Service Chain Rerouting for NFV Load Balancing

Lingnan Gao[a, *] , George N. Rouskas[b,c]
[a]Facebook, Menlo Park, CA, USA [b]NC State University, NC, USA, [c]King Abdulaziz University, Jeddah, Saudi Arabia

***Abstract*—Network function virtualization (NFV), with its potential to facilitate network service provisioning, has drawn growing interest from both academia and industry. One essential challenge is to allocate efficiently the bandwidth and computational resources to the service requests. In an online context, service chain requests may arrive, depart or evolve in an arbitrary fashion, adding more difficulty to the problem. Service chain reconfiguration may help improve the performance by individually rerouting a subset of the service chain requests. In this paper, we propose a new service chain reconfiguration framework to achieve load balancing in an NFV environment under varying levels of support from the underlying infrastructure. We show that our framework can achieve an approximation ratio of $O(\ln m/\ln\ln m)$ with high probability for the service chain request rerouting problem.**

I. Introduction

Network function virtualization (NFV) [1] is a networking paradigm that draws attention from the industry and academia alike. NFV aims to replace conventional middleboxes, most of which are implemented as dedicated hardware devices, with virtual network functions (VNFs). These dedicated hardware devices are typically expensive to acquire, difficult to configure, and hard to extend in terms of functionality [2]. Such inherent shortcomings pose challenges for service provisioning to keep up with the ever-changing user demands. With the help of virtualization techniques, network operators may instead implement the network functions as software and deploy them onto high capacity servers. This paradigm decouples the network functionality and services from the underlying physical infrastructure and introduces flexibility into the service provisioning process.

The most basic functional entity in an NFV environment is the VNF. Similar to conventional service provisioning where traffic may need to pass through one or more middleboxes in a predefined order, NFV service provisioning requires the concatenation of multiple VNFs hosted by the physical infrastructure. Each of the VNFs can accomplish a single task, such as firewall security, load balancing, etc., and multiple VNFs form a service chain to fulfill an overall service objective.

In an NFV environment, the NFV management and orchestration (NFV-MANO) unit manages the physical resources, allocates them in response to user demands, and is responsible for the configuration, orchestration, and life-cycle management of the VNFs [3]. One critical challenge in the allocation of resources is the online nature of demands: the arrival pattern of service chain requests may not be predictable, the amount or mix of resources requested by an existing service chain may change over time, and the lifetime of a service chain may be arbitrary. Consequently, NFV resource allocation cannot be viewed as a static problem, as the failure to include these dynamics may lead to inefficient configurations and degraded performance for the existing network services. Thus, there is a compelling need for the NFV-MANO module to incorporate the dynamic behavior of service chain requests into its resource allocation decisions. One option to address such dynamics is to incorporate the uncertainties into the decision making. Examples to such approaches can be found in [4–6]. These works aim to allocate resources to a given service chain with a provably good analytical bound, without using any information about future requests.

An alternative approach and the focus of this work, aims at improving performance via reconfiguration, or rerouting, of service chains. In particular, we consider two different types of support that the underlying infrastructure may provide: route optimization (RO) and state transfer (ST). With route optimization, one can set up any path on the substrate network from the source to the destination of a service chain, while with state transfer the chain may be handed from one VNF instance in the network to another. Recognizing that reconfiguration may be disruptive to ongoing services, our aim is to limit the number of rerouted service chains.

In this work, we focus on algorithm design for a suite of NFV service chain reconfiguration strategies. Our contributions include: (1) a new framework for modeling service chain rerouting that accounts for various levels of support from the infrastructure; (2) a formulation of the reconfiguration problem as a variant to the integer multi-commodity flow (MCF) problem with the objective of minimizing congestion while bounding the amount of disruption to users or the network; and (3) a fully polynomial time approximation scheme and a randomized rounding approach for the rerouting problem, and derivation of their optimality properties.

