Congestion Minimization for Service Chain Routing Problems With Path Length Considerations

Lingnan Gao, *Student Member, IEEE*, and George N. Rouskas, *Fellow, IEEE*

***Abstract*—Network function virtualization (NFV), with its perceived potential to accelerate service deployment and to introduce flexibility in service provisioning, has drawn a growing interest from industry and academia alike over the past few years. One of the key challenges in realizing NFV is the service chain routing problem, whereby traffic must be routed so as to traverse the various components of a network service that have been mapped onto the underlying network. In this work, we consider the online service chain routing problem. We route the service chain with the goal of jointly minimizing the maximum network congestion and the number of hops from the source to the destination. To this end, we present a simple yet effective online algorithm in which the routing decision is irrevocably made without prior knowledge of future requests. We prove that our algorithm is $O(\log m)$-competitive in terms of congestion minimization, where m is the number of edges of the underlying network topology, and we show that this ratio is asymptotically optimal.**

***Index Terms*—Network function virtualization, virtual network functions, NFV orchestration, online algorithm, resource allocation.**

I. Introduction

NETWORK function virtualization (NFV) [1] is an emerging networking paradigm that promises to ease the complexity of deploying new services into today's network. With the help of virtualization techniques, NFV relies on commercial off-the-shelf hardware to replace existing networking devices [2], thus separating the network functionality from the underlying network equipment and decoupling the service entity from the service location. Such a paradigm opens the door for network operators to implement both existing and future networking functions as software modules and consolidate them on general-purpose commodity servers based on demand. This approach simplifies the deployment process and introduces flexibility in service provisioning.

A virtual network function (VNF) represents a single functional block in an NFV environment. Each network service is composed of one or more VNFs, which is an implementation of a network function (NF). VNFs are realized (instantiated) by deploying them on virtual resources, such as virtual machines. The NFV management and orchestration (NFV-MANO) unit is responsible for the management of the VNFs [3], and maintains a database with data models and information for the NFs, virtual resources, and network services. The NFV-MANO unit utilizes this information to configure, orchestrate, and manage the life-cycle of the VNFs [4].

One key aspect in the management of requests for network services is the "service chain routing problem" in which the objective is to route user traffic along a path that starts at the source node, passes through the network locations where VNFs are implemented, and finally reaches the destination node. This problem becomes challenging when there are multiple VNFs for the same function available at distinct network nodes, and it is important to select one that aligns with the routing objectives. In an offline scenario, where all service requests are known in advance, service chain routing is NP-hard; this result follows from the fact that the unsplittable flow problem, which was proven to be NP-hard in [5], is a special case of the service chain routing problem. In practice, service chain routing is an online problem: service requests may arrive at arbitrary times and they must be placed onto the network without prior knowledge of future requests. These conditions pose additional challenges in developing effective and efficient algorithms for the online problem.

In this work, we focus on algorithm design for service chain routing in an online scenario, and we develop an algorithm to find a feasible routing of the service chain with respect to the service ordering constraints. The major objective is to improve the network performance by minimizing the maximum congestion. Different from our works in [6], we also consider minimizing the number of hops needed to route a service request, an important goal for reducing the end-to-end delay of a network service. We solve this problem using an efficient algorithm, based on the shortest path tour problem (SPTP) [7], [8], with a customized length function. Under a reasonable assumption that is satisfied in practice, i.e., that congestion does not result from a single request, we prove that this algorithm is $O(\log m)$-competitive, where m is the number of edges in the network graph. We further show that this competitive ratio is asymptotically optimal.

Following the introduction, in Section II we review the literature in this field. In Section III, we present the model for the network and service requests, and formally define the service chain routing problem we consider. In Section IV, we develop an online algorithm to minimize the maximum congestion and derive its competitive ratio, and we evaluate

Manuscript received March 26, 2019; revised January 11, 2020 and July 27, 2020; accepted August 3, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. Llorca. Date of publication August 26, 2020; date of current version December 16, 2020. *(Corresponding author: Lingnan Gao.)*

Lingnan Gao is with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206 USA (e-mail: glingna@ncsu.edu).

George N. Rouskas is with North Carolina State University, Raleigh, NC 27695-8206 USA, and also with the Department of Computer Science, King Abdulaziz University, Jeddah 21589, Saudi Arabia.

Digital Object Identifier 10.1109/TNET.2020.3017792

its performance in Section V. We conclude the paper in Section VI.

II. Related Work

In recent years, research efforts have been directed towards the service chain embedding problem, and a survey of resource allocation problems in NFV was presented in [9]. A majority of these works addressed an offline problem, where the aggregated requests need to be mapped onto the underlying network in one shot, under various embedding objectives and scenarios, as in [10]–[16]. In [10], [11], the authors considered the throughput maximization problem and proposed a randomized algorithm with performance guarantees, with applications to inter-datacenter networks and cellular networks. In [12], the authors considered the placement of VNFs and routing the traffic with a heuristic algorithm so as to minimize the expensive optical/electrical/optical conversions in datacenters. In [13], a heuristic algorithm based on game theory was proposed to place the virtual network functions and route the traffic to minimize operational cost. In [14], Barcelo *et al.* investigate the cloud service distribution problem that jointly determines the VNF placement and content distribution on a cloud platform. Their formulation as an integer linear programming problem aims at minimizing overall network operation cost while simultaneously satisfying user demand and capacity constraints. In [15], Feng *et al.* study a similar cloud service distribution problem and propose a fully polynomial time approximation scheme. In [16], the authors consider maximizing the total amount of VNF processed traffic under the computational capacity and budget constraints, and propose an algorithm with performance guarantees.

Apart from the works that addressed the offline service chain routing problem, there also exist several studies that consider the online case. In [17], a service chain orchestration routing strategy was proposed to route the traffic so as to satisfy the service ordering constraints. However, the underlying link load and capacity were not taken into account in that work. The work in [18] studied the mechanism design problem in the NFV market, and designed an online stochastic auction for on-demand NFV provisioning and pricing over a geo-distributed resource pool. References [19]–[22] focus on operational cost minimization for service provisioning. A forecast assisted provision scheme is proposed in [19] to predict future VNF requirements so as to pre-allocate VNFs at service maintenance time for a lower-provisioning cost. The work by Jia *et al.* [20] considers the need to dynamically scale the VNFs among geo-distributed datacenters to adapt to time-varying traffic volumes. They proposed an online algorithm based on regularization and dependent rounding to determine the total number and the placement of the running VNF instances to minimize total operational cost. In [21], the authors further extend the cloud service distribution problem in [14] to a dynamic environment and present a control algorithm for flow scheduling and resource allocation to adapt to changes in network conditions and service demands, with system throughput and cost optimality guarantees, while [22] further incorporates the impact of wireless channel into flow scheduling and resource allocations.

In [23], the authors propose algorithms to establish a multicast connection on a NFV-enabled networks aiming at minimizing the connection cost, and develop heuristic algorithms to handle the case where multicast endpoints dynamically join and leave the multicast session. In [24], the authors presented a multipath routing algorithm for online service provisioning, which was obtained by solving a linear programming problem, while in [25], the authors proposed an admission control scheme to admit and route online requests; the work of [26] studies the generalized flow problem with a mixture of unicast and multicast traffic, and provides a throughput-optimal algorithm, In this work, on the other hand, we consider the online problem of routing unicast traffic along a single path.

The studies most related to our work are [27]–[29]. The objective of these works are to map incoming network service requests to the physical network with finite capacity, and jointly consider the service chain embedding problem with admission control. In [27], the authors proposed an online algorithm that maximizes the number of admitted requests, under node capacity constraints, and has an $O(\log K)$ competitive ratio, where K is the number of network functions in the service chain. A "standby" mode was introduced in [28] to defer the acceptance of a request when sufficient resources are not available. Under this architecture, the authors proposed an online algorithm for the service chain embedding problem with the objective of maximizing the revenue with link capacity considerations. In [29], Jia *et al.* proposed an online algorithm to maximize the admitted number of requests, under a finite link capacity constraint. In our work, we tackle the problem from a different perspective, i.e., we assume that all services are admitted and our objective is to embed the service chain in a way that minimizes the maximum congestion.

III. Network Model and Problem Formulations

A. Network Model

We model the network as an undirected graph $G = (V, E)$, with $n = |V|$ number of vertices, and $m = |E|$ number of edges. Both the nodes and edges are capacitated, with c_u denoting the capacity of the node $u \in V$ and $c_{u,v}$ denoting the capacity of edge $(u, v) \in E$. For the ease of presentation, when the two endpoints of an edge are irrelevant, we denote the edge as $e \in E$, and its capacity as $c(e)$.[1] The network supports a set of K distinct network functions, $NF = \{NF_1, NF_2, \dots, NF_K\}$, and each network function NF_k is deployed (instantiated) at a subset of the network nodes. In addition, each network node may support an arbitrary number of NFs. We assume that the placement of NFs on network nodes is provided as input to the problem.

B. Service Chain Request

In an online scenario, service chain requests arrive in real time, whereby NFV-MANO has no information

[1]For ease of presentation, we assume an undirected graph with symmetric link capacity. Traffic in both directions share the capacity of the same link. However, the model may be extended to the asymmetric case by creating two directed links in the two directions such that traffic only consumes resources on the corresponding link.

