also make the logical links connecting other servers more congested. The congestion level of the system should be more accurately the congestion level of the most congested physical link. We can extend our algorithm to deal with this case: Each time we send traffic along the link between server u and server v, as long as we can obtain feedback on the available bandwidth of physical links constituting the overlay link, we can retain and update estimations of background traffic on these physical links instead of logical overlay links in our algorithm; when we compute the optimal placement solution, we will also make use of bandwidth estimations of these physical links. The rest of our Alg. 2 remains.

VI. Performance Evaluation

We evaluate the performance of our online algorithms using trace-driven simulation and small-scale experiments. When evaluating our online VNF provisioning algorithm in Alg. 1, we compare it to the offline optimum solution and the RHC(0) algorithm in [25], which solves the ILP problem (9) exactly in each time slot. When evaluating our bandit-based online

VNF placement algorithm in Alg. 2, we compare it to an algorithm which knows the distributions of background traffic beforehand and always chooses the optimal placement solution in each time slot based on the placement solution in the previous time slot.

A. Settings

Servers and VNFs: In our simulations, there are 16 servers inter-connected through a 4-ary fat tree and a 10Gbps Ethernet. There are multiple paths between two servers, and we explicitly control which path is chosen to route traffic from one server to another, as exemplified in Fig. 1: When there are multiple available uplinks to aggregation switches, if the server is connected to the first (second) port in the edge switch, we choose the first (second) uplink. We take a similar approach when there are multiple uplinks to core switches. Note that the logical links among servers are correlated in the fat tree topology. For example, in the figure, we can see that the green path is shared by flows from the red server to the green server and from the blue server to the green server. We will show even without explicit feedback on physical-link level available bandwidth, our bandit algorithm still performs better than others.

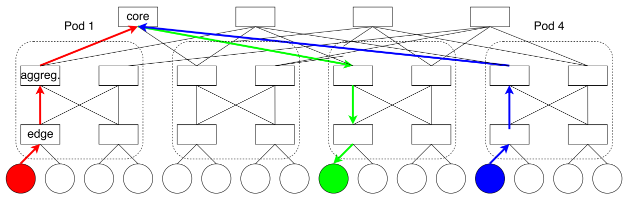

Fig. 1. 4-ary Fat Tree

The service chain is composed by four types of VNFs, i.e., load balancer, firewall, NAT and IDS. Following configurations in [12], a load balancer, firewall, NAT or IDS instance can handle an incoming traffic rate of 900Mbps, 900Mbps, 900Mbps or 600Mbps respectively. Firewall and IDS have a flow rate change ratio of 0.9 and 0.8, respectively. We assume that at most three VNF instances can be placed on a server. The operational cost of running a VNF instance depends on the number of CPUs it requires: a load balancer, firewall, NAT or IDS instance require 2, 4, 2 or 8 CPUs, respectively.

Workload traces: As in [20] and [4], we use a week of I/O traces at MSR Cambridge [26], 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 5Gbps, and the peak-to-mean ratio (PMR), which captures the gap between the peaks and valleys of the workload, is 5.13.

We study the performance of our algorithms under two types of background traffic, i.e., uniform and permutation traffic matrix. For uniform background traffic, a server sends 200Mbps traffic to every other server in expectation with support in the range of $[0, 1]$ Gbps. For permutation traffic, we randomly generate a permutation matrix and route 2Gbps traffic between two servers in expectation, with support in $[0, 3]$ Gbps. We use the following distribution to guarantee that the expected background traffic volume is a with support in $[0, b]$. Let $p(x)$ denote the probability density function:

$$p(x) = \begin{cases} \frac{b-a}{ab} & x < a, \\ \frac{a}{b^2-ab} & a \le x \le b. \end{cases} \tag{18}$$

B. Performance of Online VNF Provisioning

We vary φ_i/ϕ_i, the ratio of unit deployment cost to unit operational cost, and show in Fig. 2 the competitive ratio achieved by our Alg. 1 and the RHC(0) algorithm, as compared to the offline optimum derived by solving problem (9) exactly. We can see that our randomized online algorithm performs better than RHC(0) when the deployment cost is more prominent, and has a stable performance.

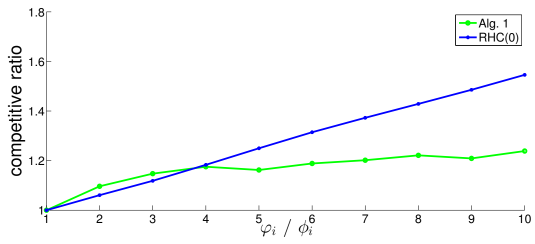

Fig. 2. Impact of φ_i/ϕ_i on online VNF provisioning

C. Performance of Online VNF Placement

We place VNF instances randomly on the servers at the beginning. Then in each later time slot, we allow at most one VNF instance to be migrated from one server to another. The regret achieved by Alg. 2 under two different background traffic patterns is shown in Fig. 3. The positive regret values show that our bandit-based online learning algorithm performs only slightly worse than an algorithm which knows the distribution of background traffic rates beforehand. Our bandit algorithm performs consistently well regardless of the characteristic of the background traffic and the regret is stable over time.