Service chain routing has been extensively studied by the existing literature [7]. While a major portion of these works tackle the problem in the offline setting [8–11], several studies have addressed the online routing problem [4–6, 12, 13] within various application scenarios. The works [4, 5, 13] consider the online admission control problem with the resource capacity constraints [4, 5], the end-to-end delay constraints [13], or for the multi-cast traffic [12], and present online algorithms that can jointly admit a subset of incoming requests to the physical infrastructure, while the work of [6] considers the online routing of the service chain request for a better load balancing. By resorting to the competitive analysis [14], these works can make the routing decisions without any information about the future requests, while achieving a provably good

* Work was done while at NC State University

performance. Although these online algorithms can achieve a good performance for static service chain requests, they do not consider the resource demand fluctuation, which may lead to performance degradation for the current configurations. Also, to the best of our knowledge, no previous work has considered the service chain rerouting problem under different levels of support from the underlying infrastructure to achieve a better load balancing.

Following the introduction, in Section II we formally state the service chain reconfiguration problem and in Section III we develop service chain rerouting algorithms. We evaluate the performance of the proposed algorithms in Section IV, and conclude the paper in Section V.

II. Problem Statement

As traffic demands evolve over time, contention for certain resources may lead to congestion. Our objective is to reconfigure service chains so as to eliminate hotspots within the network and improve performance. Specifically, we reroute traffic of an existing service chain individually and consider two different types of support from the underlying network:

- **Routing Optimization (RO):** The provider has fine-grained control in routing traffic over the underlying network. As a result, it is possible to set up any path for a given service chain to align with the optimization goal. Such control is possible, e.g., in software defined networks (SDN).
- **State Transfer (ST):** The provider may transfer the state of a network service from one VNF instance to another at a different location in the network. With ST support, it is possible to transfer the state from a VNF instance at node A to an instance of the same VNF at node B, and then reroute the service chain to node B.

As the underlying enetwork may independently provide the two types of support, RO and ST, we consider four cases:

- **RO**: In this scenario, the service chain must pass through the same set of VNF instances, but the network operator may reroute the traffic of the chain by optimizing the path segments between successive VNFs.
- **ST**: For each service chain, it is possible to select new VNFs of the same type and reroute the traffic to the new VNF instances. However, the paths between the new VNF instances are provided by the underlying network and are not subject to optimization.
- **RO & ST**: Under this scenario, when rerouting a service chain, the provider may optimize both the VNF instances and the path segments connecting them.
- **No RO or ST**: Service chain rerouting is impossible.

A. Modeling

1) Substrate Networks: To model the substrate network, we use an undirected graph $G = (V, E)$ with $n = |V|$ vertices, and $m = |E|$ edges. Each edge in G is capacitated, with the edge capacity $c(e),\ e \in E$, reflecting the amount of available bandwidth. There exist a total of K types of virtual network functions, $\{VNF_1, VNF_2, \dots, VNF_K\}$. Each type of VNFs may be instantiated on one or more substrate nodes, while each substrate node may support any number of VNFs.

2) Service Request: We use tuple $(src_i, dst_i, d_i, \mathcal{F}_i)$ to model service chain $\mathcal{C}_i$, where src_i and dst_i represent the chain's source and destination nodes, respectively, and d_i denotes the amount of its traffic. The traffic of chain $\mathcal{C}_i$ must go through one or more VNFs in the order they appear in the list$\mathcal{F}_i = f_i^1, f_i^2, \dots, f_i^k$, imposing a set of *ordering constraints* on the route of service chain $\mathcal{C}_i$.

3) Service Chain Embedding: As we are considering a reconfiguration problem, we assume that each active service chain $\mathcal{C}_i$ has been mapped onto the underlying network. In other words, a route from the source to the destination node of the service chain has been determined that satisfies the ordering constraints. We define a *segment* of a service chain as the path that interconnects: the source to VNF f_i^1, two successive VNFs in $\mathcal{F}_i$, or VNF f_i^k to the destination of the chain. Therefore, the route for service chain $\mathcal{C}_i$ is a sequence of paths. The implication is that the route may use an edge more than once, and we use the term *walk* to denote the route for $\mathcal{C}_i$. We use w_i^c to denote the walk on the substrate network to which service chain $\mathcal{C}_i$ has been mapped.

4) Valid Walk Set: We use W_i to represent the set of all valid walks for the service chain request $\mathcal{C}_i$. Upon completion of the rerouting process, the new route will be one of the walks in W_i, including w_i^c; in the latter case, the chain has not been selected for rerouting.

