Online Admission Control and Embedding of Service Chains[⋆]

Tamás Lukovszki[1] and Stefan Schmid[2]

[1]Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary
`lukovszki@inf.elte.hu`
[2]TU Berlin & Telekom Innovation Laboratories, Berlin, Germany
`stefan.schmid@tu-berlin.de`

Abstract. The virtualization and softwarization of modern computer networks enables the definition and fast deployment of novel network services called *service chains*: sequences of virtualized network functions (e.g., firewalls, caches, traffic optimizers) through which traffic is routed between source and destination. This paper attends to the problem of admitting and embedding a maximum number of service chains, i.e., a maximum number of source-destination pairs which are routed via a sequence of ℓ to-be-allocated, capacitated network functions. We consider an Online variant of this maximum Service Chain Embedding Problem, short *OSCEP*, where requests arrive over time, in a worst-case manner. Our main contribution is a deterministic $O(\log \ell)$-competitive online algorithm, under the assumption that capacities are at least logarithmic in ℓ. We show that this is asymptotically optimal within the class of deterministic and randomized online algorithms. We also explore lower bounds for offline approximation algorithms, and prove that the offline problem is APX-hard for unit capacities and small $\ell \geq 3$, and even Poly-APX-hard in general, when there is no bound on ℓ. These approximation lower bounds may be of independent interest, as they also extend to other problems such as Virtual Circuit Routing. Finally, we present an exact algorithm based on 0-1 programming, implying that the general offline SCEP is in NP and, by the above hardness results, it is NP-complete for constant ℓ.

Keywords: Computer Networks, Network Virtualization, Virtual Circuit Routing, Online Call Admission, Competitive Analysis

1 Introduction

Today's computer networks provide a rich set of in-network functions, including access control, firewall, intrusion detection, network address translation, traffic shaping and optimization, caching, among many more. While such functionality

[⋆] Supported by the FP7 EU project UNIFY and the DFG SFB 901 project: Part of this research was done when the first author visited the Heinz Nixdorf Institute, Germany.

is traditionally implemented in hardware middleboxes, computer networks become more and more virtualized [12, 24]: *Network Function Virtualization (NFV)* enables a flexible instantiation of network functions on network nodes, e.g., running in a virtual machine on a commodity x86 server.

Modern computer networks also offer new flexibilities in terms of how traffic can be routed through such network functions. In particular, using *Software-Defined Networking (SDN)* [19] technology, traffic can be steered along arbitrary routes, i.e., along routes which depend on the application [13], and which are not necessarily shortest paths or destination-based, or not even loop-free [11].

These trends enable the realization of interesting new in-network communication services called *service chains* [8, 14, 25, 26]: sequences of network functions which are allocated and stitched together in a flexible manner. For example, a service chain c_i could define that traffic originating at source s_i is first steered through an intrusion detection system for security (1^{st} network function), next through a traffic optimizer (2^{nd} network function), and only then is routed towards the destination t_i. Such advanced network services open an interesting new market for Internet Service Providers, which can become "miniature cloud providers" [27], specialized for in-network processing.

1.1 Paper Scope

In this paper, we study the problem of how to optimally admit and embed service chain requests. Given a redundant distribution of network functions and a sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_k)$, where each $\sigma_i = (s_i, t_i)$ for $i \in [1, k]$ defines a source-destination pair (s_i, t_i) which needs to be routed via a sequence of network function instances, we ask: Which requests σ_i to admit and where to allocate their service chains c_i? The service chain embedding should respect capacity constraints as well as constraints on the length (or stretch) of the route from s_i to t_i via its service chain c_i.

Our objective is to maximize the number of admitted requests. We are particularly interested in the *Online Service Chain Embedding Problem (OSCEP)*, where σ is only revealed over time. We assume that a request cannot be delayed and once admitted, cannot be preempted again. Sometimes, we are also interested in the general (offline) problem, henceforth denoted by SCEP.

1.2 Our Contribution

We formulate the online and offline problems OSCEP and SCEP, and make the following contributions:

1. We present a deterministic online algorithm ACE[1] which, given that node capacities are at least logarithmic, achieves a competitive ratio $O(\log \ell)$ for OSCEP. This result is practically interesting, as the number of to be traversed network functions ℓ is likely to be small in practice. In our analysis, we adapt a proof strategy known from virtual circuit routing [22]. Note however that in contrast to virtual circuit routing, where the end nodes have to

[1] **A**dmission control and **C**hain **E**mbedding.

be connected by a path in the network, in the SCEP, the path must traverse a sequence of ℓ nodes, such that the ith node of this sequence hosts network function f_i. Furhermore, in the SCEP, the path length must be bounded by r hops. So far, only heuristic and offline approaches to solve the service chain embedding problem have been considered [6, 4, 20, 26].

