Distributed Federated Service Chaining for Heterogeneous Network Environments

Chen Chen
Loughborough University
Loughborough, UK
c.chen@lboro.ac.uk

Lars Nagel
Loughborough University
Loughborough, UK
l.nagel@lboro.ac.uk

Lin Cui
Jinan University
Guangzhou, China
tcuilin@jnu.edu.cn

Fung Po Tso
Loughborough University
Loughborough, UK
p.tso@lboro.ac.uk

ABSTRACT

Future networks are expected to support cross-domain, cost-aware and fine-grained services in an efficient and flexible manner. Service Function Chaining (SFC) has been introduced as a promising approach to deliver these services. In the literature, centralized resource orchestration is usually employed to process SFC requests and manage computing and network resources. However, centralized approaches inhibit the scalability and domain autonomy in multi-domain networks. They also neglect location and hardware dependencies of service chains.

In this paper, we propose federated service chaining, a distributed framework which orchestrates and maintains the SFC placement while sharing a minimal amount of domain information and control. We first formulate a deployment cost minimization problem as an Integer Linear Programming (ILP) problem with fine-grained constraints for location and hardware dependencies, which is NP-hard. We then devise a Distributed Federated Service Chaining placement approach (DFSC) using inter-domain paths and border nodes information. Our extensive experiments demonstrate that DFSC efficiently optimizes the deployment cost, supports domain autonomy and enables faster decision-making. The results show that DFSC finds solutions within a factor 1.15 of the optimal solution. Compared to a centralized approach in the literature, DFSC reduces the deployment cost by 12% while being one order of magnitude faster.

CCS CONCEPTS

• **Networks** → **Cloud computing**; • **Computing methodologies** → **Distributed algorithms**.

KEYWORDS

Multi-domain, Distributed orchestration, Affinity, Federated Service Chaining

ACM Reference Format:
Chen Chen, Lars Nagel, Lin Cui, and Fung Po Tso. 2021. Distributed Federated Service Chaining for Heterogeneous Network Environments. In *2021 IEEE/ACM 14th International Conference on Utility and Cloud Computing (UCC'21), December 6–9, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3468737.3494091

UCC'21, December 6–9, 2021, Leicester, United Kingdom

ACM ISBN 978-1-4503-8564-0/21/12…$15.00
https://doi.org/10.1145/3468737.3494091

1 INTRODUCTION

In recent years, due to the rise of edge computing and Internet of Things (IoTs), there is a significant increase of diverse applications across heterogeneous network domains [5], [26]. In addition, these applications are greatly dependent on computing and network resources (e.g., CPU, memory, bandwidth) [23], [8], [14]. In practice, Service Function Chaining (SFC) has been introduced as a promising approach to deliver these complex end-to-end applications [15]. In particular, the SFC deployment consumes computing and network resources which are significantly associated with the revenue of service providers.

In the meantime, network services are shifting from centralized clouds to distributed edge networks [7]. With more and more edge service operators joining the market, the network is becoming increasingly more fragmented across different administrative domains. Towards this end, we envision that future network services will span across cloud data centers, ISP and edge networks. Figure 1 shows such an example. It exemplifies three administrative domains with multiple clouds, ISP clouds and edge clouds. There is a jaywalk detection SFC, that detects where people jaywalk, to determine where to install crosswalks [3].

Realizing this application requires a high degree of resource sharing and coordination across multiple administrative domains. However, existing SFC schemes are unable to support this architecture because they either focus on a centralized resource orchestration [12] [22] or only consider inter-cloud collaboration [25] [9], both of which assume that all the topological and resource information are known and that a centralized orchestrator has the control and visibility of all the domains. On the contrary, network operators are notoriously known for restricting the exchange of detailed network information such as topologies, resources and services [11] [1] as doing so could mean revealing their own business-sensitive competing advantages. As a result, a network operator prefers to manage the administrative domain by using its own orchestrator. The distinct management policies and the lack of detailed intra-domain information in domains of other operators exacerbate the difficulties in multi-domain networks. Clearly, existing works would fail to find hosting nodes and routing paths for the SFC request because the topology and resource information are confidential in each domain. Centralized and inter-cloud based solutions are also challenged by management complexity which is brought by the scale of multi-domains. Orchestrating different domains such as public data centers, edge clouds, customer premises is very complex for a centralized orchestrator due to the sheer volume of incoming requests and computational resources. Consequently, the current centralized

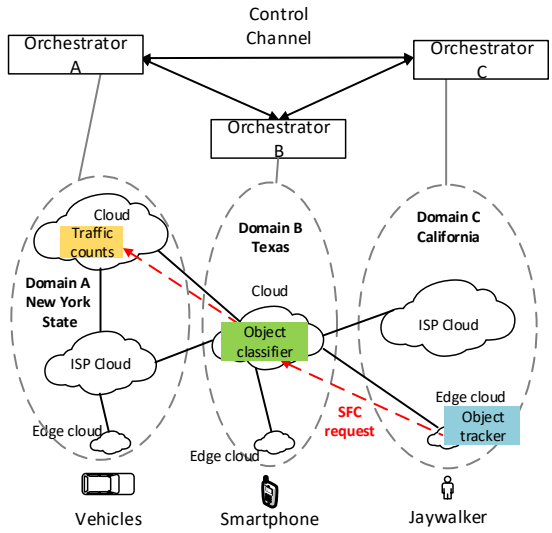


Figure 1: An example for SFC in 3-tier (cloud-ISP-edge) architecture

approaches also lacks of scalability to support large-scale networks with multiple domains.

Moreover, certain Network Functions (NFs) are location and hardware dependent. For example, Figure 1 demonstrates that the object tracker should be placed at Domain California if the jaywalker behaviour in California is to be researched. Hence, there is a need to specify the domain. The object classifier usually needs GPUs to support complex Deep Neural Networks (DNNs) to recognize jaywalkers and cars. A cloud can be equipped with specific hardware such as GPUs. For this reason, the object classifier should be placed at a cloud rather than an edge cloud. Hence, there is also a need to specify the tier. In this paper, we use the term 'tier' to refer to the type of a cloud.