TABLE I
NOTATION TABLE

symbol	meaning
G	graph representation of the SN
V	substrate nodes of SN
E	substrate links of SN
n	number of substrate nodes
m	number of substrate links
$\delta(v)$	neighbors of $v \in V$ on G
$c_{u,v}$	capacity for substrate link $(u, v) \in E$
c_u	capacity for substrate node u
K	number of different NF types
NF	collections of NFs supported by the SN
$\mathcal{C}_i$	service chain request i
src_i	source node of $\mathcal{C}_i$
dst_i	destination node of $\mathcal{C}_i$
$\mathcal{F}_i^k$	k-th NF specified by request $\mathcal{C}_i$
$V(\mathcal{F}_i^k)$	set of substrate nodes hosting $\mathcal{F}_i^k$
$d_i(k)$	resource request by segment k of $\mathcal{C}_i$
$u(e)$	link utilization of e
$l(e)$	amount of traffic on edge e
U_i	maximum congestion after $\mathcal{C}_i$ is placed
w_i	a feasible walk to route $\mathcal{C}_i$
W_i	complete set of valid walks to route $\mathcal{C}_i$
$tr_i(w, e)$	amount of traffic on e with $\mathcal{C}_i$ placed on w
$f_{u,v}^{i,k}$	1 if the traffic of k-th segment of $\mathcal{C}_i$ is routed on edge u, v
$h(w_i)$	number of hops to route $\mathcal{C}_i$ along w_i
L^*	maximum congestion of the offline optimal solution
λ	estimation for $L*$ via "doubling technique"
α	trade-off parameter for congestion agaist hop-number
γ	parameter used as the base for the length function
ω	length of the shortest walk for $\mathcal{C}_i$
$p_i(e)$	edge penalty function
ϕ_i	sum of penalties on all edges after $\mathcal{C}_i$ is routed
$len_i(e, k)$	length function of edge e for k-th segment of $\mathcal{C}_i$

about future requests when it makes the routing decision. We model a single service chain request $\mathcal{C}_i$ as a tuple of $\mathcal{C}_i = (src_i, dst_i, \mathcal{F}_i, d_i^n(k)|_{k=1,\ldots,k_i}, d_i^e(k)|_{k=0,\ldots,k_i})$, where src_i and dst_i are the source and destination nodes for the service chain; $\mathcal{F}_i = \left\{\mathcal{F}_i^1, \mathcal{F}_i^2, \ldots, \mathcal{F}_i^{k_i}\right\}$ represents k_i NFs that $\mathcal{C}_i$ requests, where $\mathcal{F}_i^k \in NF$, is a specific NF type that the traffic of $\mathcal{C}_i$ must traverse in the given order; $d_i^n(k)$ specifies the resource requirement for the k-th NF (namely, $\mathcal{F}_i^k$) on substrate node n; while $d_i^e(k)$ represents the amount of traffic between $\mathcal{F}_i^k$ and $\mathcal{F}_i^{k+1}$ (when $k = 0$, it presents the traffic from src_i to $\mathcal{F}_i^1$, and when $k = k_i$ it represents the traffic from $\mathcal{F}_i^{k_i}$ to dst_i). Similar to the work in [30], we assume that some network functions (e.g., an encoder or WAN optimizer) may have traffic changing effects, such that the amount of traffic coming out of the network function may be different than the amount of traffic that goes in. As a result, the value of $d_i^e(k)$ can be different with respect to k. We define a *segment* as the sub-path a packet traverses to reach one VNF, which includes the sub-paths from src_i to $\mathcal{F}_i^1$, $\mathcal{F}_i^j$ to $\mathcal{F}_i^{j+1}$, and $\mathcal{F}_i^{k_i}$ to dst_i. We respectively denote these sub-paths as the 0-th segment, j-th segment, and k_i-th segment. And we use $V(\mathcal{F}_i^k)$ to denote the set of substrate nodes that host $\mathcal{F}_i^k$.

In this work, we assume that requests are permanent, i.e., once a request arrives it will never terminate; extending the algorithm to the scenario whereby requests have a certain holding time after which they release resources and leave the network is the subject of ongoing research.

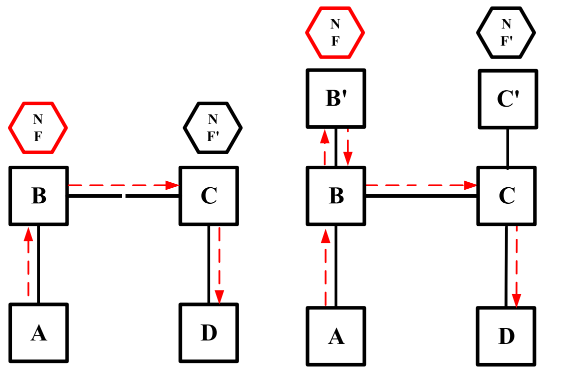

Fig. 1. Network traffic in the original network graph G (left) and the new graph G' with artificial nodes (right).

C. Unification of Node and Link Resources

To simplify the presentation and modeling process, we omit the physical node capacity and the VNF node processing requirements as they can be incorporated into the link capacities and traffic requirements, respectively. Such a transformation is carried out via the notion of augmenting graphs, whose appliance can be found in [14], [15], [22], [31], [32], by extending the original graph G with artificial nodes and links, and incorporate the computational resources to such links.

To demonstrate, consider Fig. 1. Given a substrate node $v \in V$ with node capacity c_v hosting a NFV NF, one can create an artificial node v' attached to v and move the NF to the artificial node v', as shown in the left part of Fig. 1. Notionally, the introduction of this artificial node separates the packet forwarding ability of the substrate node from the NFV processing power. When node v only forwards a packet, this packet will go through v, while a packet only reaches v' when it needs to be processed by NF. The link capacity of the link between the artificial node and the substrate node is $c_{vv'} = 2 \times c_v$, as a packet needs to traverse the edge (v, v') twice. We denote the graph with artificial nodes as $G' = (V', E')$.

The above process unifies resource capacities and demands for network links and nodes. For ease of presentation, in the subsequent paragraphs, we use $d_i(k)$ to uniformly denote the node and link demands and we use $c(e)$ to denote the capacities. Notice that this does not necessarily mean that the demand and capacity for the nodes are proportional to the traffic demand. Instead, such a difference is encoded within e: when $e \in E$, the demand and capacity stand for the traffic demand and link capacity, otherwise, for $e \in E' \setminus E$, the demand and capacity are for node demand and capacity, and we use network congestion to uniformly denote the maximum congestion on most congested links or nodes.

D. Problem Formulation

Service requests are routed in an online fashion, such that each request is routed without any information about the arrival time, traffic volume, or network functions of the future requests. The service chain will be routed in a way that each packet passes through the VNFs in a predefined order. We define the decision variable $f_{u,v}^{i,k}$ to represent if the k^{th} segment of $\mathcal{C}_i$ is routed on the edge (u, v). The objective is to route a new request $\mathcal{C}_i$ so as to minimize the maximum network congestion. We define the congestion metric after we route the request $\mathcal{C}_i$ as $U_i = max_{(u,v)\in E'} \sum_{j=0}^{i} \sum_{k=0}^{k=k_j} (f^{j,k}_{u,v} + f^{j,k}_{v,u}) d_j(k)/c_{u,v}$.

For the first i requests, based on the above definitions, we formulate the offline version of the congestion minimization problem as the following integer linear programming (ILP) problem.

$$minimize\ U_i \tag{1}$$

$$s.t. \sum_{u\in\delta(src)} f^{j,0}_{src_j,u} - \sum_{u\in\delta(src_j)} f^{j,0}_{u,src_j} = 1, \quad j \le i \tag{2}$$

$$\sum_{u\in\delta(dst_j)} f^{j,k_j}_{u,dst_j} - \sum_{u\in\delta(dst_j)} f^{j,k_j}_{dst,u} = 1, \quad j \le i \tag{3}$$

$$\sum_{v\in V(\mathcal{F}^k_j)} \sum_{u\in\delta(v)} f^{j,k}_{u,v} - \sum_{v\in V(\mathcal{F}^k_j)} \sum_{u\in\delta(v)} f^{j,k}_{v,u} = 1, \quad j \le i, \quad 1 \le k \le k_j \tag{4}$$

$$\sum_{u\in\delta(v)} f^{j,k}_{u,v} - \sum_{u\in\delta(v)} f^{j,k}_{v,u} = \sum_{u\in\delta(v)} f^{j,k+1}_{v,u} - \sum_{u\in\delta(v)} f^{j,k+1}_{u,v}, \quad j \le i, \quad 0 \le k < k_j, \ v \in V(\mathcal{F}^k_j) \tag{5}$$

$$\sum_{u\in\delta(v)} f^{j,k}_{u,v} - \sum_{u\in\delta(u)} f^{j,k}_{v,u} = 0, \quad v \notin V(\mathcal{F}^k_j), \quad j \le i, \quad 1 \le k < k_j \tag{6}$$

$$\sum_{j=0}^{i} \sum_{k=0}^{k_j} (f^{j,k}_{u,v} + f^{j,k}_{v,u}) d_j(k) \le U_i c_{u,v}, \quad j \le i, \quad (u, v) \in E' \tag{7}$$

$$f^{j,k}_{u,v} = \{0, 1\} \quad \forall j \le i, \ (u, v) \in E', \ k \le k_i \tag{8}$$

Expression (1) represents the objective of minimizing the maximum congestion at the time request $\mathcal{C}_i$ is routed. As we are solving an online problem, the same objective must have been applied to all earlier requests $\mathcal{C}_j, j \le i$.

Constraints (2) - (6) are the flow conservation constraints: Constraint (2) and Constraint (3), respectively, guarantee that the traffic originates from the source node and directs towards the destination node. Constraint (4) ensures that, for the k^{th} segment of the service chain, the traffic will be routed towards one of the VNFs in $\mathcal{F}_k$. Constraint (5) ensures that the flow in and out of a single VNF is consistent. Because the traffic will be processed by one of the VNFs in $\mathcal{F}_k$, when the traffic comes out of this VNF, it will switch to the next segment. In another words, the destination VNF in the previous segment needs to be the source of the next segment. Constraints (4) and (5) together ensure that the traffic will pass through the VNFs in a predefined order. Constraint (6) guarantees that the net flow in and out of a substrate node that does not host VNF remains zero. The flow conservation constraints, combined with Constraints (8) that enforce a binary value for the decision variable, ensure that all the traffic of one request will be routed along a single walk.

Lastly, Constraint (7) specifies that the total amount of traffic carried by any edge will not exceed the product of the edge capacity times U_i. Notice from the above discussion, when $(u, v) \in E$, $d_i(k)$ stands for the traffic requirement for k-th segment of request i whereas $c_{u,v}$ stands for link capacities, and the constraints here stand for resource constraints on network links; whereas $(u, v) \in E \setminus E'$, $d_i(k)$ and $c_{u,v}$, respectively, stand for the node demand and capacity, and Constraints (8) are bounds on node resource requirement and capacity. Consequently, by minimizing U_i in the objective function we minimize the maximum resource utilization.

For an online problem, the difference to the offline problem we consider is that information is revealed incrementally, and the decision is non-revertible. In other words, when routing service request $\mathcal{C}_i$, there is no information provided for future requests with $\mathcal{C}_j, j > i$, while all the decision variables are fixed for $f^{j,k}_{u,v}, j \le i-1$. In addition, the value for i is arbitrary, meaning that we intend to minimize the congestion at any point in time for all the incoming service chain requests.