2. We prove that ACE is asymptotically optimal in the class of both deterministic and randomized online algorithms, by adapting a proof strategy from virtual circuit routing in [2]. Moreover, we initiate the study of lower bounds for the offline version of our problem, and show that no good approximation algorithms exist, unless $P = NP$: for unit capacities and already small ℓ, the offline problem SCEP is APX-hard. For arbitrary ℓ, the problem can even become Poly-APX-hard. These results also apply to the offline version of classic online call control problems, which to the best of our knowledge have not been studied before.
3. We present a 0-1 program for SCEP, which also shows that SCEP is in NP for constant ℓ and, taking into account our hardness result, that SCEP is NP-complete for constant ℓ. More precisely, if the number of all possible chains that can be constructed over the network function instances is polynomial in the network size n, then the number of variables in the 0-1 program is also polynomial, and thus the problem is in NP. If m_i is the number of instances of network function f_i in the network, $i = 1, ..., \ell$, and $m = \max_i\{m_i\}$, then the size of the 0-1 program is polynomial for $m^\ell = \text{poly}(n)$. For example, this always holds for constant ℓ. When m is constant, then it holds for $\ell = O(\log n)$.

1.3 Outline

This paper is organized as follows. Section 2 introduces our model and puts the model into perspective with respect to classic online optimization problems. Section 3 presents and analyzes the $O(\log \ell)$-competitive algorithm, Section 4 presents our lower bound, and in Section 5 we present the 0-1 linear program. We summarize our results and conclude our work in Section 6.

2 Model

We are given an undirected network $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. On this graph, we need to route a sequence of requests $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_k)$: σ_i for any i represents a node pair $\sigma_i = (s_i, t_i) \in V \times V$. Each pair σ_i needs to be routed (from s_i to t_i) via a sequence of ℓ network functions $(F_1, \ldots, F_\ell)$. For each network function type F_i, there exist multiple instantiations $f_i^{(1)}, f_i^{(2)}, \ldots$ in the network. (We will omit the superscript if it is irrelevant or clear in the context.) Each of these instances can be applied to σ_i along the route from s_i to t_i. However, in order to minimize the detour via these functions and in order to keep the route from s_i to t_i short, a "nearby instance" $f_i^{(j)}$ should be chosen, for each i. A service chain instance for (s_i, t_i) is denoted by $c_i = (f_1^{(x_1)}, f_2^{(x_2)}, \ldots, f_\ell^{(x_\ell)})$, for some function instances $f_j^{(x_y)}$, $j \in [1, \ell]$.

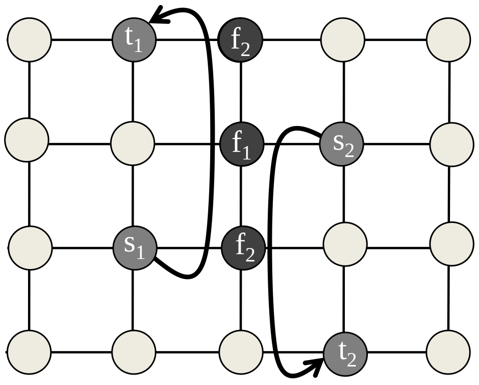

Fig. 1. Illustration of the model: The communication from s_1 to t_1 and from s_2 to t_2 needs to be routed via a service chain (F_1, F_2). In this example, function F_1 is instantiated once, and function F_2 is instantiated twice. Resources for (s_1, t_1) are allocated only at the second instance of F_2 (the upper one).

For ease of presentation, we will initially assume that requests σ_i are of infinite duration. We will later show how to generalize our results to scenarios where requests can have arbitrary and unknown durations.

Concretely, in order to satisfy a request $\sigma_i = (s_i, t_i)$, a route of the following form must be computed:

1. The route must start at s_i, traverse a sequence of network functions $(f_1^{(x_1)}, f_2^{(x_2)}, \ldots, f_\ell^{(x_\ell)})$, and end at t_i. Here, $f_j^{(x_y)}$, $j \in [1, \ell]$ is an instance of the network function of type F_j.
2. The route must not violate capacity constraints on any node $v \in V$. Nodes $v \in V$ are capacitated and resources need to be allocated for each network function which is used, for any (s_i, t_i) pair. Multiple network functions may be available on the same physical machine, and only consume resources once they are used in certain service chains. The capacity $\kappa(v)$ of each node $v \in V$ hence defines the maximum number of requests σ_i for which v can apply its network functions. However, node v can always simply serve as a regular forwarding node for other requests, without applying the function.
3. The route should be of (hop) length at most r (or have a bounded stretch).

Otherwise, a request σ_i must be rejected. For ease of notation, in the following, we will sometimes assume that for a rejected request σ_i, $c_i = \emptyset$. Also note that the resulting route may not form a simple path, but more generally describes a *walk*: it may contain forwarding loops (e.g., visit a network function and come back).

Our objective is to maximize the number of satisfied requests σ_i, resp. to embed a maximum number of service chains. We are mainly interested in the online variant of the problem, where σ is revealed over time. More precisely, and as usual in the realm of online algorithms and competitive analysis, we seek to devise an online algorithm which minimizes the so-called *competitive ratio*: Let $\mathbf{ON}(\sigma)$ denote the number of accepted requests of a given online algorithm for σ and let $\mathbf{OFF}(\sigma)$ denote the number of accepted requests of an optimal offline algorithm. The competitive ratio ρ is defined as the worst ratio (over all possible σ) of the value of **ON** compared to **OFF**. Formally, $\rho = \max_\sigma \mathbf{OFF}(\sigma)/\mathbf{ON}(\sigma)$.

Note that solving this optimization problem consists of two subtasks:

1. *Admission control:* Which requests σ_i to admit, and which to reject?
2. *Assignment and routing:* We need to assign $\sigma_i = (s_i, t_i)$ pairs to a sequence of network functions and route the flow through them accordingly.