To overcome these challenges, we propose a federated SFC scheme which will not only be scalable but also preserve autonomy and privacy of each participating domain. Our federated SFC scheme supports these by distributing SFC decision making process among relevant domains. Additionally, our federated scheme also enables network operators to take full advantage of the underlying infrastructure in different locations and domains. To the best of our knowledge, this is the first paper to jointly consider the domain autonomy, privacy and scalability in the SFC deployment problem.

Then, we formulate the SFC placement problem as an optimization problem aiming to minimize the cost of computing resources and traffic routing in monetary term. In addition, we provide constraints to specify the affinity correlation of an SFC request with domains and tiers, respectively. We propose a Distributed Federated Service Chaining (DFSC) placement algorithm which embeds a k-shortest path algorithm [24] in an aggregated graph. Our proposed distributed approach merely requires the information of inter-domain paths and border nodes which can be achieved by Border Gateway Protocol (BGP). As illustrated in Figure 1, the SFC request will be partitioned into sub-requests by the ingress orchestrator (i.e., orchestrator C) based on the aggregated graph. Then, the sub-requests are assigned to orchestrator A, B and C, respectively. Thus, the orchestrator A, B and C will hanlde the sub-requests in their own domains to preserve the autonomy and privacy. Also, the decision-making time is significantly reduced because SFC requests are simultaneously processed by multiple orchestrators.

We conducted extensive simulations in two topologies with different scales. We compare our results with the Gurobi solver [13] to demonstrate the optimality gap and validate the proposed scheme. We also implemented the SFCO-AMD (SFC orchestration across multiple domains) approach which manages the multi-domain environment with a centralized orchestrator [22]. We selected the SFCO-AMD to demonstrate the efficiency of the proposed distributed framework in terms of cost reduction and decision-making time. The experiments demonstrate that the DFSC algorithm is within a factor 1.15 of the optimal solution while being several orders of magnitude faster than Gurobi. Compared to the SFCO-AMD approach, DFSC efficiently reduces the overall deployment cost by 12% and is at least one order of magnitude faster.

In short, our contributions are as follows.

- We formulate an Integer Linear Programming (ILP) problem, which takes the location and hardware dependency into account, to minimize the deployment cost.
- We propose a distributed framework which significantly reduces decision-making time and improves the scalability while using a minimal amount of information. This preserves domain autonomy and privacy.
- We have conducted extensive simulations to evaluate DFSC against the Gurobi solver and SFCO-AMD to demonstrate the optimality gap and validate the proposed framework.

The rest of the paper is organized as follows. Section 2 summarizes the main related works, Section 3 introduces the system model, Section 4 proposes the optimization problem, Section 5 presents the proposed DFSC algorithm, Section 6 evaluates the algorithm by means of experiments, and, finally, conclusions are drawn and future works are outlined in Section 7.

2 RELATED WORK

Some previous research works focused on the SFC orchestration problem by considering NF placement, traffic routing and NF chaining in a multi-domain environment. Sun [22] has proposed the SFCO-AMD approach by using a centralized approach. The key idea is to partition the SFC requests by using the link status information such as the minimum latency. Gouareb [12] considered a multi-domain framework to minimize the overall latency which is defined as queueing delay within the edge clouds and inter-cloud links. Another interesting approach is proposed to solve the problem of service chaining across multi-providers [9]. The author formulated an objective function to minimize service cost and resource consumption. They modelled the problem by decomposing it into two sub-problems which are named NF-partitioning problem and NF-subgraph mapping problem. Moreover, they introduced a new service model to simplify the specification of service chaining requests. In order to solve multi-domain SFC problem, Bhamare et al. [6] proposed an ILP problem aiming to minimize inter-cloud

traffic and response time. Furthermore, they compared the results with a simple greedy approach.

Most of the above literature requires a centralized orchestrator, the global topology information or link status information. There are also a few works employing a distributed approach. Flavio [10] proposed the Necklace arichitecture to deploy SFC with convergence and performance guarantees. Although Necklace provides a fully distributed approach, it inevitably requires some global information such as the maximum utility on each single node. Also, sharing the same host node among multiple SFCs leads to a slower convergence time which negatively impact the decision-making time. Kiril [4] proposed an approach to federate multiple domains by employing distributed ledger technologies. However, it takes around 5 seconds for single service federation without considering the deployment time. In [1], Ahmed proposed DistNSE, a distributed framework for service chain embedding across multi-domains. DistNSE employs a bidding mechanism to partition SFC. However, DistNSE chooses to enumerate all the paths between the peering nodes in each domain to find optimal mapping solution. Therefore, this approach requires excessive time to make decision which significantly influences the scalability.

Unlike aforementioned works, the design of DFSC merely requires the inter-domain paths and border nodes information to preserve the domain autonomy. The proposed distributed algorithm provides fast runtime and scalability for large-scale network.

3 SYSTEM MODEL

3.1 Physical Network

The network is modelled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, ..., v_V\}$ is the set of all nodes and $\mathcal{E} = \{e_1, e_2, ..., e_E\}$ is the set of all edges or network links. We use $\mathcal{M} = \{m_1, m_2, ..., m_M\}$ to denote the set of all clouds in the network. In this work, every node in $\mathcal{V}$ is regarded as a cloud. A cloud can host NF instances on their Virtual Machines (VMs), containers or physical hosts. For a cloud m, there are several fields $\{m.cpu, m.mem, m.domain, m.tier\}$, $m.cpu$ and $m.mem$ denote the maximum CPU and memory capacity of cloud m. We use $m.domain$ to denote the domain id of the cloud, and $m.tier$ to denote the tier of the cloud (i.e., tier 1, tier 2 and tier 3). Each link in $\mathcal{E}$ has a capacity of network bandwidth. By $\mathcal{P} = \{p_1, p_2, ..., p_P\}$ we denote all paths in the network.