Given the existing route w_i^c for $\mathcal{C}_i$, the composition of W_i depends on the underlying network support, RO and/or ST:

1) **RO & ST:** W_i contains all the walks from src_i to dst_i in the underlying substrate network that satisfy the ordering constraints posed by $\mathcal{F}_i$.
2) **RO:** Suppose w_i^c passes through a sequence of substrate nodes that host the VNFs $f_i^1, f_i^2, \cdots, f_i^k$, in this order. Then, W_i contains all the walks from src_i to dst_i that pass through the same substrate nodes in the same order.
3) **ST:** W_i contains all the walks from src_i to dst_i in the underlying substrate network that satisfy the ordering constraints posed by $\mathcal{F}_i$, with the additional constraint that between any pair of substrate nodes in such a walk, there exists only one path determined by the routing policy (i.e., routing of path segments is not subject to optimization).

5) Valid Tree Set: Consider a VNF v instantiated on a certain substrate node. Whenever VNF v is part of a service chain, the substrate node where it is hosted is connected to a number of neighbors that could be: the source or destination of the chain, or the substrate nodes hosting v's neighbors in $\mathcal{F}_i$. Based on the above discussion, the substrate node of v is connected to each these neighboring nodes by a path. Since VNF v is the common endpoint of all these paths, we refer to this set of paths as a *tree* rooted at VNF v.

For a given tree t_v for the VNF v, the root of the tree, $root(t_v)$, is the substrate node where v is hosted. The leaves of the trees are substrate nodes, either hosting other VNFs, or representing the source or destination nodes. We use the notation $leaf(t_v, u)$ to denote the substrate node on tree t_v that hosts VNF u.

We use T_v to denote the set of trees that VNF v may use to route its traffic during a remapping operation. In the non-RO case, set T_v consists of one tree for each substrate node where VNF v is instantiated (since the routing is determined by policy and not subject to optimization). In the RO case, on the other hand, there could be multiple trees within T_v that use one substrate node as the root. A tree $t \in T_v$ represents both the placement of VNF v and the routing of its traffic.

B. Service Chain Rerouting Formulation

Based on the above definitions, we can formulate the service chain re-routing problem as an Integer Linear Programming (ILP) problem. We use the decision variable x_i^w to denote whether the service chain $\mathcal{C}_i$ is routed along walk w, and we assume that there is a total of R active chains.

$$\min \quad U \tag{1}$$

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

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

$$\sum_{i} \sum_{w \in W_i} \rho(w) x_i^w \leq M_C \tag{4}$$

$$x_i^w = \{0,1\} \quad \forall i \leq R,\ w \in W_i \tag{5}$$

Expression (1) captures the reconfiguration objective, which is to minimize the maximum utilization after carrying out the reconfiguration. Constraints (2) guarantee that the total amount of traffic on each edge may not exceed the edge capacity by more than a factor of U. In this expression, $tr_i(e,w)$ denotes the amount of traffic to be placed on edge e if chain $\mathcal{C}_i$ were to route along walk w. Hence, by minimizing U in the objective function, we are minimizing the maximum congestion.

Constraints (3) guarantee that all traffic of each chain $\mathcal{C}_i$ is routed along a valid walk. Combined with the constraints (5) they ensure that all traffic of a given chain is along a single walk. In (4), quantity $\rho(w)$ takes a constant value: $\rho(w) = 0$ when w is the original walk, i.e., $w = w_i^c$, otherwise, $\rho(w) = 1$. Therefore, Constraints (4) ensure that the number of rerouted chains does not exceed a given limit M_C.

We note that while the above ILP formulation is straightforward and intuitive, there could exist a huge number of decision variables. For a single service chain, the total number of valid walks is exponential to the size of the network. This makes it almost impossible to explicitly write down the exact ILP formulation. However, as we show next, we do not need the exact form of the ILP to solve the reconfiguration problem.

III. Service Chain Rerouting Algorithms

In this section, we present a general solution approach that may be used to tackle all three VNF rerouting problems we discussed in Section II. However, due to page constraints, we only discuss how to apply it to the RO case.