See Figure 2 for an illustration of our model.

2.1 Putting the Model into Perspective

From an algorithmic perspective, the models closest to ours occur in the context of online call admission respectively virtual circuit routing. There, the fundamental problem is to decide, in an online manner, which "calls" resp. "virtual circuits" or entire networks, to admit and how to route them, in a link-capacitated graph. [2, 3, 9, 10, 22]

Instead of routes, in our model, service functions have to be allocated and connected to form service chains. In particular, in our model, nodes have a limited capacity and can only serve as network functions for a bounded number of source-destination pairs. The actual routes taken in the network play a secondary role, and may even contain loops. In particular, our model supports the specification of explicit constraints on the length of a route, but also on the stretch: the factor by which the length of a route from a source to a destination can be increased due to the need to visit certain network functions.

Nevertheless, as this paper shows, several techniques from classic literature on online call control can be applied to our model. At the same time, to the best of our knowledge, some of our results also provide new insights into the classic variants of call admission control. For example, our lower bounds on the approximation ratio also translate to classic problems, which so far have mainly been studied from an online perspective.

3 Competitive Online Algorithm

We present an online algorithm ACE for OSCEP. ACE admits and embeds at least a $\Omega(\log \ell)$-fraction of the number of requests embedded by an optimal offline algorithm **OFF**.

Let us first introduce some notation. Let A_j be the set of indices of the requests admitted by ACE just *before* considering the jth request σ_j. The index set of all admitted requests after processing all k requests in σ, will be denoted by A_{k+1} resp. A.

The relative load $\lambda_v(j)$ at node v before processing the jth request, is defined by the number of service chains c_i in which v participates, divided by v's capacity:

$$\lambda_v(j) = \frac{|\{c_i \ : \ i \in A_j, v \in c_i\}|}{\kappa(v)}.$$

We seek to ensure the invariant that capacity constraints are enforced at each node, i.e., $\forall\ v \in V, j \leq k+1 : \lambda_v(j) \leq 1$.

We define $\mu = 2\ell + 2$, and in the following, will assume that

$$\min_v\{\kappa(v)\} \geq \log \mu \tag{1}$$

3.1 Algorithm

In a preprocessing step we compute the length $d(u, v)$ of the shortest path between all pairs of nodes $u, v \in V$ in the network G. Then we compute the set of all possible chains $\mathcal{C}$ that can be constructed from the network function instances $\mathcal{C} = \{c = (f_1, ..., f_\ell)\ :\ \sum_{i=2}^{\ell} d(f_{i-1}, f_i) \leq r,\ f_1 \in F_1, ..., f_\ell \in F_\ell\}$. For a request $\sigma_j = (s_j, t_j)$, let C_j be the set of chains, such that σ_j can be routed through the chains $c \in C_j$ on a path of length at most r, i.e. $C_j = \{c = (f_1, ..., f_\ell) \in \mathcal{C}\ :\ d(s_j, f_1) + d(f_\ell, t_j) + \sum_{i=2}^{\ell} d(f_{i-1}, f_i) \leq r\}$.

The key idea of ACE is to assign to each node, a cost which is exponential in the relative node load. More precisely, with each node we associate a cost $w_v(j)$ just before processing the jth request σ_j:

$$w_v(j) = \kappa(v)(\mu^{\lambda_v(j)} - 1).$$

Our online algorithm ACE simply proceeds as follows:

- When request σ_j arrives, ACE checks if there exists a chain $c_j \in C_j$ satisfying the following condition:

$$\sum_{v \in c_j} \frac{w_v(j)}{\kappa(v)} \leq \ell \tag{2}$$

- If such a chain c_j exists, then *admit* σ_j and assign it to c_j. Otherwise, reject σ_j.

In order to ensure that chains selected for Condition 2 also fulfill the constraint on the maximal route length, ACE simply uses preprocessing. We maintain at each node its relative load. When a new request arrives, ACE has to test the costs of at most $O(n^\ell)$ chains, and the cost can be computed in $O(\ell)$ time per chain. The overall runtime of ACE per step is hence bounded by $O(\ell \cdot n^\ell)$, which is polynomial for constant ℓ.

3.2 Analysis

For the analysis of ACE, we adapt the proof strategy used in [22] in the context of virtual circuit routing. First, in Lemma 1 we prove that the set A of requests admitted by ACE are feasible and respect capacity constraints. Second, in Lemma 2, we show that at any moment in time, the sum of node costs is within a factor $O(\ell \cdot \log \mu)$ of the number of requests already admitted by ACE.

Third, in Lemma 3, we prove that the number of requests admitted by the optimal offline algorithm **OFF** but rejected by the online algorithm, is bounded by the sum of node costs after processing all requests.

Let W be the sum of the node costs after ACE processed all k request, let $A_{\mathbf{OFF}}$ be the indices of the requests admitted by **OFF**, and let $A^* = A_{\mathbf{OFF}} \setminus A$. Then, from Lemma 2 we will obtain a bound $|A| \geq W/(2\ell \cdot \log \mu)$, and from Lemma 3 that $|A^*| \leq W/\ell$.

Thus, even by conservatively ignoring all the requests which ACE might have admitted which **OFF** did not, we obtain that the competitive ratio of ACE is at most $O(\log \ell)$.

Let us now have a closer look at the first helper lemma.