3.2 Service Function Chaining Model

We use $\mathcal{R}$ to denote the set of all SFC requests. Each request is defined by an 8-tuple $r = \left\{src, dst, \mathcal{N}, \Psi^{bw}, \Psi^{tr}, l^{td}, T_t, D_d\right\}$ where src and dst are the ingress and egress node and $\mathcal{N} = \{n_1, n_2, ..., n_N\}$ is the set of NF in predefined order. Every NF n has several properties $\{type, cpu, mem\}$. The property $n.type$ represents the NF type such as load balancer or firewall. $n.cpu$ and $n.mem$ denote the required CPU and memory resources, respectively. Ψ^{bw} denotes the required bandwidth, Ψ^{tr} the required traffic rate and l_{td} the maximum tolerated delay. D_d and T_t are the domain and tier constraints, respectively.

3.3 SFC Request Affinity

As aforementioned, network operators could have demands for specific constraints on domain and tier due to their operational concerns such as specific hardware or location. Understanding the complexity of such fine-grained constraints is vitally important to influence the future development of SFC services. We define domain and tier constraints by employing the idea of affinity or anti-affinity rules as follows.

$$T_t = \begin{cases} 1 & \text{indicates SFC request to be deployed at tier t.} \\ -1 & \text{prevents SFC request to be deployed at tier t.} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

$$D_d = \begin{cases} 1 & \text{indicates SFC request to be deployed at domain d.} \\ -1 & \text{prevents SFC request to be deployed at domain d.} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

These constraints can be easily adapted to include different granularities such as a set of SFCs, one SFC or one NF. In this paper, we only discuss constraints in the SFC granularity.

4 PROBLEM DESCRIPTION

In geo-distributed multi-domain networks, service providers search for solutions to deploy SFC requests under constraints. The selection of the clouds and the routing path are, for example, subject to resource availability, resource costs and a bound on the maximum tolerated delay. The problem examined in this paper is to find a placement that minimizes the overall deployment cost while meeting the constraints. This problem is formulated as an ILP problem in this section. All symbols and variables are listed in Table 1.

4.1 Problem Statement

The provisioning of an SFC request comes with a monetary cost. When running a chain over geographically distributed clouds, resource costs are caused by consuming computing resources in clouds, and traffic routing costs are incurred by the usage of bandwidth while steering traffic through the chain.

Given a set of SFC requests, our goal is to minimize the overall deployment cost of the request. The deployment cost C of one SFC request consists of two components: the resource cost C_R and the traffic routing cost C_T.

$$C = C_R + C_T \tag{3}$$

Instead of computing the deployment cost for a given time interval, we study the deployment cost per second.

4.1.1 Resource Cost.

Implementing an SFC request costs computing resources. We consider CPU and memory and define the resource cost C_R as

$$C_R = \sum_{n_i \in \mathcal{N}} \sum_{m \in \mathcal{M}} (\beta_d^{cpu} c_{n_i}^{cpu} y_m^{n_i} + \beta_d^{mem} c_{n_i}^{mem} y_m^{n_i}) \tag{4}$$

where β_d^{cpu} and β_d^{mem} are the costs of using one unit of CPU or memory in domain d, respectively. $c_{n_i}^{cpu}$ and $c_{n_i}^{mem}$ are the amount of CPU and memory required by NF n_i. Finally, $y_m^{n_i}$ is an indicator variable which is 1 if NF n_i is deployed on cloud m, and 0 otherwise.

4.1.2 Traffic Routing Cost.

When network traffic is routed through a chain across multiple domains, routing costs can incur for the network links usage [27] [20]. The cost is caused by the total bandwidth consumed on virtual links that are deployed on physical links. Since this cost is determined by the routing between source and target cloud, we use a unified cost parameter c_p to denote the overall traffic routing cost of one unit of traffic when using path p. The traffic routing cost C_T of one SFC request is defined as follows.

$$C_T = \sum_{p\in\mathcal{P}} c_p \Psi^{tr} y_p^{src,n_1} + \sum_{p\in\mathcal{P}} \sum_{n_i,n_j\in\mathcal{N}} c_p \Psi^{tr} y_p^{n_i,n_j} + \sum_{p\in\mathcal{P}} c_p \Psi^{tr} y_p^{n_N,dst}. \tag{5}$$

Ψ^{tr} denotes the traffic rate of the request. c_p is the unified traffic routing cost parameter which is proportional to the geographical distance. $y_p^{n_i,n_j}$ is the indicator variable which is 1 if path p is used between NF n_i and n_j, and 0 otherwise. Hence, y_p^{src,n_1} indicates whether path p is used to connect the ingress node with the first NF, and $y_p^{n_N,dst}$ indicates whether path p is used to connect the last NF in the request with the egress node.

4.2 SFC Orchestration Problem In Heterogeneous Multi-domain Networks

In this paper, we consider the SFC orchestration as an online problem which means that the SFC requests are processed one by one.

Problem 1. *Given the domain and tier constraints, the amount of resources available at each cloud, the amount of CPU and memory required by the NF components, the problem is to find the placement for the NFs and the subsequent traffic route that minimizes the deployment cost.*

$$minimize \quad C = C_R + C_T \tag{6}$$

The problem is subject to a number of constraints.