IV. Online Routing Algorithm

In this section, we propose an online service chain routing algorithm. We design the algorithm under two different considerations: first, minimizing the maximum link utilization, in which it achieves an asymptotically optimal competitive ratio of $O(\log m)$[2]; second, reducing the number of hops for routing so as to lower the delay and packet loss rate. The algorithm is inspired by the virtual circuit routing problem [33] and routes each incoming request along the shortest walk under a customized length function. In the following, we first define a set of concepts used in developing the algorithm, and then provide the algorithm details and evaluate its performance.

A. Definitions

Valid Walk: Observe that when a service chain $\mathcal{C}_i$ is routed on a physical network, each segment of the traffic is routed along a path. This renders the routing of $\mathcal{C}_i$ as a walk on the graph G. We use W_i to denote the set of all valid walks for request $\mathcal{C}_i$, i.e., the walks that start at node src_i, traverse the required set of network functions in the given order, and terminate at node dst_i. We denote the walk selected by the routing algorithm for service request $\mathcal{C}_i$ as $w_i \in W_i$. If the request is routed along the walk w_i, then the amount of traffic along each edge e of the walk may be determined from quantities $d_i(k)$ of the request tuple. This is illustrated in Figure 2, where the service request shown at the top of the figure requires one unit of traffic from node A to the network function NF and from NF to node D. The dotted line on the bottom of Figure 2 shows a walk that represents a valid embedding of this service chain on the network topology, assuming that the network function is located at node E. From this walk, we determine that one unit of traffic goes through the edges (A, B) and (B, D), while two units of traffic are

[2]Competitive ratio measures the outcomes of an online algorithm by comparing to its optimal solution in hindsight. In our work, an $O(\log m)$-competitive ratio refers to the congestion achieved by our proposed algorithm *versus* the optimal offline congestion L^*. Notice that it is possible for the achieved congestion $U_i > 1$, implying a violating to the capacity constraints. Thus, one can interpret the $O(\log m)$ bound as either a) the congestion achieved by the proposed algorithm is at most $O(\log m)L^*$; or b) one needs to augment the existing resource by an $O(\log m)$ factor to match the performance in an offline scenario.

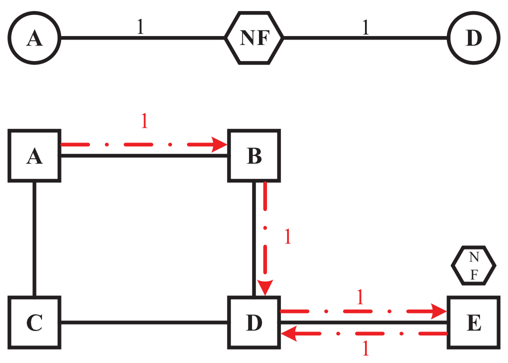

Fig. 2. A service chain request (top) and corresponding walk on the topology graph (bottom).

placed on edge (D, E). We use $tr_i(e, w)$ to denote the amount of traffic from request $\mathcal{C}_i$ that travels along edge e of walk w.

Penalty Function: The penalty function serves as an indicator for the congestion of an edge e. For each edge e, we define the penalty function after we route request $\mathcal{C}_i$ as[3]

$$p_i(e) = \gamma^{\frac{l_i(e)}{c(e)L^*}}, \tag{9}$$

where $\gamma \geq 1$ is a constant value, and L^* is the optimal maximum congestion of the network in hindsight, *i.e.* L^* is the optimal solution to U_i in an offline scenario where information for all the requests are known in advance, and $l_i(e)$ is the load on edge e after we route the first i requests, namely, $l_i(e) = \sum_{j=0}^{i} tr_j(e, w_j)$.

Potential Function: We use the potential function $\phi(i)$, defined as the sum of the edge penalty functions after we route the request $\mathcal{C}_i$,

$$\phi(i) = \sum_{e \in E'} p_i(e), \tag{10}$$

to capture the overall cost of placing the first i requests.

Shortest Path Tour: The shortest path tour problem (SPTP) [7], [8] has been studied extensively in the literature in various contexts. The input to the problem is a weighted graph G, source src and destination dst nodes, and multiple subsets of nodes $\{T_1, T_2, \ldots, T_K\}$. An algorithm for SPTP finds a walk (i.e., one or more edges may be traversed multiple times as part of the walk) from src to dst that visits at least one of the nodes in each set $T_k, 1 \leq k \leq K$, sequentially. Additionally, the algorithm must construct a walk whose weighted length is shortest among all valid walks.

We note that, assuming subset T_k represents the set of substrate nodes hosting $\mathcal{F}_i^k$, namely, $V(\mathcal{F}_i^k)$, then an (online) algorithm for SPTP will find a walk for routing the incoming service chain request $\mathcal{C}_i$. Therefore, our goal is to define a length function $len(e)$ for each edge such that the online SPTP algorithm will have a low competitive ratio with respect to congestion minimization. To this end, we first examine how congestion minimization is related to the potential function, and in turn how it translates into an appropriate length function $len(e)$ for SPTP.

[3]Here, $e \in E'$. Notice that the penalty function is link specific, $c(e)$ and $l(e)$ can either stand for link capacity and load when $e \in E$ or node capacity and load for $e \in E' \setminus E$. The same applies to the subsequent length function.

Our first observation is that the $\log$ value of the potential function, $\log(\phi(i))$, is an upper bound for the competitive ratio. Specifically, the potential function is the sum of all penalties, and therefore greater than any single penalty value. More formally:

$$\phi(i) = \sum_{e \in E} p_i(e) \geq \max_e \; p_i(e) \tag{11}$$

$$= \max_e \; \gamma^{\sum_{j=0}^{i} tr_j(e,w_j)/c(e)L^*}. \tag{12}$$

By taking the logarithm of both sides, and given the earlier definitions of L^*, it follows that the competitive ratio is bounded by $\log_\gamma(\phi(i))$.

Furthermore, notice that the value of $\phi(i-1)$ is constant since the routing of all previous service requests is given (i.e., it is fixed and may not change). Therefore, minimizing the increment of the potential function, $\phi(i) - \phi(i-1)$, is inherently equivalent to minimizing the potential function $\phi(i)$ for this online problem. Now, we also observe that when service request $\mathcal{C}_i$ is routed along the walk w_i, only the penalty function for the edges $e \in w_i$ will change. Thus, the increment to the potential function can be rewritten as:

$$\phi(i) - \phi(i-1) = \sum_{e \in w_i} (p_i(e) - p_{i-1}(e))$$

$$= \sum_{e \in w_i} \gamma^{\frac{l_{i-1}(e)}{c(e)L^*}} (\gamma^{\frac{tr_i(e,w_i)}{c(e)L^*}} - 1) \tag{13}$$

We conclude that minimization of the potential function reduces to finding the shortest valid walk on G whose length is defined by (13).

However, the penalty associated with walk w depends on both the number of times and the order in which w traverses an edge e. This is a crucial difference with SPTP, as the traffic is in the exponent of the second factor in (13), i.e., $\gamma^{\frac{tr_i(e,w)}{c(e)L^*}}$. In other words, if w traverses e multiple times, we cannot simply add the cost to obtain the penalty of the walk. This raises the question of what value to assign to each edge if we are to use an algorithm for SPTP to find the walk. In particular, we face a crucial dilemma: apparently, $tr_i(e, w)$, the amount of traffic we put on e, depends on the actual walk w, but we do not have knowledge of the walk until we find out the route.

Length Function: To address the above issue, we must define the length function so that it is independent of the walk selected for a service request. Specifically, we define the length function for edge e that is part of the walk for request $\mathcal{C}_i$ as:

$$len_i(e, k) = \gamma^{\frac{l_{i-1}(e)}{c(e)L^*}} (\gamma^{\frac{d_i(k)}{c(e)L^*}} - 1) \tag{14}$$

With this definition, the length function depends on the number k of segments of the walk (a value that is provided as input to the problem), as the amount of the traffic will change on different segments, but not on the specific walk selected.

Nevertheless, the length function (14) introduces another challenge. Specifically, a solution to SPTP (i.e., the shortest walk) using (14) as the length function may not be optimal under the function (13), since the length of the walk is inherently different. An example is shown in Figure 3, where we assume the new request is for one unit of traffic to be

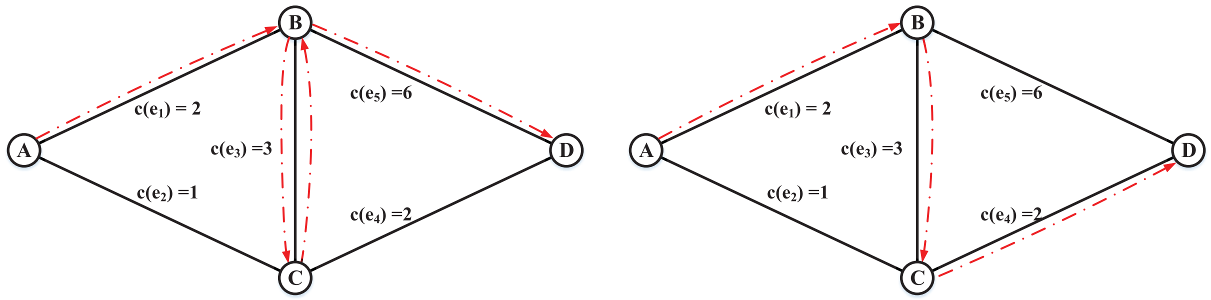

Fig. 3. Walks under the two length functions for the example of Section IV-A.

routed from source A to destination D after passing a network function located at node C and the capacity of each edge is as shown on Fig. 3. For this example, the utilization $\frac{l_{i-1}(e)}{c(e)L^*}$ is assumed the same for all edges e prior to routing this new request, and also we assume $L^* = 1$. In this case, the solution to SPTP under length function (14) is the walk shown with a dotted line on the left side, while the optimal walk that minimizes the increase in potential function of (13) is different and shown in the right side.

Let us now assume that the maximum congestion cannot be the result of any one request, i.e., $\frac{\sum_k d_i(k)}{c(e)} \leq L^*, \ \forall i, \ e$. This is a reasonable assumption that is satisfied in practice in the common scenario that the capacity of each edge is large compared to the traffic demand of any single request. Under this assumption, we can state the following result regarding the walk selected by an SPTP algorithm with length function (14):

Lemma 1: For a request $\mathcal{C}_i$, the shortest path tour w_i under the length function (14) is a γ-approximation to the optimal walk which minimizes (13).