Lemma 1. *For all nodes $v \in V$:*

$$\sum_{j \in A : v \in c_j} 1 \leq \kappa(v).$$

Proof. Let σ_j be the first request admitted by ACE, such that the relative load $\lambda_v(j+1)$ at some node $v \in c_j$ exceeds 1. By definition of the relative load we have $\lambda_v(j) > 1 - 1/\kappa(v)$.

By the assumption that $\log \mu \leq \kappa(v)$, we get

$$\frac{w_v(j)}{\kappa(v)} = \mu^{\lambda_v(j)} - 1 > \mu^{1-1/\log \mu} - 1 = \mu/2 - 1 = \ell.$$

Therefore, by Condition (2), the request σ_j could not be assigned to c_j. We established a contradiction. □

Next we show that the sum of node costs is within an $O(\ell \cdot \log \mu)$ factor of the number of already admitted requests.

Lemma 2. *Let A be the set of indices of requests admitted by the online algorithm. Let k be the index of the last request. Then*

$$(2\ell \log \mu)|A| \geq \sum_v w_v(k+1).$$

Proof. We show the claim by induction on k. For $k = 0$, both sides of the inequality are zero, thus the claim is trivially true. Rejected requests do not change either side of the inequality. Thus, it is enough to show that, for each $j \leq k$, if we admit σ_j, we get:

$$\sum_v (w_v(j+1) - w_v(j)) \leq 2\ell \log \mu.$$

Consider a node $v \in c_j$. Then by definition of the costs:

$$\begin{aligned} w_v(j+1) - w_v(j) &= \kappa(v)(\mu^{\lambda_v(j)+1/\kappa(v)} - \mu^{\lambda_v(j)}) \\ &= \kappa(v)(\mu^{\lambda_v(j)}(\mu^{1/\kappa(v)} - 1)) \\ &= \kappa(v)(\mu^{\lambda_v(j)}(2^{(\log \mu)\cdot 1/\kappa(v)} - 1)) \end{aligned}$$

By Assumption (1), $1 \le \kappa(v)/\log\mu$. Since $2^x - 1 \le x$, for $0 \le x \le 1$, it follows:

$$w_v(j+1) - w_v(j) \ \le \mu^{\lambda_v(j)} \log\mu = \log\mu(w_v(j)/\kappa(v) + 1).$$

Summing up over all the nodes and using the fact that the request σ_j was admitted and chain c_j was assigned, and that the number of nodes $|c_j|$ in c_j is ℓ, we get:

$$\sum_v (w_v(j+1) - w_v(j)) \le \log\mu(\ell + |c_j|) = 2\ell\log\mu.$$

This proves the claim. $\square$

We finally prove that ℓ times the number of requests rejected by ACE but admitted by the optimal offline algorithm **OFF** is bounded by the sum of node costs after processing all requests.

Lemma 3. *Let $A_{\mathbf{OFF}}$ be the set of indices of the requests that were admitted by the optimal offline algorithm, and let $A^* = A_{\mathbf{OFF}} \setminus A$ be the set of indices of requests admitted by $A_{\mathbf{OFF}}$ but rejected by the online algorithm. Then:*

$$|A^*| \cdot \ell \le \sum_v w_v(k+1).$$

Proof. For $j \in A^*$, let c_j^* be the chain assigned to request σ_j by the optimal offline algorithm. By the fact that σ_j was rejected by the online algorithm, we have:

$$\ell < \sum_{v \in c_j^*} \frac{w_v(j)}{\kappa(v)}.$$

Since the costs $w_v(j)$ are monotonically increasing in j, we have

$$\ell < \sum_{v \in c_j^*} \frac{w_v(j)}{\kappa(v)} \le \sum_{v \in c_j^*} \frac{w_v(k+1)}{\kappa(v)}.$$

Summing over all $j \in A^*$, we get

$$|A^*|\ell \ \le \sum_{j \in A^*} \sum_{v \in c_j^*} \frac{w_v(k+1)}{\kappa(v)} \le \sum_v w_v(k+1) \cdot \sum_{j \in A^* : v \in c_j^*} \frac{1}{\kappa(v)} \le \sum_v w_v(k+1).$$

The last inequality follows from the fact that capacity constraints need to be met at any time. $\square$

Theorem 1. ACE *is $O(\log\ell)$-competitive.*

Proof. By Lemma 1, capacity constraints are never violated. It remains to show that the number of requests admitted by the online algorithm is at least $1/(2\log 2\mu)$ times the number of requests admitted by the optimal offline algorithm. The number of requests admitted by the optimal offline algorithm $|A_{\mathbf{OFF}}|$

can be bounded by the number of requests admitted by the online algorithm $|A|$ plus the number of requests in $A^* = A_{\mathbf{OFF}} \setminus A$. Therefore,

$$|A_{\mathbf{OFF}}| \leq |A| + |A^*|.$$

By Lemma 3 this is bounded by

$$|A_{\mathbf{OFF}}| \leq |A| + \frac{1}{\ell} \sum_v w_v(k+1).$$

By Lemma 2 this is bounded by

$$|A_{\mathbf{OFF}}| \ \leq |A| + 2 \cdot (\log \mu) \cdot |A| = (1 + 2\log \mu)|A|$$

Therefore, the number of requests admitted by the optimal offline algorithm is at most $(1 + 2\log \mu)$ times the number of requests admitted by ACE. □

Remarks. We conclude with some remarks. First, we note that our approach leaves us with many flexibilities in terms of constraining the routes through the network functions. For instance, we can support maximal path length requirements: the maximal length of the route from s to t *via the network functions*. A natural alternative model is to define a limit on the *stretch*: the factor by which the "detour" via the network functions can be longer than the shortest path from s to t. Moreover, so far, we focused on a model where requests, once admitted, stay forever. Our approach can also be used to support service chain requests of bounded or even unknown duration. In particular, by redefining μ to take into account the duration of a request, we can for example apply the technique from [22] to obtain competitive ratios for more general models.