The CPU and memory requirements must not exceed the remaining CPU capacity C_m^{cpu} and memory capacity C_m^{mem} on cloud m.

$$\sum_{n_i\in\mathcal{N}} c_{n_i}^{cpu} y_m^{n_i} \le C_m^{cpu}, \quad \forall m \in \mathcal{M} \tag{7}$$

$$\sum_{n_i\in\mathcal{N}} c_{n_i}^{mem} y_m^{n_i} \le C_m^{mem}, \quad \forall m \in \mathcal{M} \tag{8}$$

The required bandwidth amount must not exceed the remaining bandwidth capacity C_p^{bw} on path p.

$$\sum_{p\in\mathcal{P}} \sum_{n_i,n_j\in\mathcal{N}} \Psi^{bw} y_p^{n_i,n_j} \le C_p^{bw} \tag{9}$$

Also, each NF in the request can only be assigned once.

$$\sum_{m\in\mathcal{M}} y_m^{n_i} \le 1 \tag{10}$$

Table 1: Symbols and Variables

Symbols and Variables	Description
Physical Network	
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Physical network graph.
$\mathcal{R} = \{r_1, r_2, ..., r_R\}$	All SFC requests.
$\mathcal{M} = \{m_1, m_2, ..., m_M\}$	All cloud data centers.
$\mathcal{V} = \{v_1, v_2, ..., v_V\}$	All nodes in the network.
$\mathcal{E} = \{e_1, e_2, ..., e_E\}$	All physical links in the network.
$\mathcal{P} = \{p_1, p_2, ..., p_P\}$	All paths in the network.
C_m^{cpu}, C_m^{mem}	Remaining CPU and memory capacity in cloud m.
C_p^{bw}	Remaining bandwidth capacity on path p.
SFC Graph	
src	Ingress point of the SFC request.
dst	Egress point of the SFC request.
$\mathcal{N} = \{n_1, n_2, ..., n_N\}$	All NFs in the SFC request.
$m.cpu, m.mem$	Total capacity of CPU and memory on cloud m.
$c_{n_i}^{cpu}, c_{n_i}^{mem}$	Amount of CPU and memory required by NF n_i.
Ψ^{bw}	Required bandwidth.
Ψ^{tr}	Required traffic rate.
l^{td}	Maximum tolerated delay.
l^d	End to end delay of given SFC.
D_d	Domain constraint.
T_t	Tier constraint.
Parameters	
β_d^{cpu}	Resource cost of one unit of CPU in domain d.
β_d^{mem}	Resource cost of one unit of memory in domain d.
c_p	Traffic routing cost parameter of path p.
Binary Variables	
$y_m^{n_i}$	Indicator if NF n_i is hosted on cloud m.
$y_p^{n_i,n_j}$	Indicator if traffic from NF n_i to n_j is routed through path p.

The end to end delay of the path for SFC request must meet the constraint of the maximum tolerated delay.

$$l^d \le l^{td} \tag{11}$$

where l^d denote the end to end delay of the request.

Finally, we phrase the domain and tier constraints for an SFC. If the SFC request includes a tier affinity constraint specifying tier t, the request must be deployed at tier t.

$$\text{if} \quad T_t = 1, \quad \sum_{m\in\mathcal{M}^t} y_m^{n_i} = 1, \quad \forall n \in \mathcal{N} \tag{12}$$

where $\mathcal{M}^t$ is the set of clouds that belongs to tier t.

If the SFC request includes a tier anti-affinity constraint specifying tier t, the request must not be deployed at tier t.

$$\text{if} \quad T_t = -1, \quad \sum_{m\in\mathcal{M}^t} y_m^{n_i} = 0, \quad \forall n \in \mathcal{N} \tag{13}$$

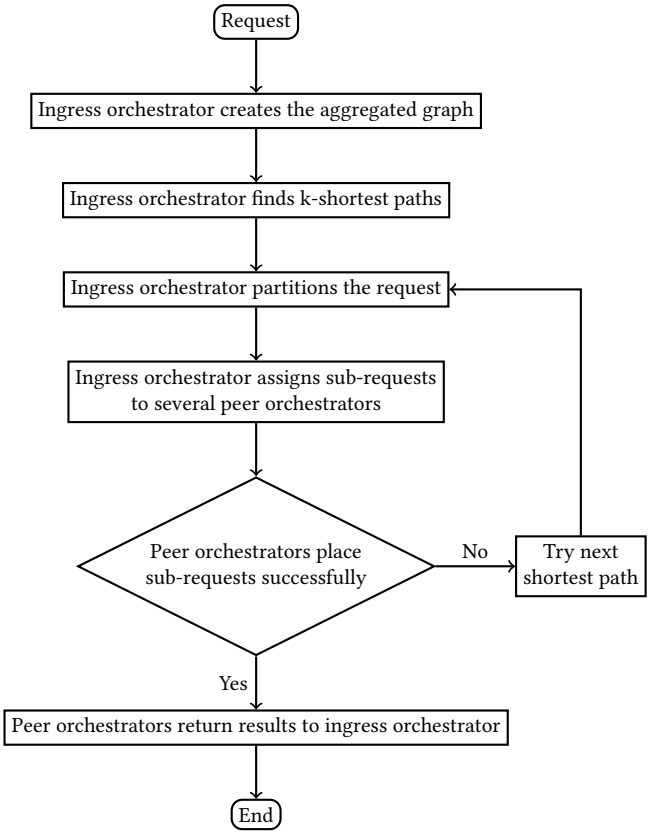


Figure 2: The workflow of DFSC

If the SFC request includes a domain affinity constraint specifying domain d, the request must be deployed at domain d.

$$\text{if} \quad D_d = 1, \quad \sum_{m \in \mathcal{M}^d} y_m^{n_i} = 1, \quad \forall n \in \mathcal{N} \tag{14}$$

where $\mathcal{M}^d$ is the set of clouds that belongs to domain d.

If the SFC request includes a domain anti-affinity constraint specifying domain d, the request must not be deployed at domain d.

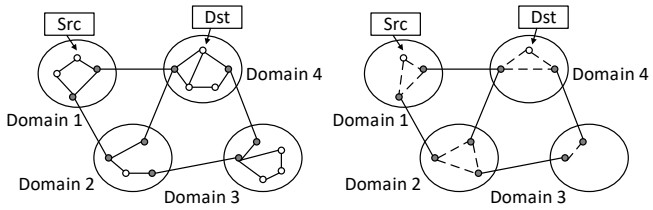
$$\text{if} \quad D_d = -1, \quad \sum_{m \in \mathcal{M}^d} y_m^{n_i} = 0, \quad \forall n \in \mathcal{N} \tag{15}$$

This cost minimization problem is NP-hard because the classical minimum knapsack problem (MKP), which is known to be NP-hard, can be reduced to it [27].