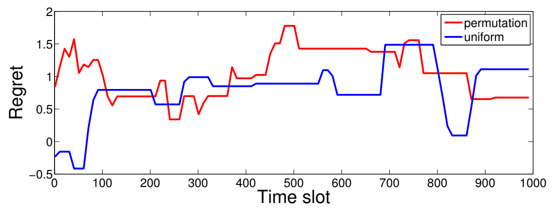

Fig. 3. Regret of the bandit algorithm: simulation

D. Experimental Evaluation of Online Learning Algorithm

Other than simulations, we also test the performance of our bandit algorithm using testbed experiments in a small server cluster. The cluster consists of 8 servers, connected by a 10Gbps switch. Each server has 16 CPUs and can accommodate 4 firewalls, 8 load balancers, 8 NATs or 4 IDS instances. We run the permutation background traffic among

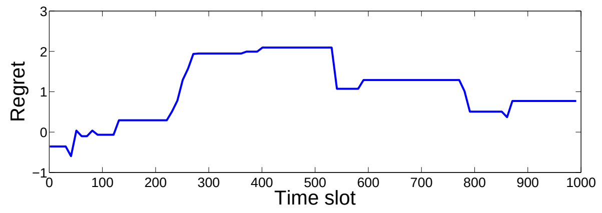

Fig. 4. Regret of the bandit algorithm: experiments

these servers. Other settings in the experiments are the same as in the simulations. The regret achieved by Alg. 2 is shown in Fig. 4. We can see that the regret is small and does not grow linearly with the increase of time either, very similar to the simulation results.

VII. Concluding Remarks

Network function virtualization provides a flexible way to deploy, operate and orchestrate network services with much lower capital and operational expenses. Dealing with the uncertainties in the input traffic rates and the background workloads when deploying VNF service chains is critical for cost reduction and service performance guarantee. This paper proposes practically useful online algorithms for VNF service chain deployment and scaling, in the face of such uncertainties. The randomized online VNF provisioning algorithm achieves a competitive ratio of $e/(e-1)$, and the bandit-based online learning algorithm for VNF placement can guarantee an $O(\log(T))$ regret, compared with algorithms with full information. Our trace-driven evaluations demonstrate that the overall cost can be reduced significantly as compared to existing strategies, and the regret due to online learning grows sublinearly with the time span.

VIII. acknowledgements

This work was supported in part by grants from Hong Kong RGC under the contracts HKU 718513, 17204715, 17225516, C7036-15G (CRF), a grant NSFC 61628209, and the Huawei Innovation Research Program (HIRP) HO2016050002BE. This research was also partially sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors 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 Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

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

[2] 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.

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

[4] 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.

[5] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding Data Center Traffic Characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[6] M. Welzl, *Network congestion control: managing internet traffic*. John Wiley & Sons, 2005.

[7] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

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

[9] M. Gahlawat and P. Sharma, "Survey of Virtual Machine Placement in Federated Clouds," in *Advance Computing Conference (IACC), 2014 IEEE International*. IEEE, 2014, pp. 735–738.

[10] 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.

[11] 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.

[12] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 2015, pp. 50–56.

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

[14] X. Wang, C. Wu, F. Le, A. Liu, Z. Li, and F. Lau, "Online VNF Scaling in Datacenters," in *Proceedings of the 9th IEEE International Conference on Cloud Computing (IEEE Cloud)*, 2016.

[15] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: a framework for nfv applications," in *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP)*, 2015.

[16] S. Bubeck and N. Cesa-Bianchi, "Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems," *arXiv preprint arXiv:1204.5721*, 2012.

[17] Y. Gai, B. Krishnamachari, and R. Jain, "Combinatorial Network Optimization with Unknown Variables: Multi-armed Bandits with Linear Rewards and Individual Observations," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 5, pp. 1466–1478, 2012.

[18] W. Chen, Y. Wang, and Y. Yuan, "Combinatorial multi-armed bandit: General framework and applications," in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 151–159.

[19] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[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] R. G. Parker and R. L. Rardin, "Guaranteed Performance Heuristics for the Bottleneck Travelling Salesman Problem," *Operations Research Letters*, vol. 2, no. 6, pp. 269–272, 1984.

[22] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The Nonstochastic Multiarmed Bandit Problem," *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2002.

[23] M. Zinkevich, "Online Convex Programming and Generalized Infinitesimal Gradient Ascent," 2003.

[24] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge university press, 2006.

[25] 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.

[26] 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.