Proof: The proof to this lemma is in the Appendix. □

B. Online Algorithm

We first describe below how to select a walk with fewer hops. We recognize that a non-zero constant term in a length function would encourage the selection of a walk with fewer number of hops. If we switch the length function from (14) to

$$\begin{aligned} len'_i(e,k) &= len_i(e,k) + \alpha * \rho \\ &= \gamma^{\frac{l_{i-1}(e)}{c(e)L^*}}(\gamma^{\frac{d_i(k)}{c(e)L^*}} - 1) + \alpha * \rho, \end{aligned} \tag{15}$$

where α and ρ are two positive constants (we shall discuss how to determine their values shortly) and run the SPTP algorithm, then we can expect a shorter walk than the one under (14). For the sake of brevity, we call the first term, $len_i(e,k)$, the *congestion term*, and the second, $\alpha * \rho$, the *constant term*. The reason we can expect a shorter walk is as follows. Suppose we have two different walks w_i and w'_i, with the number of hops being $h(w_i)$ and $h(w'_i)$, while $h(w_i) > h(w'_i)$. The constant terms in the length functions respectively contribute $h(w_i) * \alpha * \rho$ and $h(w'_i) * \alpha * \rho$ to the total length of the two walks. This would discourage the selection of the walk w_i so long as the difference in the congestion term is not significant enough.

It remains to be a problem on how to select the constant term for each edge. We assign the value in the following way: first, given a service request, we route the service chain along an SPTP under the length function (14). This results in a walk w'_i. The weighted length of w'_i is ω, and the number of hops is $h(w'_i)$. We select $\rho = \omega / h(w'_i)$ and have the following observation:

Lemma 2: The shortest path tour under the length function (15) is $(1+\alpha)\gamma$-approximation to the optimal walk that minimizes (13).

Proof: We denote the Shortest Path Tour under the length function (14) as w', and under length function (15) as w.

First, it is easy to verify that the weighted length of w under (15) is at most $(1+\alpha)\omega$. This is because the walk w' has the length of ω under the length function (14) and $(1+\alpha)\omega$ under (15). As w' is not necessarily the Shortest Path Tour, and we are running a Shortest Path Tour algorithm, the shortest walk we find under (15), namely, w, has a weighted length of at most $(1+\alpha)\omega$.

Second, notice that the constant term in length function (15) is positive, meaning that the length of w under (14) is strictly smaller than its length under (15). This implies that the length of w' is at most $(1+\alpha)\omega$ under length function (14). Following the result from Lemma 1, where ω is a γ-approximation solution to the weighted length, we can conclude the weighted length of w is at most $(1+\alpha)\gamma$-approximation to the optimal solution. □

We now explain how to circumvent the fact that the value of L^*, the optimal congestion in hindsight, is unknown. This follows from the *doubling techniques* in [33]: since we do not have any prior knowledge, we use λ as an estimate for L^*. Initially, we set λ set to be the minimum possible congestion for the first request, and we use this value in the length function (instead of the unknown L^*). For request $\mathcal{C}_i$, after we map it using the initial value of λ, we examine all the edges. If there exists an edge e such that

$$tr_i(e, w_i) + l_{i-1}(e) \geq \log_\gamma(\sigma m)\lambda, \quad \forall e \in w_i \tag{16}$$

where $m = |E|$ and $\sigma = \frac{1}{1-(1+\alpha)\gamma(\gamma-1)}$ is a constant number[4] determined by α and γ, then we double the value of λ and remap the request $\mathcal{C}_i$. This approach does not affect the asymptotic competitive ratio. The proof to the correctness of this approach is in [33].

With the proper concepts defined, our online algorithm is presented as Algorithm 1 below; we refer to this algorithm as

[4]Please refer to end of the Appendix for the definition of σ.

Algorithm 1 The OL-CompCL Algorithm for Online Service Chain Routing

```
Input:
    $l_{i-1}(e)$: existing load on edge $e$
    $C_i$: new service chain to be routed.
    $\lambda$: estimation for the optimal congestion in hindsight.
Output:
    $w_i$: selected route for service chain $i$
1: Construct a weighted undirected graph $G$ with edge length
       $len_i(e,k) = \gamma^{\frac{l_{i-1}(e)}{c(e)\lambda}}(\gamma^{\frac{d_i(k)}{c(e)\lambda}} - 1)$
2: Compute the shortest path tour $w_i'$ w.r.t $len_i(e,k)$ with
       length $\omega$ and the number of hops $h(w_i')$ of $w_i'$.
3: Re-compute the shortest path tour $w_i$ under length function,
       $len_i'(e,k) = \gamma^{\frac{l_{i-1}(e)}{c(e)\lambda}}(\gamma^{\frac{d_i(k)}{c(e)L^*}} - 1) + \alpha * \frac{\omega}{h(w_i')}$
4: if $\exists e \in w_i,\ l_{i-1}(e) + tr_i(e, w_i) \geq \lambda \log_\gamma(\sigma m)$ then
5:     $\lambda \leftarrow 2\lambda$, goto Step 1
6: end if
7: Update the load on each link.
```

"online-competitive-congestion-length" (*OL-CompCL*) algorithm. We first build a graph using the length function (14), based on the link load from previous requests, the demand of the new request, and the estimate for the optimal congestion λ. Then, we route the request using a shortest path tour algorithm to find the walk w_i' based on this graph. We obtain the weighted length for this Shortest Path Tour ω, and the number of hops for this tour $h(w_i')$. Using these two values, we re-compute the Shortest Path Tour under the length function (15), and obtain the tour w_i'. For all edges $e \in w_i$ along this walk, we examine if the inequality (16) holds. If so, this suggests that our estimate of L^* is low; in this case, we double the value of the estimate λ, recompute the length function, and reroute the request $\mathcal{C}_i$. Otherwise, we route the request along walk w_i and update the load of the corresponding edges.

C. Performance Analysis

1) Time Complexity: Building a weighted graph can be completed in $O(m)$ time, hence the time complexity of Algorithm 1 is dominated by finding the shortest path tour, which can be completed in $(Km\ \log n)$ time [17], where K is the number of network functions in the request (i.e., the number of segments of the walk).

Due to the potential underestimation of L^*, the SPTP algorithm may have to be run multiple times for a single request. For each request, this will happen at most $\log_2(KDC)$ times, where D is a ratio of the maximum to minimum traffic demand, and C is the ratio of the maximum to minimum edge capacity. This follows from the fact that the minimum value that λ takes is $\lambda = \frac{\min_{i,k} d_i(k)}{\max_e c(e)}$, while a maximum value in (16) is $\frac{\max_i \sum_k d_i(k)}{\min_e c(e)}$. As the estimate doubles each time, the number of estimates is upper bounded by $O(\log(KDC))$.

Thus, the overall time complexity of this online algorithm is in $O(Km\ \log n\ \log(KDC))$.

2) Competitive Ratio: We now prove that our algorithm achieves a competitive ratio that is asymptotically optimal. First, we prove that the growth of the potential function $\phi(i)$ is bounded.

Lemma 3: The potential function can be upper-bounded by $\phi(i) \leq \sigma m,\ \forall i$, where m is the number of edges of the network graph and σ a constant number, with a proper choice of α and γ.

Proof: The proof to this lemma is in the Appendix. □

Theorem 4: The competitive ratio of Algorithm 1 is $O(\log m)$, which is asymptotically optimal for congestion minimization.

Proof: First, we show that $O(\log m)$ is a lower bound on the competitive ratio. Observe that in the special case of $K = 0$, i.e., when the service does not request any network function between the source and destination nodes, the service chain routing problem reduces to the virtual circuit routing problem in [33], the optimal competitive ratio of which is $O(\log m)$. This suggests that $O(\log m)$ is the asymptotically optimal competitive ratio for Algorithm 1.

Next, we show that the competitive ratio achieved by our algorithm is $O(\log m)$. Combining inequality (12) and Lemma 2 and taking the logarithm on both sides, we obtain

$$max_{e\in E}\frac{l_i(e)}{c(e)L^*} \leq \log_\gamma(\sigma m) \tag{17}$$

which shows that the algorithm is $\log m$-competitive. □

V. Numerical Results

In this section, we present the numerical result to evaluate the performance of the online service chain routing algorithm.

A. Baseline Algorithms

We compare our *OL-CompCL* algorithm to the following four beaseline algorithms:

- *Offline-LP (OFF-LP):* The first approach is via solving the LP problem in hindsight. In an offline scenario, we have all the information on the service chain requests in advance. We relax the integral constraint (8) for the ILP in Section III, and solve the corresponding LP-problem. The solution to this LP-problem represents a lower bound for the maximal utilization minimization problem. As we have more information than an online scenario and the solution to the LP problem may be fractional, it would mean that this *Offline-LP* will only provide a better load balancing than any online algorithm who steers the traffic along a single-walk. Thus, the gap between our proposed algorithm, *OL-CompCL*, and *offline-LP* serves as an indicator of how well the algorithm performs against any possible online algorithms.
- *Online-SPTP (OL-SPTP):* The second baseline algorithm we use is another online algorithm via the SPTP. It routes the service chain on a Shortest Path Tour, with a load dependent length function. The length function we use is a piece-wise linear function proposed by [34] for OSPF routing that is crafted to minimize the network congestion

while discouraging a long detour, i.e.,

$$len(e) = \begin{cases} 1 & u(e) \in [0, \frac{1}{3}) \\ 3 & u(e) \in [\frac{1}{3}, \frac{2}{3}) \\ 10 & u(e) \in [\frac{2}{3}, \frac{9}{10}) \\ 70 & u(e) \in [\frac{9}{10}, 1) \\ 500 & u(e) \in [1, \frac{11}{10}) \\ 5000 & u(e) \in [\frac{11}{10}, +\infty), \end{cases}$$

where $u(e)$ is the link utilization at the time of routing the service chain requests. Upon arrival of each service chain request, we compute the length function based on the current load of each edge, and we route along the Shortest Path Tour accordingly.

