On Congestion Minimization for Service Chain Routing Problems

Lingnan Gao[a], George N. Rouskas[a,b]
[a]North Carolina State University, Raleigh, NC 27695-8206 USA
[b]Department of Computer Science, King Abdulaziz University, Jeddah, Saudi Arabia

***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 with the goal of minimizing the maximum network congestion. 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, 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 objective is to improve network performance by minimizing the maximum congestion. We solve this problem using an efficient algorithm, based on the shortest path tour problem (SPTP) [6], [7], 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. We conclude the paper in Section V.

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 [8]. A majority of these works [9], [10], [11], [12] 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. In [9], [10], 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 [11], 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 [12], a heuristic algorithm based on game theory was proposed to place the virtual network functions and route the traffic to minimize operational cost.

Apart from the works that addressed the offline service chain routing problem, there also exist several studies that consider the online case. In [13], 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. In [14], the authors presented a multipath routing algorithm for online service provisioning, which was obtained by solving a linear programming problem, while in [15], the authors proposed an admission control scheme to admit and route online requests. In this work, on the other hand, we consider the online problem of routing unicast traffic along a single path.

The two studies most related to our work are [16], [17]. The objective of both is 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 [16], 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 [17] 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 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. The edges are capacitated, with $c(e)$ denoting the capacity of edge $e \in E$. The network supports a set of L distinct network functions, $NF = \{NF_1, NF_2, \ldots, NF_L\}$, and each network function NF_l is deployed (instantiated) at a subset $V_l \subseteq V$ of the network nodes. In addition, each network node may support an arbitrary number of the NFs. We assume that the placement of NFs on network nodes (i.e., the sets $V_l, l = 1, \ldots, L$) is provided as input to the problem.

B. Service Chain Request

We model the service chain request $\mathcal{C}_i$ as a tuple $\mathcal{C}_i = (src_i, dst_i, d_i(k), \mathcal{F}_i)$, where src_i and dst_i are the source and destination nodes for the service chain and $\mathcal{F}_i = \left\{f_i^{(1)}, f_i^{(2)}, \ldots, f_i^{(k_i)}\right\}$, $f_i^{(k)} \in NF$, is the set of network functions that the traffic of this request must traverse in the given order. We use k_i to represent the number of NFs in the service chain $\mathcal{C}_i$. Similar to the work in [18], 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 be different than the amount of traffic that went in. As a result, the amount of traffic on each segment of the path may vary, and we use $d_i(k)$ to represent the amount of traffic on the k-th segment.

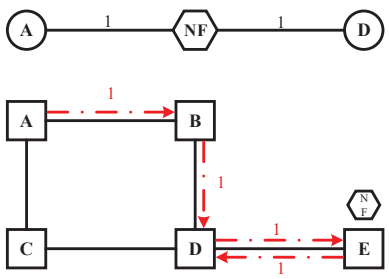

Fig. 1. A service chain request (top) and corresponding walk on the topology graph (bottom)

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.

C. 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 future requests. Observe that the route of a request is 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 1, 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 1 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 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.

Our objective is to route a new request $\mathcal{C}_i$ so as to minimize the maximum congestion. We define the congestion metric after we route the request $\mathcal{C}_i$ as $U_i = max_e \sum_j tr_j(e, w_j)/c(e)$, where $c(e)$ is the capacity of edge e.

Based on the above definitions, we formulate the online congestion minimization problem as the following integer linear programming problem (ILP), where the binary variable x_i^w indicates whether service request $\mathcal{C}_i$ is routed along walk $w \in W_i$.

$$minimize \quad U_R \tag{1}$$

$$s.t. \quad \sum_{i=0}^{R} \sum_{w \in W_i} x_i^w tr_i(e,w) \leq U_R c(e) \quad \forall e \in E \tag{2}$$

$$\sum_{w \in W_i} x_i^w = 1 \quad \forall i \leq R \tag{3}$$

$$x_i^w = \{0,1\} \quad \forall i \leq R,\ w \in W_i \tag{4}$$

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

Constraint (2) specifies that the total amount of traffic carried by any edge will not exceed the product of the edge capacity times U_R. Consequently, by minimizing U_R in the objective function we minimize the maximum edge congestion. Constraint (3) guarantees that all the traffic of request $\mathcal{C}_i$ will be routed along a valid walk. Combined with constraint (4) that enforces a binary value for the decision variable, the formulation ensures that all traffic of the request will be routed along a single walk.

The above walk-based formulation is compact and intuitive but it leads to a vast solution space. Specifically, there exists an exponential number of walks for each service request, resulting in an exponential number of decision variables. Therefore, the above problem of determining the route of a single request is computationally intractable to solve. However, we do not solve the ILP formulation directly. Instead, we build upon the shortest path tour concept to develop an efficient online algorithm, as we describe in the next section.