5 DISTRIBUTED FEDERATED SERVICE CHAINING

In this section, we elaborate the distributed federated service chaining algorithm. The proposed algorithm is designed to find solutions for Equation 6. The aim of this algorithm is to distribute the decision-making process to multiple domain orchestrators and preserve the autonomy of domains. A global orchestrator is not required. Thus, the current network configuration will not be changed.

Figure 2 shows the workflow of our proposed algorithm. In this work, we assume that each domain has a domain orchestrator which



(a) The physcial network (b) The aggregated graph

Figure 3: Topology aggregation

is aware of all the intra-domain information but has only limited information about other domains (e.g., the inter-domain link to other domains). When an SFC request arrives at a domain, the ingress orchestrator partitions the request into sub-requests. In the example shown in Figure 1, the chain of the SFC request starts in Domain C and ends in Domain A. Hence, Orchestrator C is the ingress orchestrator for this request. Subsequently, Orchestrator A and B are referred as peer orchestrators.

The ingress orchestrator builds an aggregated graph including inter-domain links and aggregated nodes. It employs the k-shortest path algorithm [24] in the aggregated graph and decides how to assign the NFs to the different domains. The k-shortest path algorithm provides several candidate paths which helps to avoid resource bottlenecks at the cost of only a slight increase in runtime. Subsequently, the decision is sent to peer orchestrators. Finally, each peer orchestrator finds a solution within their own domain and sends deployment results back to the ingress orchestrator. Details are provided in the following subsections.

5.1 Constructing Aggregated Graph

Privacy is always a paramount concern in multi-domain networks [16]. Hence, we assume that only border nodes (nodes connected to a inter-domain link) are visible to other peer orchestrators. This can be achieved by the Border Gateway Protocol [1]. In Figure 3(a), grey nodes represent the border nodes, and the link that traverses two domains is an inter-domain link. Thus, the intra-domain network connectivities and topologies are regarded as confidential information of each domain. In this setup, domain internal information are well preserved.

The aggregated graph consists of inter-domain links, border nodes and logical intra-domain links. As shown in Figure 3(b), the grey nodes represent border nodes. Then, the solid and dotted lines denote the inter-domain and logical links, respectively. The white nodes represent the ingress and egress node of the SFC request. The aggregated graph is derived from the physical network by employing Full-Mesh aggregation [18]. The approach is to only keep the border nodes and remove other nodes. Then, each pair of border nodes in the same domain is connected by a logical link. The latency l^d and traffic routing cost parameter c_p of logical link are temporarily set to 0 in the aggregation phase because such information is regarded as confidential. In the aggregation graph, there are enough information for the ingress orchestrator to make first-phase decision.

5.2 Distributed Federated Service Chain Placement

In this subsection we explain how to find solutions with the distributed framework. The pseudocode of DSFC algorithm is shown in Algorithm 1. First, the aggregated graph is constructed by the ingress orchestrator when the SFC request arrives. Then, the ingress orchestrator employs the k-shortest path algorithm to find k candidate paths from src to dst based on traffic routing cost [24]. Also, the ingress orchestrator assigns NFs to domains on the candidate path according to the resource cost parameter β_d^{cpu} and β_d^{mem}. Finally, each peer orchestrator invokes Algorithm 2 which strives to find a solution for intra-domain placement. If the first intra-domain placement fails, the algorithm will try to deploy the request on the second candidate path until all candidate paths fail.

Algorithm 1 Distributed Federated Service Chaining Placement

1: Input: SFC request r, resource cost parameter β_d^{cpu} and β_d^{mem} of each domain, traffic routing cost parameter c_p.
2: Output: Routing path, hosted nodes, deployment cost.
3: Construct aggregated graph $\mathcal{G}^a = (\mathcal{V}^a, \mathcal{E}^a)$;
4: Find k-shortest path $\mathcal{P}^k = p_1, p_2, ..., p_k$ according to traffic routing cost;
5: **while** $p \in P^k \neq \emptyset$ **do**
6: **if** $\Psi^{bw} > C_p^{bw}$ **then**
7: continue;
8: **end if**
9: Sort the set of domains on the path p based on β_d^{cpu} and β_d^{mem};
10: **for** $n \in \mathcal{N}$ **do**
11: Find the available domain d with lowest
12: resource cost;
13: Check the domain constraint;
14: Assign the NF to d;
15: **end for**
16: **for** each domain d on path p **do**
17: Call **Algorithm 2** on peer orchestrators;
18: **if Algorithm 2** fails **then** break;
19: **end if**
20: **end for**
21: **if** All NF are assigned **then**
22: **return** Routing path, hosted nodes, the
23: deployment cost;
24: **end if**
25: **end while**
26: **return** Deployment failure.

The DFSC algorithm is initialized in line 1 in Algorithm 1. Then, line 3-15 and line 21-26 are executed on the ingress orchestrator while line 16-20 is carried out on peer orchestrators. Line 3 is to construct the aggregated graph. In line 4, the k-shortest path algorithm calculates the candidate paths. In line 6-8, the bandwidth requirement is checked. In line 9, the ingress orchestrator sorts all related domains. For simplicity, we take the sum of β_d^{cpu} and β_d^{mem} as the sorting criteria. Then, the ingress orchestrator iterates over all domains which have enough resource capacity to host the NF. Next, a feasible domain with lowest resource cost is selected and the corresponding NF is assigned to the domain. In line 17, Algorithm 2 is called. Finally, the ingress orchestrator checks the deployment and return the result in line 21-26.