- *Fewest-hop (FH):* This is identical to *online-SPTP* except for the length function. For all edges, we assign their length as 1. By computing the Shortest Path Tour under this value, we obtain a service chain routing which solely minimizes the number of hops from the source to the destination. This baseline algorithm serves as a lower-bound in terms of the hop number.
- *Online-competitive (OL-Comp):* This is the algorithm we presented in our work [6]. The difference between *OL-CompCL* and *OL-Comp* is that while both algorithms achieve an $O(logm)$-competitive ratio in congestion minimization, *OL-competitive* does not aim at reducing the number of hops.

B. Simulation Setup

We set up a set of simulations to evaluate the effectiveness of our proposed algorithm on two types of topology. First, we generate a graph following the Waxman model, which captures the topology of intra-domain networks [35]. Then we evaluate the performance of the proposed algorithm on a Fat-Tree [36], a topology that represents the design of data-center networks.

The topology generated by Waxman model consists of 50 nodes, and the capacity of each edge follows a uniform distribution in the range of $[50, 100]$. We define a total number of six different types of services supported by the physical networks. For each service, we randomly select ten physical network nodes, representing the locations where the VNFs are instantiated.

We use randomly generated service requests. For each service chain request i, we randomly select $k_i, 1 \leq k_i \leq K$ virtual functions, and these VNFs are concatenated in a random order. The traffic demand between each network function follows a uniform distribution of $[1, 5]$.

For the Fat-Tree topology [36] we use a 16-ary fat-tree consisting of 1024 hosts and 320 switches. Each link provides 10Gbps connectivity. In this case, we define 10 types of virtual network functions, with each type of service supported by 10 to 20 randomly instantiated hosts. The traffic

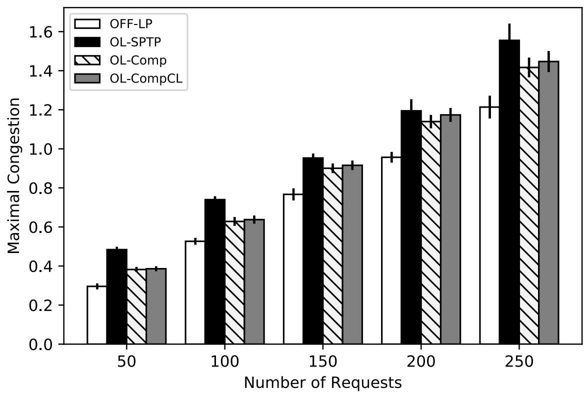

Fig. 4. Congestion *vs.* number of requests with $K = 4$.

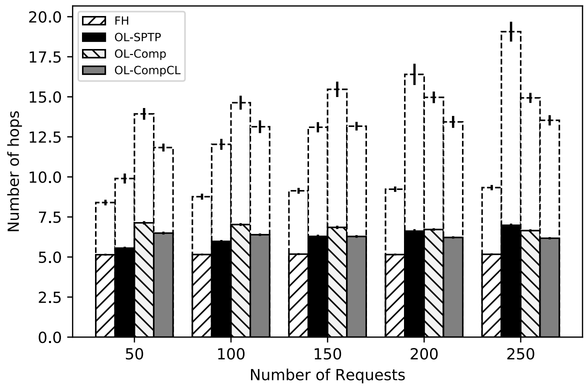

Fig. 5. Number of hops *vs.* number of requests with $K = 4$.

distribution of the service chain follows a uniform distribution of $[40Mbps, 200Mbps]$.

The metrics we evaluate include: 1) the maximum link utilization after we route the service requests; 2) the average and the maximal number of hops for the service chain to reach the destination; and 3) the running time for the routing algorithm. For each scenario, we run the simulation for 30 times to obtain the average value and the corresponding 95-percent confidence interval on those metrics. Because the *offline-LP* and the *fewest-hop* are algorithms baselines for the maximum congestion and the number of hops respectively, we omit their performance on other metrics. For our online routing algorithm, we set $\gamma = 1.4$, $\alpha = 0.5$, and $\sigma = 6.25$.

1) Waxman Topology: Fig. 4 to Fig. 9 plot the results for the Waxman topology. Fig. 4 plots the maximal congestion versus the total number of requests being routed on the substrate network, with the maximal length of the chain being $K = 4$. Compared to *offline-LP*, the lower-bound on the maximal congestion, the relative gap between the two is in the range of $(9.4\%, 23.3\%)$. Notice that while *offline-LP* is a lower-bound to the service chain routing problem, the bound is not tight. The reason is that *offline-LP* may return a fractional routing for the service chain, meaning the congestion will be even lower than the optimal value in hindsight. This implies that the gap between *online-competitive* algorithm and the optimal solution in hindsight is potentially smaller than the one indicated in Fig. 4. Compared to *online-SPTP* algorithm,

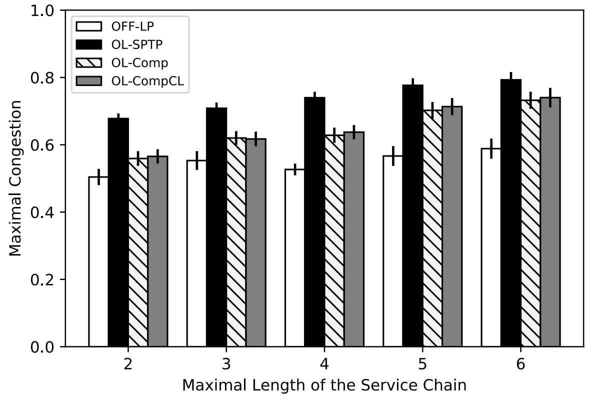

Fig. 6. Congestion *vs*. maximal chain length with $N = 100$.

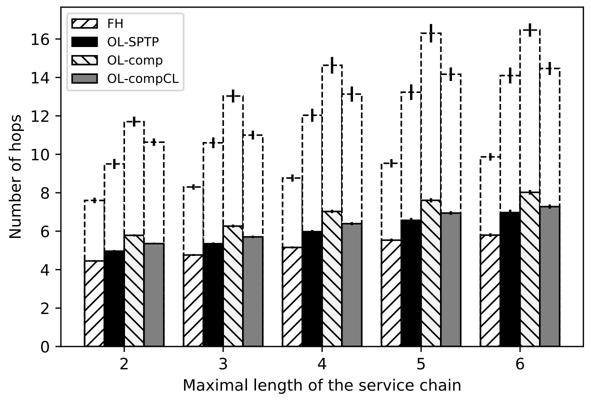

Fig. 7. Number of hops *vs*. maximal chain length with $N = 100$.

OL-CompCL delivers a better performance and minimizes the network congestion a ratio from 0% to 30%. On the other hand, compared to *OL-Comp*, this algorithm offers a similar performance, with the difference of the algorithm is within $\pm 2.5\%$ in all cases.

Fig. 5 plots the number of hops from the source to the destination. The dotted bars represent the maximal number of hops, while the solid ones represent the average value. For the average number of hops, in all cases, the proposed algorithm, *OL-CompCL*, routes the service chain with less than 1.6 additional hops (up to 30% in increase) against the lower-bound, i.e., *fewest-hop*. While compared to *online-SPTP*, we can make the following observations: the number of hops is affected by the number of service requests routed, which relates to the load on the substrate network. For *online-SPTP*, the number of hops increases with the load, while the other two online algorithms, see a drop in its value.

When the substrate network is under-utilized, *OL-CompCL* routes the service chain with as much as 1.16 (or 20.7%) extra number of hops on average; otherwise, it selects the route with 0.69 (or 9.9%) fewer hops. Compared to *competitive-online*, it minimizes the average number of hops between 0.5 to 0.6.

For the maximum number of hops, compared against *OL-SPTP*, *OL-CompCL* routes the request on a longer walk, with up to 3.0 extra hops on average when the load on the network is low, while it routes the request with 5.7 fewer hops when the load is heavy; compared against *OL-Comp*, the maximum number of hops sees an improvement of 1.3 to 2.7 hops.

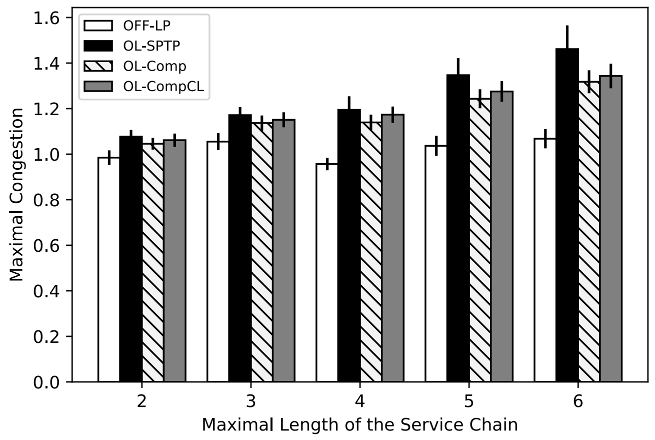

Fig. 8. Congestion *vs*. maximal chain length with $N = 200$.

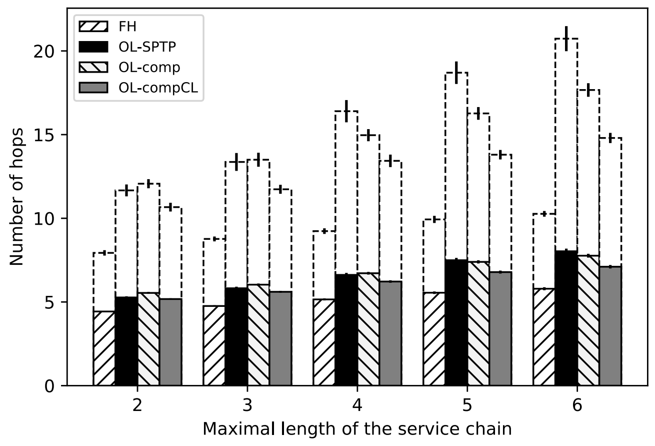

Fig. 9. Number of hops *vs*. maximal chain length with $N = 200$.

Fig. 11 plots how the value α impacts the tradeoff between the number of hops and the maximal congestion. In Fig. 11, the number of hops monotonically decreases with increasing α, from 7.8 hops when $\alpha = 0$ to 7.1 hops when $\alpha = 0.5$. Congestion, on the other hand, stays nearly the same, with utilization increasing less than 1% at $\alpha = 0.5$ than $\alpha = 0$.