IV. Online Routing Algorithm

We propose an online service request routing algorithm which achieves an asymptotically optimal competitive ratio of $O(\log m)$. The algorithm is inspired by the virtual circuit routing problem [19] 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

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:

$$p_i(e) = \gamma^{\frac{l_i(e)}{c(e)L^*}}, \tag{5}$$

where L^* is the optimal congestion of the network in hindsight, γ is a constant (more details on L^* and γ shortly), and $l_i(e)$ is the load on edge e after we route the first i requests, namely, $l_i(e) = \sum_j tr_j(e, w_j)$.

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

$$\phi(i) = \sum_{e \in E} p_i(e), \tag{6}$$

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

Shortest path tour. The shortest path tour problem (SPTP) [6], [7] 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 subsets T_k represent the sets of nodes V_l where instances of each network function NF_l are placed, then an (online) algorithm for SPTP will find a walk for routing an incoming service chain request. 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.

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:

$$\phi(i) = \sum_{e \in E} p_i(e) \geq \max_e p_i(e) = \max_e \gamma^{\sum_i tr_i(e,w_i)/c(e)L^*}. \tag{7}$$

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

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:

$$\begin{aligned} \phi(i) - \phi(i-1) &= \sum_{e \in w_i} (p_i(e) - p_{i-1}(e)) \\ &= \sum_{e \in w_i} \gamma^{\frac{l_{i-1}(e)}{c(e)L^*}} (\gamma^{\frac{tr_i(e,w_i)}{c(e)L^*}} - 1) \end{aligned} \tag{8}$$

We conclude that minimization of the potential function reduces to finding the shortest valid walk on G whose length is defined by (8).

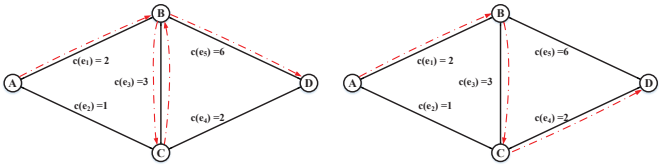

Fig. 2. Walks under the two length functions for the example of Section IV-A.

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 (8), 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:

$$len_i(e,k) = \gamma^{\frac{l_{i-1}(e)}{c(e)L^*}}(\gamma^{\frac{d_i(k)}{c(e)L^*}} - 1) \tag{9}$$

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 (9) introduces another challenge. Specifically, a solution to SPTP (i.e., the shortest walk) using (9) as the length function may not be optimal under the function (8), since the length of the walk is inherently different. An example is shown in Figure 2, where we assume the new request is for one unit of traffic to be 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. 2. 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 (9) is the walk shown with a dotted line on the left side, while the optimal walk that minimizes the increase in potential function of (8) 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 (9):

Lemma 1. *For a request $\mathcal{C}_i$, the shortest path tour w_i under the length function (9) is a γ-approximation to the optimal walk which minimizes (8).*

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

B. Online Algorithm

Before we describe our online algorithm, let us explain how to circumvent the fact that the value of L^*, the optimal congestion in hindsight, is unknown. 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

$$tr_i(e,w_i) + l_{i-i}(e) \geq \log_\gamma(4m)\lambda,\ \forall e \in w_i \tag{10}$$

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 [19].

With the proper concepts defined, our online algorithm is presented as Algorithm 1 below. We first build a graph using the length function (9), 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. For all edges $e \in w_i$ along this walk, we examine if the inequality (10) 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 [13], 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 $\log_2(KDC)$

Algorithm 1 Online service chain routing

Input:
$l_{i-1}(e)$: existing load on edge e
C_i: new service chain to be routed.
λ: 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 set according to (9), with λ in place of L^*.
2: Compute the shortest path tour w.
3: **if** $\exists e \in w,\ l_{i-1}(e) + tr_i(e,w) \geq \lambda \log_\gamma(4m)$ **then**
4: $\lambda \leftarrow 2\lambda$, **goto** Step 1
5: **end if**
6: Update the load on each link.