At a high-level, our approach consists of two steps:

1) **ILP Relaxation**: As we mentioned earlier, we cannot hope to solve the ILP formulation (1)-(5) within a reasonable amount of time. Thus, our first step is to relax the integral constraints (5). Although one can obtain the optimal solution to the resulting Linear Programming (LP) problem in polynomial time, we present a fully polynomial approximation scheme (FPTAS) to further expedite the solution process.
2) **Randomized Rounding:** The solution to the LP problem may yield fractional values for the decision variables, violating the constraint that each service chain be routed along one and only one walk. Starting with a feasible solution to the LP problem, we develop a randomized rounding method to construct the integral solution that represents a valid configuration for the network.

We discuss these steps in more detail next.

A. FPTAS Algorithm For ILP Relaxation

It is possible to approximate the solution to the LP problem derived from relaxing the integral constraints (5) via FPTAS, where one can control the trade-off between accuracy and efficiency. The FPTAS for the service chain rerouting problem is shown in pseudo-code as Algorithm 1. This FPTAS is a direct extension of the FPTAS for the Maximum Concurrent Multi-Commodity Flow (MCMCF) problem [15]. Before we present the FPTAS, we first restate the service chain rerouting problem to make it align with the MCMCF problem.

The ILP problem we presented in Section II aims at minimizing congestion, and is equivalent to the following formulation that maximizes throughput:

$$\max \quad \lambda \tag{6}$$

$$s.t. \quad \sum_{i=0}^{R} \sum_{w \in W_i} tr_i(e,w) x_i^w \leq c(e) \quad \forall e \in E \tag{7}$$

$$\sum_{w \in W_i} x_i^w \geq \lambda \quad \forall i \leq R \tag{8}$$

$$\sum_{i} \sum_{w \in W_i} \rho(w) x_i^w \leq \sigma M_C \tag{9}$$

The objective function (6) maximizes the minimum throughput for the service chains. The maximum throughput is reciprocal to the minimum congestion, i.e., $\lambda^* U^* = 1$. It is easy to verify that Constraints (7)-(9) relate to Constraints (2)-(4), respectively, when $\sigma = \lambda$. In the above formulation, σ is a constant, and we shall discuss shortly how to determine its value. We also note that the optimal solution to the throughput maximization problem may be transformed to one for the congestion minimization problem via scaling down the flows by a factor of λ^*.

The FPTAS is presented in Algorithm 1. It starts with an empty initial solution with throughput zero. For Algorithm 1 to approximate the solution within a factor of $1 + \omega$, we choose the value for ϵ so that $(1-\epsilon)^{-3} = 1 + \omega$. We then assign a positive initial weight to all edges *w.r.t.* the edge capacity and the desired accuracy level, namely, $l(e) = \delta/c(e)$, where $\delta = (m/(1-\epsilon))^{-1/\epsilon}$ and m is the number of edges in the network. In addition, there is a re-routing penalty $\phi = \delta/\sigma M$.

The algorithm then proceeds in phases, where each phase has multiple iterations and each iteration contains one or more steps. Within a single phase, for every service chain request, $\mathcal{C}_i,\ i \leq R$, we route d_i traffic and as a result the throughput increases by one. Note that, after each phase, the amount of traffic placed on an edge may well exceed its capacity. Nevertheless, we can obtain a feasible solution by scaling down the flows by the amount of violation.

Assuming there are R distinct active service chains, each phase will contain R iterations. Within iteration i, we route d_i units of traffic for chain $\mathcal{C}_i, i = 1\ldots, R$. Each iteration further consists of one or more steps, such that within each step we route the request along the shortest walk $w_i^s \in W_i$.

Recall that the set W_i of valid walks depends on the underlying network support, and that, due to page constraints, in this paper we only consider the RO case. How to find the shortest walk in this case is presented as function SHORTESTWALK. First, we find a walk w with the shortest length weighted by the demand $\sum_{e\in w} l(e)d_i$. In the RO case, one can only change the route within each segment, while the two endpoints of the segment remain the same. In other words, after rerouting, the service chain will still pass through the same set of VNFs. It is conceivable that, in this case, the shortest walk is the concatenation of the shortest path of each segment. For service chain $\mathcal{C}_i$, we can find the shortest walk by sequentially computing the shortest path between the $src_i, f_i^1, \ldots, f_i^k, dst_i$, and concatenate the shortest path to form the shortest walk. If this walk is different from w_i^c, the penalty ϕ shall apply. Thus, we shall use the walk w only if its length is less than that of w_i^c by a margin of ϕ.