4 Optimality and Approximation

It turns out that ACE is asymptotically optimal within the class of online algorithms (Theorem 2). This section also initiates the study of lower bounds for (offline) approximation algorithms, and shows that for low capacities, the problem is APX-hard even for short chains (Theorem 3), and even Poly-APX-hard in general, that is, it is as hard as any problem that can be approximated to a polynomial factor in polynomial time (Theorem 4).

Theorem 2. *Any deterministic or randomized online algorithm for OSCEP must have a competitive ratio of at least $\Omega(\log \ell)$.*

Proof. We can adapt the proof strategy of Lemma 4.1 in [2] for our model. We consider a capacity of $\kappa \geq \log \ell$, and we divide the requests in σ into $\log \ell + 1$ phases. We assume that $n \geq 2\ell^2$, and only focus on a subset L of $\ell = |L|$ nodes which are connected as a chain $(v_1, \ldots, v_\ell)$ and at which the different service chains will overlap. In phase 0, a group of κ service chains are requested, all of which need to be embedded across the nodes $L = \{v_1, \ldots, v_\ell\}$. In phases $i \geq 1$, 2^i groups of κ identical requests will need to share subsets of L of size $\ell/2^i$, that

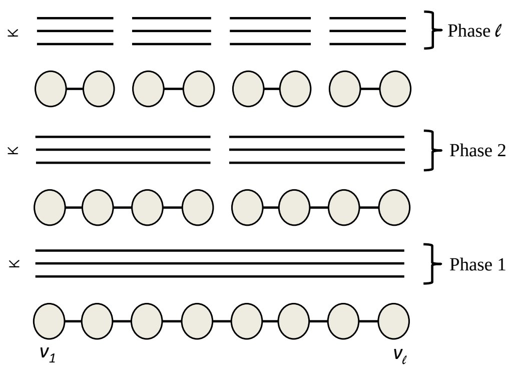

Fig. 2. Illustration of lower bound construction: The adversary issues service chain requests in $1+\log \ell$ phases, where each phase i consists of 2^i groups of $\kappa \geq \log \ell$ requests. In phase 0 the adversary issues requests that can be assigned to $L = (v_1, ..., v_\ell)$. As intersections of chains in phase i with L are becoming shorter over time, the online algorithm needs to decide whether to admit service service chain requests in phases, where each phase consists of groups with κ chains. As chains are becoming shorter over time, the online algorithm faces the problem whether to admit service chains early (and hence block precious resources), or late (in which case the adversary stops issuing new requests).

is, the jth group, $0 \leq j < 2^i$, consists of κ requests to be embedded across nodes $[v_{j\ell/2^i+1}, v_{(j+1)\ell/2^i}]$. See Figure 2 for an illustration.

Let x_i denote the number of requests an online algorithm **ON** admits in phase i. Each request accepted in phase i will occupy $\ell/2^i$ units of capacities of nodes in L. Overall, the nodes in L have a capacity of $\ell \cdot \kappa$, so it must hold that

$$\sum_{i=0}^{\log \ell} \frac{\ell}{2^i} \cdot x_i \leq \ell \cdot \kappa.$$

Now, for $0 \leq j \leq \log \ell$, define $S_j = \frac{\ell}{2^j} \cdot \sum_{i=0}^{j} x_i$. S_j is a lower bound on the occupied capacity on the nodes of L after phase j. Then:

$$\sum_{j=0}^{\log \ell} S_j \; = \sum_{j=0}^{\log \ell} \frac{\ell}{2^j} \sum_{i=0}^{j} x_i = \sum_{i=0}^{\log \ell} x_i \sum_{j=i}^{\log \ell} \frac{\ell}{2^j} \leq \sum_{i=0}^{\log \ell} x_i 2 \frac{\ell}{2^i} = 2\ell\kappa.$$

Hence there must exist a j such that $S_j \leq 2\ell\kappa/\log \ell$. Then after phase j, the number of requests admitted by the online algorithm **ON** is

$$\sum_{i=0}^{j} x_i = \frac{2^j}{\ell} S_j \leq \frac{2^j}{\ell} 2\ell\kappa/\log \ell = 2 \cdot 2^j \kappa/\log \ell.$$

The optimal offline algorithm **OFF** can reject all requests except for those of phase j. The number of requests in phase j, and thus, the number of requests admitted by **OFF** is $2^j\kappa$. $\square$

In the following, we also show that for networks with low capacities, it is not even possible to *approximate* the offline version of the Service Chain Embedding Problem, SCEP, in polynomial time. These lower bounds on the approximation ratio naturally also constitute lower bounds on the competitive ratio which can be achieved for OSCEP by any online algorithm.