We present the running time of the three online algorithms, namely, *OL-Comp*, *OL-CompCL*, and *OL-SPTP* in Fig. 10. We can conclude that *OL-SPTP* runs up to 30% faster than *OL-Comp*, due to a simpler length function, while *OL-CompCL* takes significantly more time than the two other algorithms. The increase in the running time comes from the need to compute the SPTP twice, which doubles the running time to route the service request.

Next, we look at how the maximal length of the service chain affects the performance. From the previous simulation results, we see that the load has a strong impact on the number of hops. Hence, we evaluate the performance of our algorithms under two different conditions: 1) a relatively low load, with $N = 100$ requests and 2) a relatively high load, with $N = 200$ requests.

Fig. 6 and Fig. 8 plot the congestion versus maximal service chain length, with a total number of $N = 100$ and $N = 200$

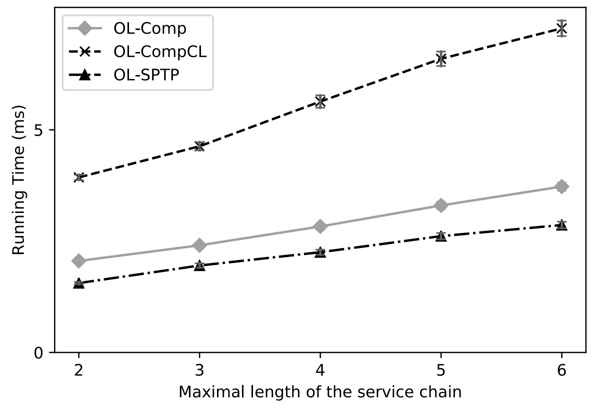

Fig. 10. Running time *vs.* maximal chain length K on Waxman topology.

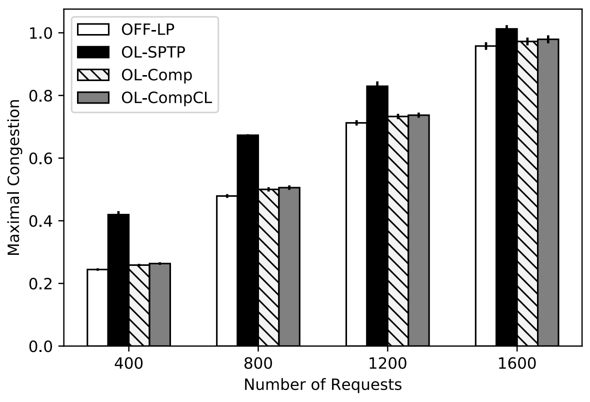

Fig. 12. Congestion *vs.* number of requests on Fat-Tree topology with $K = 4$.

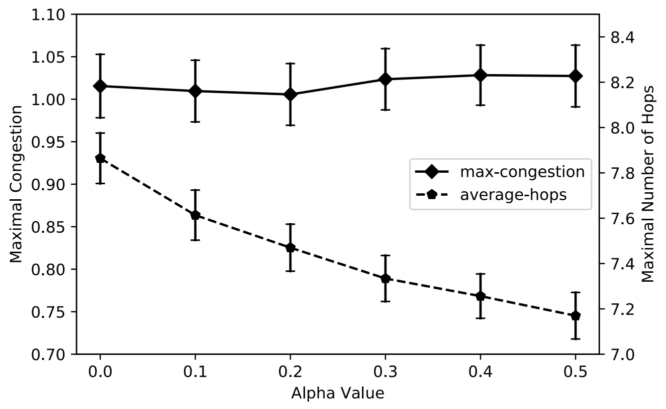

Fig. 11. Congestion and maximal chain length *vs.* value of α on Waxman topology with $N = 150$ and $K = 4$.

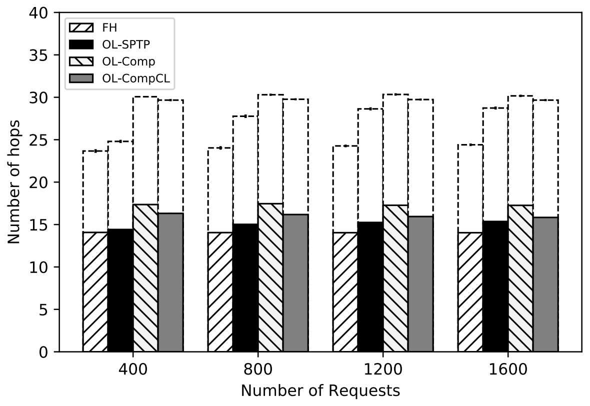

Fig. 13. Number of hops *vs.* number of requests on Fat-Tree topology with $K = 4$.

requests, respectively. In terms of the maximal congestion, the ratio between *offline-LP* and *OL-CompCL* is in a range of 1.17 to 1.27 when $N = 100$, and 1.06 to 1.21 when $N = 200$. Compared to *online-SPTP*, the *OL-CompCL* minimizes the congestion by 9% to 11% when $N = 100$ and by 3% to 15% with $N = 200$.

Fig. 7 and Fig. 9 plot the number of hops with respect to the maximal service chain length. We can observe that the load has the same impact on the number of hops on *OL-CompCL* algorithm as we see in Fig 5: it routes the service chain on the network with more number of hops when the load is lower. Compared to *FH*, *OL-CompCL* needs 1.1 to 1.7 extra hops on average (or 23% to 29% more hops) to route the service chain, when $N = 100$. Likewise, it needs 0.82 to 1.49 extra hops (or 18% to 25% more hops) when there are 200 hundred requests. Compared to *online-SPTP*, we see that our algorithm needs up to 0.57 additional hops with $N = 100$. With 200 service requests, it generally routes service requests with fewer hops. The number of hops levels up with *online-SPTP* when $K = 2$, while it can route with 0.7 fewer hops when $K = 6$.

In terms of the maximum number of hops, *OL-CompCL* can route the request with up to 2 fewer hops compared to *OL-Comp*, while compared to *online-SPTP*, this number ranges from 2 extra hops to 5 fewer hops depends on the maximal length of the service chain, and the load on the substrate networks.

2) Fat-Tree Topology: Fig. 12 plots the maximal congestion against the number of requests with maximal chain length $K = 4$. The maximal (denoted in dotted line) and average (in solid line) number of hops *versus* number of requests is plotted in Fig. 13. In terms of congestion, *OL-Comp* consistently delivers a better performance among all algorithms except for *OFF-LP*, where the congestion increase between *OL-Comp* and *OFF-LP* is less than 5.8%. *OL-CompCL*, on the other hand, sees a gap of 2.2% to 7.8% when compared to *OFF-LP*, with less than 1.8% increase when it compares to *OL-Comp*. Compared with *OL-SPTP*, *OL-CompCL* decreases maximal congestion between 3.7% and 32%. In terms of the number of hops, compared to *OL-Comp*, *OL-CompCL* requires 1.0 to 1.4 fewer hops (6.3% to 8.9%) on average, while it requires an additional 0.4 to 1.4 hops (i.e., 3.1% to 13%) and 1.8 to 2.2 (i.e., 12% to 15%) when compared to *OL-SPTP* and *FH* respectively. It also decreases the maximal hop number by 0.4 to 0.6 hops versus *OL-Comp*, while it needs 0.9 to 4.8 extra hops compared to *OL-SPTP* and 5.2 to 6.0 additional hops compared to *FH*.

Fig. 14 and Fig. 15, respectively, plot the maximal congestion and the average, maximal number of hops versus the maximal service chain length with $N = 800$. For maximal congestion, the gap between *OL-CompCL* and *OFF-LP* is within 6.3%, while it increases congestion by less than 1.1% when compared to *OL-Comp*. Compared to *OL-SPTP*,

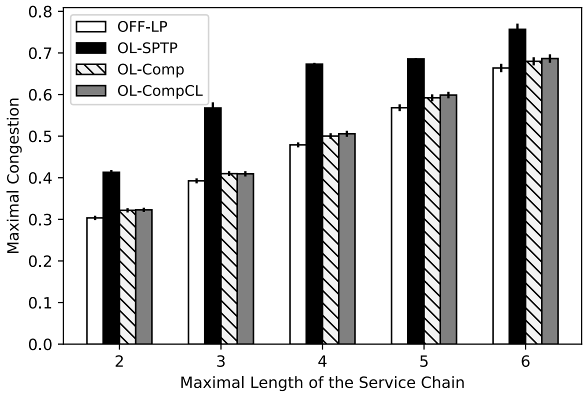

Fig. 14. Congestion *vs*. maximal chain length on Fat-Tree topology with $N = 800$.

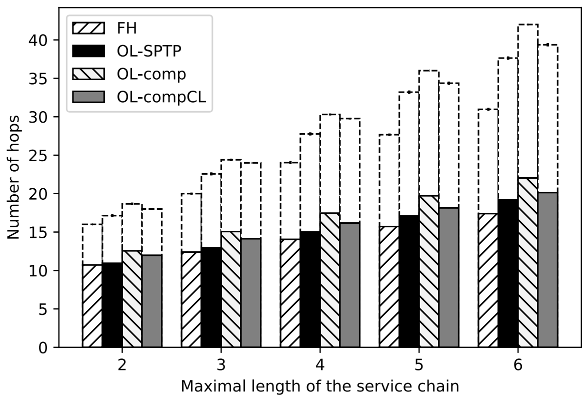

Fig. 15. Number of hops *vs*. maximal chain length on Fat-Tree topology with $N = 800$.

OL-Comp consistently improves congestion by 9.3% to 28%. For the number of hops to route a service chain, *OL-CompCL* reduces the average number of hops by 0.5 to 1.9 hops (4.6% to 8.7%) on top of *OL-Comp*, while it needs an extra 0.9-1.1 hops and 1.2-2.7 hops when compared to *OL-SPTP* and *FH*, respectively. Also, *OL-CompCL* reduces the maximal number of hops by 0.4 to 2.6 hops (1.7% to 6.3%) from *OL-Comp*, while the maximal number of hops increases by 0.86 to 2.0 hops (3.5% to 7.2%) from *OL-OSPF* and 2.0 to 8.4 hops (12% to 27%) from *FH*.

Fig. 16 shows how the factor α impacts the trade off between the maximal congestion and the average number of hops with $N = 800$ and $K = 6$. When α increases from 0 to 0.5, the average number of hops needed to route the service chain monotonically decreases, from 22.1 down to 20.2 hops; congestion, on the other hand, stays nearly the same (with $\pm 1.2\%$ variation), with a slight trend to increase along with α.