Algorithm 2 describes the intra-domain deployment which is parallelly executed on each peer orchestrator. In line 3, k-shortest path algorithm is employed to find candidate paths within a single domain. Subsequently, in line 4-11 we search for the first available cloud to host the NF. When all assigned NFs in this domain is hosted, the peer orchestrator sends the result to the ingress orchestrator. If all candidate paths are iterated, but some NFs are not able to be hosted. Then, the intra-domain deployment fails and sends message to the ingress orchestrator.

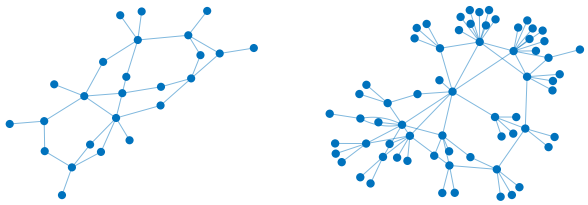
Algorithm 2 Intra-domain Deployment

1: Input: Assigned NFs $\mathcal{N}^d$ for domain d, single domain graph $\mathcal{G}^d = (\mathcal{V}^d, \mathcal{E}^d)$.
2: Output: Routing path p^d in $\mathcal{G}^d$, a set of nodes $\mathcal{V}^h$ that host NFs.
3: Find k-shortest paths $\mathcal{P}^k = p_1, p_2, ..., p_k$ according to traffic routing cost in $\mathcal{G}^d$;
4: **while** $p \in P^k \neq \emptyset$ **do**
5: **if** $\Psi^{bw} > C_p^{bw}$ **then**
6: continue;
7: **end if**
8: **for** Each n_i in $\mathcal{N}^d$ **do**
9: Find the node v on p with enough computing resources;
10: Check the tier constraint;
11: Assign the NF to v;
12: **end for**
13: **if** all NFs are assigned **then**
14: **return** p^d and $\mathcal{V}^h$;
15: **end if**
16: **end while**
17: **return** Intra-domain deployment failure.

6 EVALUATION

6.1 Evaluation environment

We have implemented DFSC and SFCO-AMD in Network Simulator 3 (NS3) and conducted our experiments on a server with Intel(R) Core(TM) i5-8265 CPU 1.60 GHz and 8 GB RAM. Each SFC request contains varying number of Network Functions randomly chosen from 5 extensively studied Network Functions (i.e., Firewall, Proxy, NAT, IDS and Load Balancer). The numbers of cores for one cloud, ISP cloud and edge cloud are 1000, 500, and 200, respectively. The memory capacities for one cloud, ISP cloud and edge cloud are 1000GB, 500GB and 200GB, respectively. The bandwidth capacities for inter-domain and intra-domain links are set to 1000Mbps and 500Mbps, respectively. The propagation delay is randomly selected between 2ms and 10ms for every link. In the simulation, the ingress nodes, the egress nodes and NFs in SFC requests are all randomly selected. For each NF, the number of CPU cores is randomly selected from (1,10). The memory requirement is set as numbers distributed randomly between (1,20) GB. Then, we set up some simulation



(a) Topolgy Agis (b) Topology Internode

Figure 4: Topology in the simulation

parameters according to some similar papers [19], [21]. The number of NFs in the chain are set to 3. The traffic rate requirement is set between 100kbps and 500kbps. Also, the domain and tier constraints are randomly created. The value k for k-shortest path is empirically selected as 8. The reason behind this setup is that the acceptance does not significantly increase when the value of k is greater than 8.

To benchmark the performance of DFSC with the optimal solution, we have used an ILP solver Gurobi to obtain the offline optimum of the cost minimization problem by solving Equation 6.

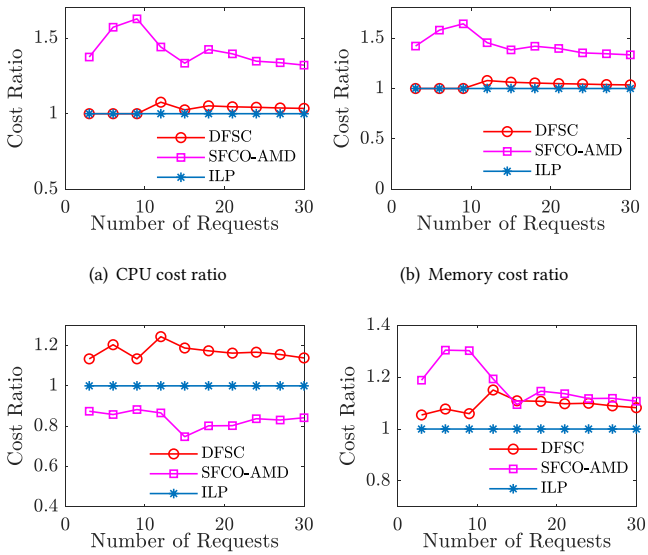
In addition, we have also compared the performance of DFSC with SFCO-AMD in various topologies. First, the SFCO-AMD approach employs a centralized orchestrator to averagely partition SFC requests into several sub-requests. Then, each sub-request is deployed within a single domain. This algorithm is implemented in the experiment to compare with DFSC algorithm. We adapted it to the proposed network architecture by introducing affinity-constraints. Finally, we calculated the deployment cost from the result of SFCO-AMD approach.

The cost settings are randomly selected from a price ranges set around Amazon instance prices [2]. The cost for one CPU per second ranges from \$0.001 to \$0.005. The cost for one GB of memory per second ranges from \$0.001 to \$0.004. The cost for sending one Mbit over one link ranges from \$0.02 to \$0.05. Moreover, the maximum tolerated end to end delay is randomly selected between 50 ms and 100 ms.