In particular, we first show that already for short chains in scenarios with unit capacities, SCEP cannot be approximated well.

Theorem 3. *In scenarios where service chains have length $\ell \geq 3$ and where capacities are $\kappa(v) = 1$, for all v, the offline problem is APX-hard.*

Proof. The proof follows from an approximation-preserving reduction from *Maximum k-Set Packing Problem (KSP)*. The *Maximum Set Packing (SP)* is one of Karp's 21 NP-complete problems, where for a given collection C of finite sets a collection of disjoint sets $C' \subseteq C$ of maximum cardinality has to be found. The KSP is the variation of the SP in which the cardinality of all sets in C are bounded from above by any constant $k \geq 3$, is APX-complete [15]. We refer to such sets as k-sets.

KSP can be reduced to our problem as follows. Let U be the universe and C be a collection of k-sets of U in the KSP. W.l.o.g., we assume that each k-set contains exactly k elements, otherwise we can add disjoint auxiliary elements to the sets in order to obtain exactly k elements in each set in C. For each $u \in U$ in the KSP instance we construct a node v_u in the SCEP instance. Furthermore, for each k-set S in C, we construct a service chain c_S, such that c_S contains exactly the nodes $\{v_u \ : \ u \in S\}$. Let $\mathcal{C}$ be the set of obtained service chains. For the set of requests σ we require that $|\sigma| \geq |\mathcal{C}|$ and that each request can be assigned to each service chain. Due to the unit capacity assumption, the set of admitted request must be assigned to mutually disjoint service chains. Thus, the maximum number of admitted requests is at most the maximum number of disjoint service chains. Since each request can be assigned to each service chain and $|\sigma| \geq |\mathcal{C}|$, an optimal solution for the SCEP determines a maximum set of mutually disjoint service chains. This maximum set of disjoint service chains determines a maximum number of disjoint k-sets, and thus, an optimal solution for the KSP. $\square$

It turns out that in general, with unit capacities, SCEP cannot even be approximated within polylogarithmic factors.

Theorem 4. *In general scenarios where capacities are $\kappa(v) = 1$, for all nodes v, and chain lengths $\ell \geq 3$, the SCEP is APX-hard, and not approximable within ℓ^ε for some $\varepsilon > 0$. Without a bound on the chain length the SCEP with $\kappa(v) = 1$, for all nodes v, is Poly-APX-hard.*

Proof. We reduce the *Maximum Independent Set (MIS)* problem with maximum degree ℓ to the SCEP with capacity $\kappa(v) = 1$, for all $v \in V$ and chain length ℓ. For graphs with bounded degree $\ell \geq 3$, the MIS is APX-complete [21] and cannot be approximated within ℓ^{ε} for some $\varepsilon > 0$ [1]. By our reduction we obtain the APX-hardness and non-approximability within ℓ^{ε} for some $\varepsilon > 0$ for the SCEP. In general, for graphs without degree bound, the MIS is Poly-APX-complete [5], i.e., it is as hard as any problem that can be approximated to a polynomial factor. By our reduction we obtain that the SCEP without chain length bound is Poly-APX-hard.

For an instance $G = (V, E)$ of the MIS problem with maximum degree ℓ, we construct an instance of the SCEP with capacity $\kappa = 1$ and chain length ℓ as follows. For each node $v \in G$, let c_v be the chain whose nodes correspond to the edges in G incident to v. If $\deg_G(v) < \ell$ then we complete the chain with $\ell - \deg_G(v)$ unique auxiliary nodes, in order to have ℓ nodes in the chain. The chain set is $C = \{c_v \,:\, v \in G\}$. For the set of requests σ, we require that $|\sigma| \geq |C|$ and each request $\sigma_i \in \sigma$ can be assigned to each $c \in C$. Assigning a σ_i to a chain $c \in C$ fills the capacity of all nodes in c and the capacity of all chains $c' \in C$ that contain a common node with c. Therefore, no further request σ_j, $j \neq i$, can be assigned to those chains. The chains having a common node with c_v correspond exactly the neighbors of v in G. Therefore, nodes u and v are independent in the MIS instance iff chains c_u and c_v do not have a common node in the SCEP instance. Since each request σ_i can be assigned to each $c \in C$ and $|\sigma| \geq |C|$, a maximum number of admitted requests is determined by a maximum chain set C', such that for all $c_u, c_v \in C'$, c_u and c_v do not contain a common node. Therefore, C' determines a maximum independent set in G. Consequently, an α-approximation for the SCEP would imply an α-approximation for the MIS problem. □

5 Optimal 0-1 Program and NP-Completeness

SCEP can be formulated as a 0-1 integer linear program. If the number of all possible chains that can be constructed over the network function instances is polynomial in the network size, then the number of variables in the 0-1 program is also polynomial, and thus the problem is in NP. 0-1 integer linear programming is one of Karp's NP-complete problems [17]. This together with our hardness results also proves NP-completeness for constant ℓ.

Let $\sigma = \{\sigma_i = (s_i, t_i) : s_i, t_i \in V\}$ be the set of requests, and let $\mathcal{C}$ be the set of possible chains over the network function instances, respecting route length constraints. We refer by $c \in \mathcal{C}$ to a potential chain. For all potential chains $c \in \mathcal{C}$, let S_c be the set of connection requests in σ that can be routed through c on a path of length at most r, i.e., for $c = (v_1, ..., v_\ell)$, let $S_c = \{\sigma_i = (s_i, t_i) \in \sigma : d(s_i, v_1) + \sum_{i=2}^{k} d(v_{i-1}, v_i) + d(v_k, t_i) \leq r\}$, where $d(u, v)$ denotes the length of the shortest path between nodes $u, v \in V$ in the network G. The shortest paths between nodes can be computed in a preprocessing step.