Fig. 17 plots the running time comparison of the three online algorithms as the value of K varies. Similar to the running time comparison on the Waxman topology, *OL-Comp* takes up to 30% percent additional time to route a service chain on average than *OL-SPTP*, while *OL-CompCL* needs twice as much time to run compared to *OL-Comp*.

To evaluate the ability of our proposed algorithm to scale to large datacenter networks, we measure the running time

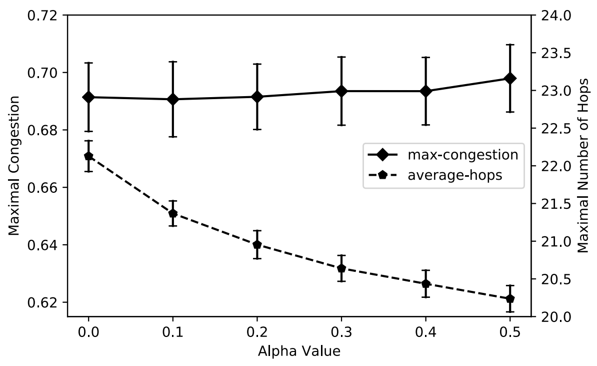

Fig. 16. Congestion and maximal chain length *vs*. value of α on Fat-Tree topology with $N = 800$ and $K = 4$.

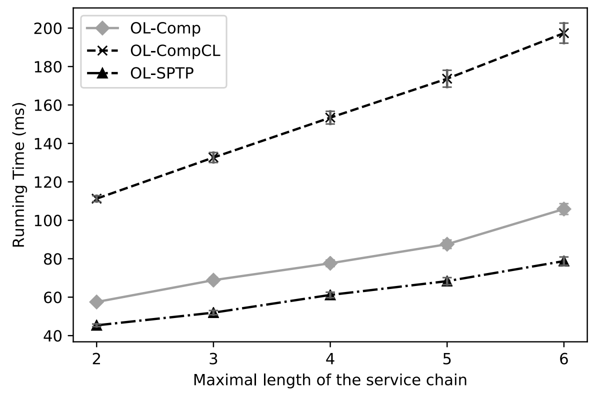

Fig. 17. Running time *vs*. maximal chain length K on Fat-Tree model.

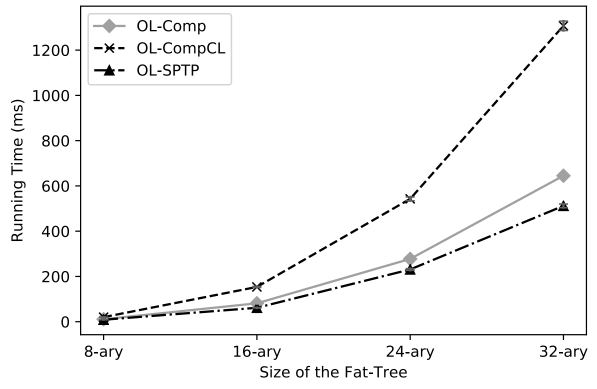

Fig. 18. Running time *vs*. size of Fat-Tree.

on Fat-Tree topologies of varying size. Namely, we evaluate the running time on a 8-ary, 16-ary, 24-ary and 32-ary Fat-Tree, with 128 (80), 1024 (320), 3456 (720), 8192 (1376) hosts (switches) and present the results in Fig. 18. Consistent with earlier results, *OL-CompCL* takes twice as much time to route a service chain compared to *OL-CompCl*, which in turn, takes up to 32% additional time than *OL-SPTP*. Importantly, however, on the large 32-ary Fat-Tree, OL-CompCL takes less than 1.4s on average to route a single request, indicating that our algorithm scales well to commercial datacenter networks.

From this set of simulation results, we conclude that our proposed algorithm is near-optimal in terms of minimizing the congestion, s demonstrated by the fact that *OL-CompCL* results in less than 27% (respectively, 7.8%) higher congestion against an offline lower-bound on Waxman (respectively, Fat-Tree) topologies, while it consistently delivers a better congestion than the baseline *SPTP-online* algorithm. As for the number of hops, the proposed algorithm generally incurs a small increase in the length of the route of a service chain, which results from the need to circumvent over-loaded links. Indeed, compared to *fewest-hop*, we can see that on average, with either the Waxman or the Fat-Tree topology, our proposed algorithm needs less than 30% percent additional hops to route a service chain. However, the minimization of the number of hops involves computing the SPTP twice. This generally indicates that the running time is twice as high compared to the two other algorithms.

VI. Concluding Remarks

We have developed an efficient online algorithm for the service chain routing in a NFV environment. The algorithm aims at jointly minimizing the network congestion and the number of hops. It achieves an optimal competitive ratio in minimizing the maximum utilization. Our work demonstrates that virtual networks may be operated effectively by routing online service chain requests along walks of near-optimal length (as shown in Lemma 1) that achieve near-optimal congestion (as Theorem 4 indicates). The focus of our current research efforts is to extend this algorithm to the case when (a) requests have a finite holding time after which they release resources and depart, and (b) the service chain is not necessarily a path.

Appendix

A. Proof to Lemma 1

From (13), we observe that the load on each edge is link-specific. For the sake of simplicity, we assume, without loss of generality, that all edges have capacity $c(e) = 1$. Also, for ease of presentation, we first denote as $len_{na}(e, w_i)$ the *non-additive* length function in (13), and as $len^w_{na}(w_i)$ the corresponding length of a walk under this non-additive edge length:

$$len_{na}(e, w_i) = \gamma^{\frac{l_{i-1}(e)}{L^*}}(\gamma^{\frac{tr_i(e,w_i)}{L^*}} - 1) \tag{18}$$

$$len^w_{na}(w_i) = \sum_{e \in w_i} len_{na}(e, w_i). \tag{19}$$

We also define an *additive* length for a walk:

$$len^w_a(w_i) = \sum_{k=0}^{k_i} \sum_{e \in s_k} len_i(e, k), \tag{20}$$

where the $len_i(e, k)$ is the length function defined in Section IV, and s_k is the path to route the traffic for segment s_k of $\mathcal{C}_i$. The total contribution of edge e to the walk is given by: $len_a(e, w_i) = \sum_{k: e \in s_k} len_i(e, k)$.

In order to prove the Lemma 1, we first prove the following lemma.

Lemma 5: For any walk w_i, the non-additive length of the walk is (a) bounded below by the additive length, and (b) bounded above by γ times the additive length, i.e.

$$len^w_a(w_i) \le len^w_{na}(w_i) \le \gamma len^w_a(w_i) \tag{21}$$

Proof: We have the following observation: for a walk w_i, the ratio of the non-additive cost to the additive cost is bounded below and above by the minimum and maximum ratio of each edge, respectively:

$$\min_{e \in w_i} \frac{len_{na}(e, w_i)}{len_a(e, w_i)} \le \frac{len^w_{na}(w_i)}{len^w_a(w_i)} \le \max_{e \in w_i} \frac{len_{na}(e, w_i)}{len_a(e, w_i)} \tag{22}$$

The two length function is different when and only when w traverses e multiple times. As w_i stands for the routing of the request, the traffic placed on edge e is the same, regardless of the penalty function. Without loss of generality, we assume that the walk traverses the edge for k times. Then, the ratio of the two functions is:

$$\begin{aligned}\frac{len_{na}(e, w_i)}{len_a(e, w_i)} &= \frac{\gamma^{\frac{l_{i-1}(e)}{L^*}}(\gamma^{\frac{tr_i(e,w_i)}{L^*}} - 1)}{\sum_k \gamma^{\frac{l_{i-1}(e)}{L^*}}(\gamma^{\frac{d_i(k)}{L^*}} - 1)} \\ &= \frac{\gamma^{\frac{\sum_k d_i(k)}{L^*}} - 1}{\sum_k (\gamma^{\frac{d_i(k)}{L^*}} - 1)}\end{aligned} \tag{23}$$

First, we prove that the non-additive cost is bounded below by the additive cost. Using the Taylor expansion of the exponential function, one may verify that $\gamma^{\sum_i x_i} - 1 \ge \sum_i (\gamma^{x_i} - 1)$ for all $\gamma \ge 1$ and $x_i \ge 0$, leading to the desired result:

$$\frac{len_{na}(e, w_i)}{len_a(e, w_i)} = \frac{\gamma^{\frac{\sum_k d_i(k)}{L^*}} - 1}{\sum_k (\gamma^{\frac{d_i(k)}{L^*}} - 1)} \ge 1 \tag{24}$$

Next, we prove that the non-additive cost is bounded above by γ times the additive cost. The difference between the two costs on any edge e is:

$$\frac{len_{na}(e, w_i)}{len_a(e, w_i)} = \frac{\gamma^{\frac{\sum_k d_i(k)}{L^*}} - 1}{\sum_k (\gamma^{\frac{d_i(k)}{L^*}} - 1)} \le \frac{(\gamma - 1)\frac{\sum_k d_i(k)}{L^*}}{\sum_k (\gamma^{\frac{d_i(k)}{L^*}} - 1)} \tag{25}$$

Inequality (25) holds due to two reasons: first, it is our assumption that a single request may not cause the most congestion, i.e., $\frac{\sum_k d_i(k)}{L^*} \le 1$; second, $\gamma^x - 1 \le (\gamma - 1)x$, for $0 \le x \le 1$ and $\gamma \ge 1$.

Using the Maclaurin series $\gamma^x = \sum_m \frac{(ln\gamma)^m}{m!} x^m$, one may verify that $\gamma^x - 1 \ge x ln\gamma$ for $\gamma \ge 1$, leading to the following inequality:

$$\begin{aligned}\frac{len_{na}(e, w_i)}{len_a(e, w_i)} &\le \frac{(\gamma - 1)\frac{\sum_k d_i(k)}{L^*}}{\sum_k (\gamma^{\frac{d_i(k)}{L^*}} - 1)} \le \frac{(\gamma - 1)\frac{\sum_k d_i(k)}{L^*}}{\sum_k ln\gamma \frac{d_i(k)}{L^*}} \quad (26) \\ &= \frac{\gamma - 1}{ln\gamma} \le \gamma \quad (27)\end{aligned}$$

The inequality (27) follows from the fact that $\frac{\gamma - 1}{ln\gamma} = \gamma(\frac{\gamma - 1}{\gamma ln\gamma})$, while $\frac{\gamma - 1}{\gamma ln\gamma}$ is a monotonically decreasing function with respect to γ, and $\lim_{\gamma \to 1} \frac{\gamma - 1}{\gamma ln\gamma} = 1$, meaning $\frac{\gamma - 1}{\gamma ln\gamma} \le 1$ when $\gamma \ge 1$, and in turn, leads to inequality (27).