In each step, when we route c amount of traffic along w_i^s, we ensure that the amount of the traffic will not exceed the capacity of the edges along this walk. The remaining traffic shall be routed in the succeeding steps. For edges along the walk, $e \in w_i^s$, we increase their weight by $l(e) = (1 + \epsilon c/c(e))l(e)$. Intuitively, a more highly loaded edge will have a greater weight, discouraging more traffic being placed on this edge in succeeding steps, and in turn, encouraging a more evenly distribution of traffic across all edges. Similarly, if we route $\mathcal{C}_i$ along a walk other than w_i^c, we increase the penalty $\phi = (1 + \epsilon c/\sigma M_C)$ to encourage the algorithm to pick the existing walk in future steps.

The FPTAS terminates when $D(l, \phi) = \sum_e c(e)l(e) + \sigma M\phi$, increases beyond 1. The value $D(l, \phi)$ is the dual objective for the throughput maximization problem, with $l(e)$ and ϕ being dual variables associates with Constraints (7) and (9) respectively.

Note that finding the shortest walk in the RO case needs to invoke Dijkstra's algorithm at most K times, once for each path segment, and the running time would be $O(Kmlogn)$. As with [15], we use $\tilde{O}(\cdot)$ to denote the running time that hides the polylog factors, and we have the following guarantee:

Algorithm 1 General FPTAS Framework

```
 1: function FPTAS($G, \{\mathcal{C}\}, \epsilon, \sigma$)
 2:     Initialization:
        $\delta \leftarrow (m/(1-\epsilon))^{-1/\epsilon}$, $l(e) \leftarrow \delta/c(e)$, $\phi \leftarrow \delta/\sigma M$
 3:     while $D(l, \phi) < 1$ do                              ▷ Phase
 4:         for $i = 1, 2, \cdots, R$ do                      ▷ Iteration
 5:             while less than $d_i$ traffic routed do       ▷ Step
 6:                 $w_s \leftarrow$ SHORTESTWALK($G, l, \phi, \mathcal{C}_i, w_i^c$)
 7:                 $c \leftarrow \min\{d_i', \text{routable traffic}\}$
 8:                 for all edges $e \in w_s$ do
 9:                     ship $c$ traffic along walk $w_s$
10:                     $x_i^{w_s} \leftarrow x_i^{w_s} + c$
11:                     $l(e) \leftarrow (1 + \epsilon c/c(e))l(e)$
12:                 end for
13:                 $\phi \leftarrow (1 + \rho(w_s)\epsilon c/\sigma M)\phi$
14:             end while
15:         end for
16:     end while
17:     Scale down flow to obtain a feasible $\hat{x}_i^w$ solution.
18: end function

19: function SHORTESTWALK($G, l, \phi, \mathcal{C}_i, w_i^c$)
20:     $w \leftarrow \{\}$
21:     for segment $s$ in $|SC_i|$ do
22:         $p \leftarrow$ shortest path for $s$
23:         $w \leftarrow w \cup p$
24:     end for
25:     if $\sum_{e\in w} l(e)d_i + \phi \leq \sum_{e\in w_i^c} l(e)$ then
26:         return $w$
27:     end if
28:     return $w_i^c$
29: end function
```

Theorem 1. *The FPTAS approximates the solution to the throughput maximization problem within $\tilde{O}(\omega^{-2}m^2RK)$ time, with an approximation ratio $O(1+\omega)$.*

Proof. The proof is similar to the one for the FPTAS in [15]. We omit the proof due to space constraints. □

The other two scenarios, namely, ST and ST & RO, differ from RO in the function SHORTESTWALK. We can find the shortest walk under these two scenarios with relatively simple modifications to the Viterbi algorithms [16]. Due to page constraints, we do not discuss these other two cases further.

B. On Randomization

The output of Algorithm 1 is a fractional solution to the Linear Programming problem. We denote this solution as $\hat{x}$. We shall now discuss how to recover an integer solution from the fractional solution with randomized rounding.