For all connection requests $\sigma_i \in \sigma$, we introduce the binary variable $x_i \in \{0,1\}$. The variable $x_i = 1$ indicates that the request i is admitted in the solution. For all potential network function chains $c \in \mathcal{C}$, we introduce the binary variable $x_c \in \{0,1\}$. The variable $x_c = 1$ indicates that c is selected in the solution. For all $c \in \mathcal{C}$ and $\sigma_i \in \sigma$, we introduce the binary variable $x_{c,i} \in \{0,1\}$. The variable $x_{c,i}$ indicates that the request $\sigma_i = (s_i, t_i) \in \sigma$ is routed through the nodes of c, such that the length of the walk from s_i to t_i through c has length at most r.

$$
\begin{aligned}
\text{maximize} \quad & \sum_{\sigma_i \in \sigma} x_i && && (3)\\
\text{s.t.} \quad & x_i - \sum_{c \in \mathcal{C}} x_{c,i} = 0 && \forall\ \sigma_i \in \sigma && (4)\\
& \sum_{c \in \mathcal{C}: \sigma_i \notin S_c} x_{c,i} = 0 && \forall\ \sigma_i \in \sigma && (5)\\
& x_c \leq x_v && \forall\ v \in V, \forall\ c \in \mathcal{C} : v \in c && (6)\\
& \sum_{c \in \mathcal{C}: v \in c} x_c \geq x_v && \forall\ v \in V && (7)\\
& \sum_{\sigma_i \in \sigma} \sum_{c \in \mathcal{C}: v \in c} x_{c,i} \leq \kappa(v) \cdot x_v && \forall\ v \in V && (8)\\
& x_i, x_v, x_c, x_{c,i} \in \{0,1\} && \forall\ v \in V, \forall\ c \in \mathcal{C}, \forall\ \sigma_i \in \sigma && (9)
\end{aligned}
$$

The objective function (3) asks for admitting a request set of maximum cardinality. The Constraints (4) enforce that each admitted request $\sigma_i \in \sigma$ is assigned to exactly one chain $c \in \mathcal{C}$, and rejected requests are not assigned to any chain, i.e., for each σ_i with $x_i = 1$, there is exactly one chain c with $x_{c,i} = 1$, and for each i with $x_i = 0$, we have $x_{c,i} = 0$ for all c. Constraints (5) state that each $\sigma_i \in \sigma$ can only be assigned to a chain $c \in \mathcal{C}$ with $\sigma_i \in S_c$. By definition of S_c, the nodes s_i and t_i can be routed through c by a path of length at most r. Constraints (6) ensure that if a node $v \in V$ is contained in a selected chain c (i.e., $x_c = 1$), then $x_v = 1$. Constraints (7) enforce that if a node $v \in V$ is not contained in any selected chain, i.e., $x_c = 0$ for all chains c with $v \in c$, then $x_v = 0$. Therefore, Constraints (6) and (7) together imply that $x_v = 1$ iff v is contained in a selected chain c. Constraints (8) describe that the number of requests routed through a node v of a selected chain is limited by the capacity $\kappa(v)$ of v. Furthermore, (8) ensures that if v is not contained in any selected chain (i.e., $x_v = 0$) then no request q is assigned to any chain c with $v \in c$.

The solution of this 0-1 program defines a maximum cardinality set of admitted requests $\sigma_{admit} = \{\sigma_i : x_i = 1\}$, and an assignment of each request $\sigma_i \in \sigma_{admit}$ to a chain $c \in \mathcal{C}$. Each request $\sigma_i \in \sigma_{admit}$ is assigned to a chain $c \in \mathcal{C}$ iff $x_{c,i} = 1$. This assignment guarantees that *(i)* the request $\sigma_i = (s_i, t_i)$ can be routed through c on a path of length at most r, *(ii)* the number of pairs routed through any node $v \in V$ of a selected chain is limited by the capacity $\kappa(v)$ of v, and *(iii)* none of the requests $\sigma_i \in \sigma_{admit}$ are assigned to a non selected chain. Furthermore, it is guaranteed that rejected requests $\sigma_i \in \sigma \setminus \sigma_{admit}$ are not assigned to any chain.

6 Summary and Conclusion

Over the last decades, a large number of middleboxes have been deployed in computer networks, to increase security and application performance, as well as to offer new services in the form of static and dynamic in-network processing (see the services by Akamai, Google Global Cache, Netflix Open Connect). However, the increasing cost and inflexibility of hardware middleboxes (slow deployment, complex upgrades, lack of scalability), motivated the advent of Network Function Virtualization (NFV) [7, 12, 16, 18, 23], which aims to run the functionality provided by middleboxes as software on commodity hardware. The transition to NFV is discussed within standardization groups such as ETSI, and we currently also witness first deployments, e.g., TeraStream [28]. Especially the possibility to chain individual network functions to form more complex services has recently attracted much interest, both in academia [20, 26], as well as in industry [25].