6.2 Evaluation results

6.2.1 Performance Comparison Between Gurobi and heuristics. In this subsection, we compared the performance of DFSC, SFCO-AMD and the ILP solver to study the optimality gap. The network topology is a US backbone network named Agis from the Internet Topology Zoo [17], as shown in Figure 4(a). The topology consists of 25 nodes and 30 links. We divided the nodes into 6 domains based on geo-distance. Then we randomly created different tiers for nodes. In order to evaluate the performance of Algorithm 1, we compared it with the optimal solution from Gurobi. Specifically, given a SFC request, the solver searches for hosting clouds and traffic routing paths that minimize the cost in Equation 6.

The cost raio is the ratio of cost achieved by heuristics (DFSC and SFCO-AMD) to that achieved by the offline minimum. Figure 5(a) and 5(b) shows that the DFSC algorithm stays within 1.1 times to the optimal solution in terms of CPU cost and memory cost. Whereas,



(a) CPU cost ratio (b) Memory cost ratio

(c) Traffic routing cost ratio (d) Total cost ratio

Figure 5: Number of requests vs. cost ratio in topology Agis

the SFCO-AMD approach only stays within 1.6 times to the optimal. From Figure 5(c) it is apparent that the DFSC is outperformed by 20% comparing with SFCO-AMD approach. This is because, DFSC needs to make trade-off between computing resource cost and routing cost so as to minimize the overall cost. Also, SFCO-AMD performs better than the Gurobi solver in terms of traffic routing cost, because Gurobi only aims to find the optimum of overall deployment cost. Finally, in Figure 5(d) the overall deployment cost of DFSC varies between 1.05 and 1.15 times to the optimum. However, the SFCO-AMD approach varies between 1.1 and 1.3 times to the optimum. Hence, the DFSC approach performs better in terms of overall deployment cost. Therefore, the DFSC approach can effectively find a near-optimal solution.

Table 2: Decision Making Time

Number of requests	DFSC	SFCO-AMD	Gurobi
3	0.6s	26.8s	201s
6	1.49s	36.9s	505s
9	3.6s	71.7s	903.2s
12	3.76s	85s	1441.3s
15	5.44s	109.9s	1978.8s
18	5.75s	127s	2958.6s
21	6.99s	157.1s	3783.2s
24	9.76s	170.91s	4857.8s
27	9.74s	177.1s	4192s
30	11.3s	221.9s	5673.9s

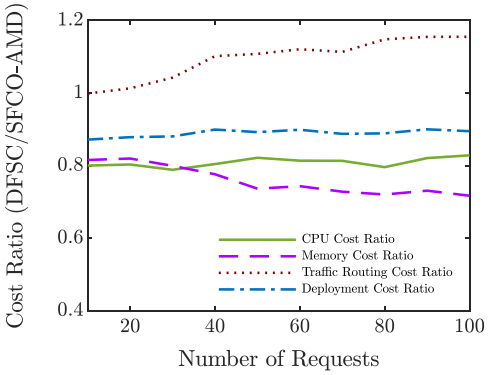


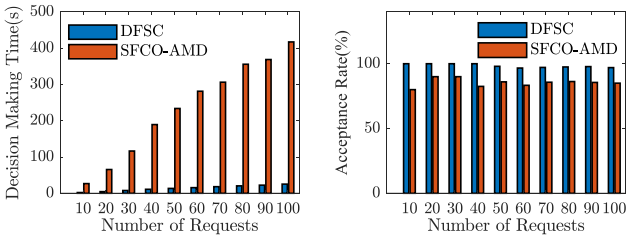
Figure 6: The DFSC/SFCO-AMD cost ratio in topology Internode

After that, Table 2 shows the average execution times of DFSC, SFCO-AMD and Gurobi. The execution time is measured by recording the running time of the algorithm. The DFSC algorithm provides solutions that are 1.15 times within the optimum and the execution time is several order of magnitude faster than Gurobi. Moreover, DFSC is at least one order of magnitude faster than SFCO-AMD.

6.2.2 Evaluation of DFSC and SFCO-AMD approach.

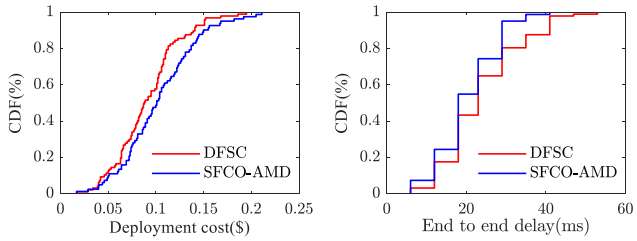
In this subsection, we compared the performance of DFSC approach and the SFCO-AMD approach. The SFCO-AMD approach is regarded as baseline. Since the integer programming solver has difficulties to identify feasible solutions in a reasonable amount of time, we did not implement Gurobi for the Internode topology in this subsection. Figure 6 illustrates the cost ratio of CPU cost, memory cost, traffic routing cost and deployment cost in the Topology Internode with 66 nodes and 75 links. The topology is illustrated in Figure 4(b) and divided into 9 domains. Each domain includes several nodes. Also, the number of requests increases from 10 to 100. Then, we calculated the average cost per request and the acceptance rate. From the Figure 6, we observed that the DFSC approach outperforms the SFCO-AMD approach by 12% in terms of total deployment cost. Also, the DFSC approach reduces the memory cost up to 26%. However, the SFCO-AMD approach outperforms the DFSC approach by 11 % in terms of traffic routing cost. This is because, DFSC needs to make trade-off between traffic routing cost and computing resource cost so as to minimize the overall cost. Since the DFSC algorithm always outperforms the SFCO-AMD approach in deployment cost, the effectiveness of our DFSC distributed approach is proved.