We assign a positive probability to each walk $w \in W_i$ that, in the fractional solution, carries non-zero traffic for chain $\mathcal{C}_i$. The probability value is proportional to the amount of traffic w carries, namely, $\textbf{Prob}(w, i) = \hat{x}_i^w/d_i$, where $\hat{x}_i^w$ is the total amount of traffic to route along the walk w. As the amount of

Algorithm 2 VNF Reconfiguration Framework

```
1: function ReconfigureVNF($G, \{\mathcal{C}\}, \epsilon$)
2:     Relax the integral constraints for the LP problem
3:     while bisection search for $\sigma$ until $\sigma = \lambda$ do
4:         $\lambda \leftarrow$ FPTAS($G, \{\mathcal{C}\}, \epsilon, \sigma$)
5:     end while
6:     for Service Chain $\mathcal{C}_i,\ \forall i \leq R$ do
7:         randomly select walk $w_i$ w.r.t. Prob$(w, i) = \hat{x}_i^w / d_i$
8:     end for
9: end function
```

flow is normalized by the total amount d_i of traffic for chain $\mathcal{C}_i$, it is easy to verify that $\sum_{w \in W_i} \mathbf{Prob}(w, i) = 1,\ \forall\ i$. Then, we randomly select one and only one walk for chain $\mathcal{C}_i$ based on this probability.

Randomization delivers the integral solution to the service chain rerouting problem. The whole procedure is summarized in Algorithm 2. Note that Algorithm 1 takes σ as an input. For a throughput maximization problem, the value σ should match the throughput λ. Although the value for λ is not known in advance, notice that λ will monotonically increase with σ. Thus, we can resort to a bisection search to find the value for σ: as the rerouting always leads to a better load balancing, the current throughput can serve as the lower bound for σ; meanwhile, the total capacity versus the total demand, $\frac{\sum_e c(e)}{\sum_i d_i}$, represents the most balanced load, thus it serves as an upper bound for the bisection search, which will terminate when σ matches the throughput.

Given a feasible solution to the linear programming problem $\hat{x}$ that achieves a congestion of $\hat{U}$ with $\hat{M}_C$ service chains being rerouted, we make the following two claims:

Lemma 2. *With high probability,* $1 - 1/m$*, randomized rounding with* $\mathbf{Prob}(w, i)$ *leads to an* $O(\hat{U} \ln m / \ln \ln m)$ *congestion.*

Proof. The analysis to the randomized rounding algorithm works is in the similar way to the analysis of the Integer Multi-Commodity Flow problem presented in [17], using the *Chernoff bound*. We omit the proof due to page limitation. □

Lemma 3. *The expected number of rerouted chains is* M_C*, and there is a strong tail distribution guarantee that the probability to have more than* M_C *requests re-routed exponentially decays. Namely, the probability that* $(1 + q)M_C$ *chains are rerouted is less than* $1 - e^{-\frac{qM_C}{3}}$ *with* $q \geq 1$.

Proof. The expected number service chain requests to be re-routed is equal to $\sum_i \rho(w) \hat{x}_i^w$. Constraints (5) ensures that it is no greater than M_C, thus the expected number of re-routed service chain is bounded by M_C.

We can take the rerouting of request $\mathcal{C}_i$ as a random variable $X_i(i)$, who takes the value of one when $\mathcal{C}_i$ is re-routed, and zero otherwise.

By taking the Chernoff bound [18] over the $X_i(i)$, we have:

$$\mathbf{Pr}[\sum_i X_i(i) \geq (1 + q)M_C] \leq e^{-\frac{q^2}{2+q} M_C} \qquad (10)$$

when $q \geq 1$, We have $e^{-\frac{q^2}{2+q} M_C} \leq e^{-\frac{qM_C}{3}}$. This implies that the probability to have more than $(1 + q)M_C$ service chain re-routed is less than $1 - e^{\frac{-qM_C}{3}}$ □

C. Performance Analysis

We make the following claims to the optimality and time complexity of the algorithm:

Theorem 4. *With a high probability, Algorithm 2 can achieve an* $O((1 + \omega) \ln m / \ln \ln m)$ *approximation ratio within* $\tilde{O}(\omega^{-2} m^2 KR)$ *under the RO case, with an expected number of* M_C *service chains rerouted.*