Our paper made a first step towards a better understanding of the algorithmic problem underlying the embedding of service chains. Our main contribution is a deterministic and asymptotically optimal online algorithm ACE which achieves a competitive ratio of $O(\log \ell)$ for OSCEP. This is an encouraging result, as the number ℓ of to-be-chained network functions is likely to be a small constant in practice.

References

1. Alon, N., Feige, U., Wigderson, A., Zuckerman, D.: Derandomized graph products. Computational Complexity 5, 60–75 (1995)
2. Awerbuch, B., Azar, Y., Plotkin, S.A.: Throughput-competitive on-line routing. In: Proc. 34th Annual Symposium on Foundations of Computer Science (FOCS). pp. 32–40 (1993)
3. Awerbuch, B., Azar, Y., Plotkin, S.A., Waarts, O.: Competitive routing of virtual circuits with unknown duration. In: Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 321–327 (1994)
4. Bari, F., Chowdhury, S.R., Ahmed, R., Boutaba, R.: On orchestrating virtual network functions in NFV. CoRR (2015)
5. Bazgan, C., Escoffier, B., Paschos, V.T.: Completeness in standard and differential approximation classes: Poly-(d)apx- and (d)ptas-completeness. Theoretical Computer Science 339(2-3), 272–292 (2005)
6. Dietrich, D., Abujoda, A., Papadimitriou, P.: Network Service Embedding Across Multiple Providers with Nestor. In: Proc. IFIP Networkin (2015)
7. Dobrescu, M., Egi, N., Argyraki, K., Chun, B.G., Fall, K., Iannaccone, G., Knies, A., Manesh, M., Ratnasamy, S.: Routebricks: Exploiting parallelism to scale software routers. In: Proc. ACM SOSP. pp. 15–28 (2009)
8. ETSI: Network functions virtualisation (nfv); use cases. http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf (2014)

9. Even, G., Medina, M.: A nonmonotone analysis with the primal-dual approach: Online routing of virtual circuits with unknown durations. In: Proc. 20th International Colloquium on Structural Information and Communication Complexity (SIROCCO). pp. 104–115 (2013)
10. Even, G., Medina, M., Schaffrath, G., Schmid, S.: Competitive and deterministic embeddings of virtual networks. Elsevier Theoretical Computer Science (TCS) (2013)
11. Fayazbakhsh, S. et al.: Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In: Proc. ACM HotSDN (2013)
12. Gember-Jacobson, A. et al.: OpenNF: Enabling innovation in network function control. In: Proc. ACM SIGCOMM (2014)
13. Gupta, A., Vanbever, L., Shahbaz, M., Donovan, S.P., Schlinker, B., Feamster, N., Rexford, J., Shenker, S., Clark, R., Katz-Bassett, E.: Sdx: A software defined internet exchange. In: Proc. ACM SIGCOMM. pp. 551–562 (2014)
14. Hartert, R., et al.: Declarative and expressive approach to control forwarding paths in carrier-grade networks. In: Proc. ACM SIGCOMM (2015)
15. Hazan, E., Safra, S., Schwartz, O.: On the complexity of approximating k-set packing. Comput. Complex. 15(1), 20–39 (May 2006)
16. Joseph, D., Stoica, I.: Modeling middleboxes. IEEE Network: The Magazine of Global Internetworking 22(5), 20–25 (Sep 2008)
17. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations (1972)
18. Martins, J., Ahmed, M., Raiciu, C., Huici, F.: Enabling fast, dynamic network processing with clickos. In: Proc. HotSDN. pp. 67–72 (2013)
19. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: Enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38(2), 69–74 (Mar 2008)
20. Mehraghdam, S., Keller, M., Karl, H.: Specifying and placing chains of virtual network functions. In: Proc. 3rd IEEE International Conference on Cloud Networking (CloudNet). pp. 7–13 (2014)
21. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. J. Comput. System Sci. 43, 425–440 (1991)
22. Plotkin, S.A.: Competitive routing of virtual circuits in ATM networks. IEEE Journal on Selected Areas in Communications 13(6), 1128–1136 (1995)
23. Schulz-Zander, J., et al.:OpenSDWN: Programmatic control over home and enterprise WiFi. In: ACM Sigcomm Symposium on SDN Research (SOSR) (2015)
24. Sekar, V., Ratnasamy, S., Reiter, M.K., Egi, N., Shi, G.: The middlebox manifesto: Enabling innovation in middlebox deployment. In: Proc. HotNets. pp. 21:1–21:6 (2011)
25. Skoldstrom, P. et al.: Towards unified programmability of cloud and carrier infrastructure. In: Proc. European Workshop on Software Defined Networking (EWSDN) (2014)
26. Soulé, R., Basu, S., Marandi, P.J., Pedone, F., Kleinberg, R., Sirer, E.G., Foster, N.: Merlin: A language for provisioning network resources. In: Proc. 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT). pp. 213–226 (2014)
27. Stoenescu, R., Popovici, M., Olteanu, V., Martins, J., Bifulco, R., Huici, F., Ahmed, M., Smaragdakis, G., Handley, M., Raiciu, C.: In-net: Enabling in-network processing for the masses. In: Proc. ACM EuroSys (2015)
28. Telekom, D.: Terastream. In: `http://www.a10networks.com/resources/files/A10-CS-80103-EN.pdf#search=%22management%22` (2013)