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 (10) 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 2. *The potential function is upper-bounded by $\phi(i) \leq 4m, \ \forall i$, where m is the number of edges of the network graph.*

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

Theorem 1. *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 [19], 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 (7) and Lemma 2 and taking the logarithm on both sides, we obtain

$$max_{e \in E} \frac{l_i(e)}{c(e)L^*} \leq \log_\gamma(4m) \tag{11}$$

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

V. Concluding Remarks

We have developed an efficient online algorithm for the service chain routing in a NFV environment. The algorithm aims at minimizing the network congestion and achieves an optimal competitive ratio. 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 1 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 (8), 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 (8), and as $len^w_{na}(w_i)$ the corresponding length of a walk under this non-additive edge length :

$$len_{na}(e, w_i) = \gamma^{\frac{l_{i-1}(e)}{L^*}}(\gamma^{\frac{tr_i(e,w_i)}{L^*}} - 1) \tag{12}$$

$$len^w_{na}(w_i) = \sum_{e \in w_i} len_{na}(e, w_i). \tag{13}$$

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

$$len^w_a(w_i) = \sum_{k=0}^{k_i} \sum_{e \in s_k} len_i(e, k), \tag{14}$$

where the $len_i(e, k)$ is the length function defined in Section IV. 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 3. *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.*

$$len^w_a(w_i) \leq len^w_{na}(w_i) \leq \gamma len^w_a(w_i) \tag{15}$$

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:

$$\min_{e \in w_i} \frac{len_{na}(e, w_i)}{len_a(e, w_i)} \leq \frac{len^w_{na}(w_i)}{len^w_a(w_i)} \leq \max_{e \in w_i} \frac{len_{na}(e, w_i)}{len_a(e, w_i)} \tag{16}$$

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 the first k times. Then, the ratio of the two functions is:

$$\begin{aligned} \frac{len_{na}(e, w_i)}{len_a(e, w_i)} &= \frac{\gamma^{\frac{l_{i-1}(e)}{L^*}}(\gamma^{\frac{tr_i(e,w_i)}{L^*}} - 1)}{\sum_k \gamma^{\frac{l_{i-1}(e)}{L^*}}(\gamma^{\frac{d_i(k)}{L^*}} - 1)} \\ &= \frac{\gamma^{\frac{\sum_k d_i(k)}{L^*}} - 1}{\sum_k (\gamma^{\frac{d_i(k)}{L^*}} - 1)} \end{aligned} \tag{17}$$

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 $\gamma^{\sum_i x_i} - 1 \geq \sum_i (\gamma^{x_i} - 1)$ for all $\gamma \geq 1$ and $x_i \geq 0$, leading to the desired result:

$$\frac{len_{na}(e, w_i)}{len_a(e, w_i)} = \frac{\gamma^{\frac{\sum_k d_i(k)}{L^*}} - 1}{\sum_k (\gamma^{\frac{d(k)}{L^*}} - 1)} \geq 1 \tag{18}$$

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:

$$\frac{len_{na}(e, w_i)}{len_a(e, w_i)} = \frac{\gamma^{\frac{\sum_k d_i(k)}{L^*}} - 1}{\sum_k (\gamma^{\frac{d_i(k)}{L^*}} - 1)} \leq \frac{(\gamma - 1)\frac{\sum_k d_i(k)}{L^*}}{\sum_k (\gamma^{\frac{d_i(k)}{L^*}} - 1)} \tag{19}$$

Inequality (19) holds due to two reasons: first, it is our assumption that a single request may not cause the most congestion, i.e., $\frac{\sum_k d_i(k)}{L^*} \leq 1$; second, $\gamma^x - 1 \leq (\gamma - 1)x$, for $0 \leq x \leq 1$ and $\gamma \geq 1$.

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

$$\frac{len_{na}(e, w_i)}{len_a(e, w_i)} \leq \frac{(\gamma - 1)\frac{\sum_k d_i(k)}{L^*}}{\sum_k (\gamma^{\frac{d(k)}{L^*}} - 1)} \leq \frac{(\gamma - 1)\frac{\sum_k d_i(k)}{L^*}}{\sum_k ln\gamma \frac{d(k)}{L^*}} \quad (20)$$

$$= \frac{\gamma - 1}{\gamma ln\gamma}\gamma \leq \gamma \quad (21)$$

The last inequality (21) stems from the fact that $\frac{\gamma-1}{\gamma ln\gamma}$ is a monotonically decreasing function with respect to γ, and $\lim_{\gamma \leftarrow 1} \frac{\gamma-1}{\gamma ln\gamma} = 1$.