Combining (24) with (27) we obtain the stated lower and upper bounds for the non-additive length of any walk. □

We are now ready to prove Lemma 1.

Proof: Denote the shortest valid walk with respect to the additive length $len^w_a(w)$ as w^a, and the optimal walk with respect to the non-additive length $len^w_{na}(w)$ as w^*.

Applying the lower and upper bounds of Lemma 4, we have the following two inequalities:

$$\frac{1}{\gamma} len^w_{na}(w^a) \leq len^w_a(w^a), \quad len^w_a(w^*) \leq len^w_{na}(w^*) \tag{28}$$

Since w^a is the shortest walk under the additive function, we have that:

$$len^w_a(w^a) \leq len^w_a(w^*) \tag{29}$$

Combining the last inequality with the two in (28), we obtain:

$$len^w_{na}(w^a) \leq \gamma \ len^w_{na}(w^*) \tag{30}$$

proving Lemma 1. □

B. Proof to the Lemma 3

Proof: Denote the shortest walk under (15) as w_i, and the optimal walk in hindsight as w^*_i.

Following the result of Lemma 2, the increase to the potential function is:

$$\phi(i) - \phi(i-1) = len^w_{na}(w_i) \leq (1+\alpha)\gamma \ len^w_{na}(w^*_i) \tag{31}$$

The sum of the differences above is given by:

$$\begin{aligned}
&\phi(i) - \phi(0) \\
&\quad = \sum_{j=1}^{i} (\phi(j) - \phi(j-1)) && (32)\\
&\quad \leq (1+\alpha)\gamma \sum_{j=0}^{i} len^w_{na}(w^*_j) && (33)\\
&\quad = (1+\alpha)\gamma \sum_{j=0}^{i} \sum_{e \in w^*_j} \gamma^{\frac{l_j(e)}{L^*}} (\gamma^{\frac{tr_j(e,w^*_j)}{L^*}} - 1) && (34)\\
&\quad \leq (1+\alpha)\gamma \sum_{j=0}^{i} \sum_{e \in w^*_j} \gamma^{l_j(e)} (\gamma - 1) tr_j(e, w^*_j)/L^* && (35)\\
&\quad = (1+\alpha)\gamma(\gamma-1) \sum_{j=0}^{i} \sum_{e \in w^*_j} \gamma^{l_j(e)} tr_j(e, w^*_j)/L^* && (36)\\
&\quad \leq (1+\alpha)\gamma(\gamma-1) \sum_{j=0}^{i} \sum_{e \in w^*_j} \gamma^{l_i(e)} tr_j(e, w^*_j)/L^* && (37)\\
&\quad = (1+\alpha)\gamma(\gamma-1) \sum_{e \in E'} \gamma^{l_i(e)} \sum_{j: e \in w^*_j} tr_j(e, w^*_j)/L^* && (38)\\
&\quad \leq (1+\alpha)\gamma(\gamma-1) \sum_{e \in w_i{}^*} \gamma^{l_i(e)} && (39)\\
&\quad = (1+\alpha)\gamma(\gamma-1)\phi(i) && (40)
\end{aligned}$$

Inequality (35) results from the fact that $l_i(e)$ is non-decreasing with respect to i, and $\gamma^x - 1 \leq (\gamma - 1)\, x$. Similarly, inequality (37) holds because $l_i(e)$ is non-decreasing. Inequality (39) holds as $\sum_{j: e \in w^*_j} tr_j(e, w^*_j)$ is the link load on e of the optimal solution in hindsight, while L^* is the maximum congestion across all edges hindsight, and therefore, $\frac{\sum_{j:e\in w^*_j} tr_j(e,w^*_j)}{L^*} \leq 1$, for the first i requests.

From inequality (37), we have $\phi(i) \leq \frac{\phi(0)}{1-(1+\alpha)\gamma(\gamma-1)}$. We denote $\sigma = \frac{1}{1-(1+\alpha)\gamma(\gamma-1)}$. For α and σ, one may select an appropriate value for $\alpha \geq 0$ and $\gamma \geq 1$ subject to $(1+\alpha)\gamma(\gamma-1) < 1$, namely $\sigma > 1$. □

References

[1] K. Joshi and T. Benson, "Network function virtualization," *IEEE Internet Comput.*, vol. 20, no. 6, pp. 7–9, 2016.

[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[3] "Gs NFV-man 001 v1. 1.1 network function virtualisation (NFV); management and orchestration," NFVISG ETSI, White Paper, 2014.

[4] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[5] B. Awerbuch, Y. Azar, and A. Epstein, "Large the price of routing unsplittable flow," in *Proc. thirty-seventh Annu. ACM Symp. Theory Comput.*, 2005, pp. 1–9.

[6] L. Gao and G. N. Rouskas, "On congestion minimization for service chain routing problems," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.

[7] P. Festa, "The shortest path tour problem: Problem definition, modeling, and optimization," in *Proc. INOC*, 2009, pp. 1–7.

[8] Festa, "Solving the shortest path tour problem," *Eur. J. Oper. Res.*, vol. 230, no. 3, pp. 464–474, 2013.

[9] J. Gil Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[10] J.-J. Kuo, S.-H. Shen, H.-Y. Kang, D.-N. Yang, M.-J. Tsai, and W.-T. Chen, "Service chain embedding with maximum flow in software defined network and application to the next-generation cellular network architecture," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[11] Z. Xu, W. Liang, A. Galis, Y. Ma, Q. Xia, and W. Xu, "Throughput optimization for admitting NFV-enabled requests in cloud networks," *Comput. Netw.*, vol. 143, pp. 15–29, Oct. 2018.

[12] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network function placement for NFV chaining in packet/optical datacenters," *J. Lightw. Technol.*, vol. 33, no. 8, pp. 1565–1570, Apr. 15, 2015.

[13] T. Lin, Z. Zhou, M. Tornatore, and B. Mukherjee, "Demand-aware network function placement," *J. Lightw. Technol.*, vol. 34, no. 11, pp. 2590–2600, Jun. 1, 2016.

[14] M. Barcelo, J. Llorca, A. M. Tulino, and N. Raman, "The cloud service distribution problem in distributed cloud networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 344–350.

[15] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the NFV service distribution problem," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[16] G. Sallam and B. Ji, "Joint placement and allocation of virtual network functions with budget and capacity constraints," in *Proc. IEEE Conf. Comput. Commun.*, Oct. 2019, pp. 523–531.

[17] S. Bhat and G. N. Rouskas, "Service-concatenation routing with applications to network functions virtualization," in *Proc. 26th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2017, pp. 1–9.

[18] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. M. Lau, "Online stochastic buy-sell mechanism for VNF chains in the NFV market," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 2, pp. 392–406, Feb. 2017.

[19] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted NFV service chain deployment based on affiliation-aware vNF placement," in *Proc. IEEE Global Commun. Conf.*, Dec. 2016, pp. 1–6.

[20] Y. Jia, C. Wu, Z. Li, F. Le, and A. Liu, "Online scaling of NFV service chains across geo-distributed datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 699–710, Apr. 2018.

[21] H. Feng, J. Llorca, A. M. Tulino, and A. F. Molisch, "Optimal dynamic cloud network control," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2118–2131, Oct. 2018.

[22] H. Feng, J. Llorca, A. M. Tulino, and A. F. Molisch, "Optimal control of wireless computing networks," *IEEE Trans. Wireless Commun.*, vol. 17, no. 12, pp. 8283–8298, Dec. 2018.

[23] S. Q. Zhang, Q. Zhang, H. Bannazadeh, and A. Leon-Garcia, "Routing algorithms for network function virtualization enabled multicast topology on SDN," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 4, pp. 580–594, Dec. 2015.

[24] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 1–9.

[25] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis, "Approximation and online algorithms for NFV-enabled multicasting in SDNs," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 625–634.

[26] J. Zhang, A. Sinha, J. Llorca, A. Tulino, and E. Modiano, "Optimal control of distributed computing networks with mixed-cast traffic flows," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 1880–1888.

[27] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *Proc. Int. Colloq. Structural Inf. Commun. Complex.*, 2015, pp. 104–118.

[28] G. Even, M. Rost, and S. Schmid, "An approximation algorithm for path computation and function placement in SDNs," in *Proc. Int. Colloq. Structural Inf. Commun. Complex.*, 2016, pp. 374–390.

[29] M. Jia, W. Liang, M. Huang, Z. Xu, and Y. Ma, "Throughput maximization of NFV-enabled unicasting in software-defined networks," in *Proc. IEEE Global Commun. Conf.*, Dec. 2017, pp. 1–6.

[30] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent NFV middleboxes," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[31] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[32] L. Gao and G. N. Rouskas, "Virtual network reconfiguration with load balancing and migration cost considerations," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 2303–2311.

[33] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, "On-line load balancing with applications to machine scheduling and virtual circuit routing," in *Proc. 25th Annu. ACM Symp. Theory Comput.*, 1993, pp. 623–631.

[34] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 4, pp. 756–767, May 2002.

[35] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, Oct. 1988.

[36] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Oct. 2008.

Lingnan Gao (Student Member, IEEE) received the B.E. degree in computer science from the Beijing University of Posts and Telecommunications in 2014 and the Ph.D. degree in computer science from North Carolina State University in 2019. His research interests include network virtualization and network design and optimization.

George N. Rouskas (Fellow, IEEE) received the bachelor's degree in computer engineering from the National Technical University of Athens and the M.S. and Ph.D. degrees in computer science from the Georgia Institute of Technology. He is currently an Alumni Distinguished Graduate Professor with the Department of Computer Science, North Carolina State University, where he has been serving as the Director of Graduate Programs for the department since January 2014. He has also been an invited Professor at King Abdulaziz University, Saudi Arabia, Paris VI University, France, and the University of Evry, France, on several occasions. His research interests include computer networking—specifically, optical networks, Internet architectures and protocols, network design and optimization, performance modeling, and scheduling.