Proof. Approximation ratio: Suppose the optimal solution to the relaxed LP is U^*, the FPTAS yields a solution with congestion bounded by $(1+\omega)U^*$. Using the Lemma 2, after randomized rounding, we can obtain an $O((1 + \omega)U^* \ln m / \ln \ln m)$ congestion with high probability. Since U^* is a lower bound to the ILP problem, we conclude that Algorithm 2 achieves an $O((1+\omega) \ln m / \ln \ln m)$ probabilistic approximation ratio, and the expected number of service chain rerouted is M_C.

Time complexity: Algorithm 2 consists of two steps: an FPTAS phase and a rounding phase. Randomized rounding completes within linear time to the number of walks that carry positive traffic. The total number of such walks is proportional to the total number of steps in the FPTAS, therefore, the FPTAS phase, whose time complexity is given in Theorem 1, dominates the time complexity of Algorithm 2. □

IV. Numerical Results

We set up a simulation to evaluate our proposed algorithm. For the substrate network, we generate the topology following the Waxman model [19] with a total of 50 substrate nodes. The link capacities are uniformly distributed in $[50, 100]$. We define a total of six types of VNFs to be supported by the substrate network. Each type of VNFs is supported by ten substrate nodes and initially, the VNFs are randomly instantiated on the substrate nodes. We randomly generate the service chain requests. Each service chain request randomly chooses the source and the destination node, and the requested VNFs, and the length of the service chain is uniformly distributed in $[1, 4]$.

We use the tool [20] to generate the time-varying traffic of the service requests, which follows the distribution measured in [21]. The arrival of service chain requests follows a Poisson distribution. Since we do not have information about the future changes in the traffic demands, we use the online service chain routing algorithm [6] to route the service chain request based on the traffic demand of the service chain upon arrival. We carry out the reconfiguration operation after admitting a predetermined number of requests.

We compare the performance of our proposed algorithm, **rand-rr**, with the following baseline algorithms:

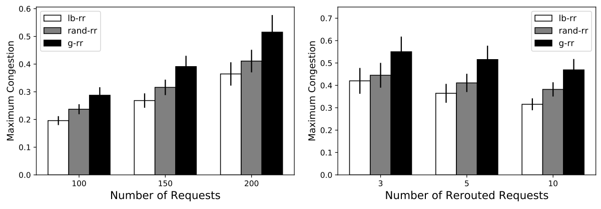

Fig. 1: congestion vs *num.* of requests

Fig. 2: congestion vs rerouted requests

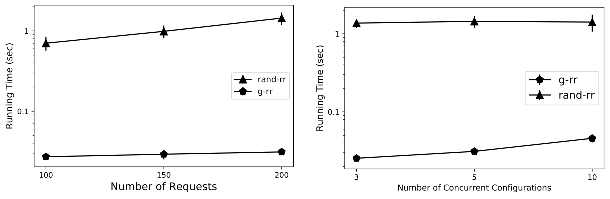

Fig. 3: running time vs *num.* of requests

Fig. 4: running time vs reconfigurations

- **reroute lower-bound (lb-rr)**: we use the solution from the FPTAS in Algorithm 2 to derive a lower bound to the congestion minimization. Suppose the FPTAS achieves a λ throughput, and an approximation ratio $1+\omega$, a lower-bound to the relaxed LP problem is $(1+\omega)/\lambda$, which in turn, is the lower-bound for the reconfiguration problem.
- **greedy reroute (g-rr)**: iteratively select and remove the service chain with the most heavy traffic passing through the most congestive edge, and reroute them back in the order they are selected. We also use the same routing algorithm [6] to reroute the selected service chain requests.

We present the simulation results in Fig. 1 to Fig. 4. We select ω to be 1, hereby, the FPTAS yields a 2-approximation solution to the LP problem in the worst case. We generate a varying number of service chain requests with the rerouting threshold M_C set to 5, and measure the running time and the maximum congestion. In addition, we generate a total of $R = 200$ service chain requests, and evaluate the proposed algorithm with different rerouting thresholds. For each scenario, we run the simulation for 30 times and present their mean and 95% confidence interval.