Combining (18) with (21) 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 $len_a^w(w)$ as w^a, and the optimal walk with respect to the non-additive length $len_{na}^w(w)$ as w^*. Applying the lower and upper bounds of Lemma 4, we have the following two inequalities:

$$\frac{1}{\gamma} len_{na}^w(w^a) \leq len_a^w(w^a), \quad len_a^w(w^*) \leq len_{na}^w(w^*) \quad (22)$$

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

$$len_a^w(w^a) \leq len_a^w(w^*) \quad (23)$$

Combining the last inequality with the two in (22), we obtain:

$$len_{na}^w(w^a) \leq \gamma\ len_{na}^w(w^*) \quad (24)$$

proving Lemma 1. □

B. Proof to the Lemma 2

Proof. Denote the shortest path walk as w_i^a, and the optimal walk in hindsight as w_i^*. The increase to the potential function is:

$$\phi(i) - \phi(i-1) = len_{na}^w(w^a) \leq \gamma\ len_{na}^w(w^*) \quad (25)$$

The sum of the differences above is given by:

$$\phi(R) - \phi(0) = \sum_{i=1}^{R} (\phi(i) - \phi(i-1)) \leq \gamma len_{na}^w(w^*) \quad (26)$$

$$= \gamma \sum_{i=0}^{R} \sum_{e \in C_i^*} \gamma^{\frac{l_{i-1}(e)}{L^*}} (\gamma^{\frac{\sum_k d_i(k)}{L^*}} - 1) \quad (27)$$

$$\leq \gamma \sum_{i=0}^{R} \sum_{e \in C_i^*} \gamma^{l_R(e)} (\gamma - 1) \sum_k d_i(k)/L^* \quad (28)$$

$$= \gamma(\gamma - 1) \sum_{e \in C_i^*} \gamma^{l_R(e)} \sum_{i=0}^{R} \sum_k d_i(k)/L^* \quad (29)$$

$$\leq \gamma(\gamma - 1) \sum_{e \in C_i^*} \gamma^{l_R(e)} = \gamma(\gamma - 1)\phi(R) \quad (30)$$

Inequality (28) results from the fact that $l_i(e)$ is non-decreasing with respect to i, and $\gamma^x - 1 \leq (\gamma - 1)x$. Inequality (30) holds as $L^* \leq \sum_{i=0}^{R} \sum_k d_i(k, j)$ is the optimal congestion, in hindsight, for e for the first R requests.

By setting $\gamma = 3/2$, we have the desired result: $\phi(R) \leq 4\phi(0) = 4|E| = 4m$.

□

REFERENCES

[1] Kaustubh Joshi et al. Network function virtualization. *IEEE Internet Computing*, 2016.

[2] Bo Han et al. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 2015.

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

[4] Mijumbi et al. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 2016.

[5] Awerbuch et al. The price of routing unsplittable flow. In *Proceedings of STOC*. ACM, 2005.

[6] Paola Festa. The shortest path tour problem: problem definition, modeling, and optimization. In *Proceedings of INOC*, 2009.

[7] Festa et al. Solving the shortest path tour problem. *European Journal of Operational Research*, 2013.

[8] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in NFV: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[9] Jian-Jhih Kuo et al. Service chain embedding with maximum flow in software defined network and application to the next-generation cellular network architecture. In *Proceedings of INFOCOM*. IEEE, 2017.

[10] Zichuan Xu, Weifa Liang, Alex Galis, Yu Ma, Qiufen Xia, and Wenzheng Xu. Throughput optimization for admitting NFV-enabled requests in cloud networks. *Computer Networks*, 2018.

[11] Ming Xia, Meral Shirazipour, Ying Zhang, Howard Green, and Attila Takacs. Network function placement for NFV chaining in packet/optical datacenters. *Journal of Lightwave Technology*, 33(8):1565–1570, 2015.

[12] Tachun Lin, Zhili Zhou, Massimo Tornatore, and Biswanath Mukherjee. Demand-aware network function placement. *Journal of Lightwave Technology*, 34(11):2590–2600, 2016.

[13] Shireesh Bhat et al. Service-concatenation routing with applications to network functions virtualization. In *Proceedings of ICCCN*. IEEE, 2017.

[14] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. Adaptive vnf scaling and flow routing with proactive demand prediction. In *INFOCOM 2018-The 37th Annual IEEE International Conference on Computer Communications*, pages 1–9, 2018.

[15] Zichuan Xu, Weifa Liang, Meitian Huang, Mike Jia, Song Guo, and Alex Galis. Approximation and online algorithms for NFV-enabled multicasting in SDNs. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 625–634. IEEE, 2017.

[16] Tamás Lukovszki and Stefan Schmid. Online admission control and embedding of service chains. In *International Colloquium on Structural Information and Communication Complexity*, pages 104–118. Springer, 2015.

[17] Guy Even, Matthias Rost, and Stefan Schmid. An approximation algorithm for path computation and function placement in SDNs. In *International Colloquium on Structural Information and Communication Complexity*, pages 374–390. Springer, 2016.

[18] Wenrui Ma, Oscar Sandoval, Jonathan Beltran, Deng Pan, and Niki Pissinou. Traffic aware placement of interdependent NFV middleboxes. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2017.

[19] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 623–631. ACM, 1993.