Fig. 1 plots the maximum congestion against the total number of service chain requests routed on the substrate network with $M_C = 5$, and Fig. 2 plots the maximal network congestion against different value for M_C, with $R = 200$. From these two figures, we can conclude that our proposed scheme, *rand-rr*, can achieve a good performance for the service chain re-routing. Compared to the case where no reconfiguration takes place (no-rcf), *rand-rr* reduces the congestion by at least 35% across all cases and the improvement increases with the threshold M_C; meanwhile it is off the theoretical lower bound, *lb-rr* by at most 21%. Compared to *g-rr*, *rand-rr* further reduces network congestion by 22%.

Fig. 3 plots the running time against different number of service chain requests routed on the network, while Fig. 4 evaluates the number of rerouted requests with different rerouting/remapping threshold. Our proposed scheme, *rand-rr*, takes at most one second across all scenarios, although the greedy algorithm takes is even faster.

V. Concluding Remarks

We have developed an efficient algorithm to perform reconfigurations to achieve load balancing in an NFC environment by re-routing service chains while taking the reconfiguration overhead into account. Our proposed algorithm can adapt to different levels of support from the underlying infrastructure and can achieve an $O(\frac{\ln m}{\ln m \ln m}(1+\omega))$-approximation ratio.

References

[1] B. Han et al. "Network function virtualization: Challenges and opportunities for innovations". In: *IEEE Communications Magazine* 53.2 (2015), pp. 90–97.

[2] J. Sherry et al. "A survey of enterprise middlebox deployments". In: (2012).

[3] R. Mijumbi et al. "Network function virtualization: State-of-the-art and research challenges". In: *IEEE Communications surveys & tutorials* 18.1 (2015), pp. 236–262.

[4] T. Lukovszki and S. Schmid. "Online admission control and embedding of service chains". In: *International Colloquium on Structural Information and Communication Complexity*. Springer. 2015, pp. 104–118.

[5] L. Guo et al. "Joint Placement and Routing of Network Function Chains in Data Centers". In: *INFOCOM*. IEEE. 2018.

[6] L. Gao and G. Rouskas. "On Congestion Minimization for Service Chain Routing Problems". In: *ICC*. IEEE. 2019.

[7] J. G. Herrera and J. F. Botero. "Resource allocation in NFV: A comprehensive survey". In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 518–532.

[8] R. Cohen et al. "Near optimal placement of virtual network functions". In: *INFOCOM*. IEEE. 2015.

[9] Z. Xu et al. "Throughput optimization for admitting NFV-enabled requests in cloud networks". In: *Computer Networks* 143 (2018), pp. 15–29.

[10] Y. Sang et al. "Provably efficient algorithms for joint placement and allocation of virtual network functions". In: *INFOCOM*. IEEE. 2017.

[11] W. Ma et al. "Traffic aware placement of interdependent nfv middleboxes". In: *INFOCOM*. IEEE. 2017.

[12] Z. Xu et al. "Approximation and online algorithms for NFV-enabled multicasting in SDNs". In: *ICDCS*. IEEE. 2017.

[13] Y. Ma et al. "Online revenue maximization in NFV-enabled SDNs". In: *ICC*. IEEE. 2018, pp. 1–7.

[14] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

[15] N. Garg and J. Koenemann. "Faster and simpler algorithms for multicommodity flow and other fractional packing problems". In: *SIAM Journal on Computing* 37.2 (2007), pp. 630–652.

[16] G. D. Forney. "The viterbi algorithm". In: *Proceedings of the IEEE* 61.3 (1973), pp. 268–278.

[17] P. Raghavan and C. D. Tompson. "Randomized rounding: a technique for provably good algorithms and algorithmic proofs". In: *Combinatorica* 7.4 (1987), pp. 365–374.

[18] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[19] B. M. Waxman. "Routing of multipoint connections". In: *IEEE journal on selected areas in communications* (1988).

[20] L. Saino et al. "A toolchain for simplifying network simulation setup." In: *SimuTools* 13 (2013), pp. 82–91.

[21] A. Nucci et al. "The problem of synthetically generating IP traffic matrices: Initial recommendations". In: *SIGCOMM Computer Communication Review* (2005).