Then, we continued to investigate the decision-making time of both algorithms. We conducted the experiment in Internode topology with different numbers of requests. The result is depicted in Figure 7(a). As the number of SFC requests increases, the decision-making time for both algorithm rises. It is apparent that the DFSC algorithm performs better in each case. The decision-making time of the DFSC algorithm increases slowly while it rapidly grows in the case of SFCO-AMD. This is because the decision of DFSC is supported by several parallel orchestrators which significantly reduces the workload for each orchestrator and provides efficient scalability. Moreover, the DFSC algorithm employs k-shortest path



(a) Decision-making time (b) Acceptance rate

Figure 7: The comparison of decision-making time and acceptance rate



(a) CDF of deployment cost (b) CDF of end to end delay

Figure 8: CDF of deployment cost and delay

algorithm which consumes much less time than searching all candidate paths in SFCO-AMD approach. Then, Figure 7(b) demonstrates the acceptance rate over distinct number of requests. As the number of requests increases, the acceptance rate falls slightly in both cases. This is because, resource bottlenecks emerge and prevent SFC requests to be deployed. However, the DFSC approach still outperforms the SFCO-AMD approach by 12% in average.

Figure 8(a) shows the percentage of requests deployed as a function of the deployment cost. While SFCO-AMD deploys only 60% of the SFC requests within $0.11, DFSC deploys 80% of the requests. Finally, Figure 8(b) shows the CDF of the end to end delay. We dynamically routed traffic through SFC when one request is deployed. Then, the end to end delay of each SFC is dynamically measured by sending one packet from ingress to egress node. The SFCO-AMD algorithm outperforms the DFSC approach, where 98% of SFC requests experience less than 35 ms delay. However, only 87% SFC requests of the DFSC algorithm got less than 35 ms delay. There are two reasons for this finding. Since the DFSC approach achieved higher acceptance rate, more traffic is introduced in the network which could cause higher delay. Second, the DFSC approach could select a longer routing path to reduce the overall cost.

6.2.3 Acceptance rate over k value.

As shown in Table 3, we investigated the relationship between acceptance rate and the k value of the k-shortest path algorithm in Algorithm 1. The k value specifies how many candidate paths the k-shortest path algorithm returns. To clearly demonstrate the performance change, we reduced the numbers of CPU cores to 100, 50 and 20, respectively. Then we reduced the memory capacity of

Table 3: DFSC performance under different k value

Value of k	Acceptance rate	Decision-making time
1	73%	17.0s
2	74%	17.6s
3	75%	18.3s
4	75%	19.1s
5	75%	20.4s
6	78%	21.1s
7	78%	21.9s
8	79%	21.5s

different tiers to 100GB, 50GB and 20GB, respectively. The bandwidth capacity for inter and intra-domain links shrank to 100Mbps and 50Mbps. The experiment was still executed in topology Internode. We conducted 8 experiments in which k increases from 1 to 8 with step size 1. Each time 100 SFC requests are delivered to the network.

As shown in Table 3, when k value increases from 1 to 8, the acceptance rate increases from approximately 73% to 79%. This is because, higher k value indicates more candidate paths for the DFSC algorithm which is used to search for less used links and nodes to accommodate SFC requests. Therefore, the orchestrator is more likely to place the request and leverage the acceptance rate. Also, Table 3 demonstrates the execution time under different number of value k. We can see that the execution time increases slightly when the value of k increases. This is because, the k-shortest path algorithm spends more time to search for candidate paths.

7 CONCLUSION

In this paper, we formulated a cost minimization problem for deploying service function chains in multi-domain networks under affinity constraints. We proposed DFSC, a distributed and scalable scheme to solve this problem. DFSC preserves the autonomy and privacy of each domain by using a minimal amount of network information. Scalability and decision-making time are improved by using a distributed algorithm. The inclusion of affinity constraints for domains and tiers allows to respect potential location and hardware dependencies of service functions.

Our extensive experiments demonstrate that DFSC reduces the deployment cost by 12% while the decision-making time is one order of magnitude faster than the SFCO-AMD approach. The proposed algorithm also has a 12% higher acceptance rate at the cost of an end-to-end latency increase by 5ms. As a part of our future work we plan to verify the proposed framework in a Mininet emulator and a real-world testbed.

8 ACKNOWLEDGMENT

This work has been partially supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grants EP/P004407/2 and EP/P004024/1, and InnovateUK grant 106199-47198; the Chinese National Research Fund (NSFC) No. 62172189 and 61772235, and the Science and Technology Program of Guangzhou No. 202002030372; the China Scholarship Council (CSC).

REFERENCES

[1] Ahmed Abujoda and Panagiotis Papadimitriou. 2016. DistNSE: Distributed network service embedding across multiple providers. In *2016 8th International Conference on Communication Systems and Networks, COMSNETS 2016*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/COMSNETS.2016.7439948

[2] Amazon. 2021. EC2 On-Demand Instance Pricing – Amazon Web Services. https://aws.amazon.com/ec2/pricing/on-demand/.

[3] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. 2017. Real-Time Video Analytics: The Killer App for Edge Computing. *Computer* 50, 10 (2017), 58–67.

[4] Kiril Antevski and Carlos J. Bernardos. 2020. Federation of 5G services using distributed ledger technologies†. *Internet Technology Letters* 3, 6 (2020), e193. https://doi.org/10.1002/itl2.193 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/itl2.193

[5] Deval Bhamare, Aiman Erbad, Raj Jain, Maede Zolanvari, and Mohammed Samaka. 2018. Efficient virtual network function placement strategies for Cloud Radio Access Networks. *Computer Communications* 127 (sep 2018), 50–60. https://doi.org/10.1016/j.comcom.2018.05.004

[6] Deval Bhamare, Mohammed Samaka, Aiman Erbad, Raj Jain, Lav Gupta, and H. Anthony Chan. 2017. Optimal virtual network function placement in multi-cloud service function chaining architecture. *Computer Communications* 102 (2017), 1–16. https://doi.org/10.1016/j.comcom.2017.02.011

[7] L Cui, F Tso, and W Jia. March 2020. Federated Service Chaining: Architecture and Challenges. *IEEE Communications Magazine* 58, 3 (March 2020), 47–53.

[8] Lin Cui, Fung Po Tso, Song Guo, Weijia Jia, Kaimin Wei, and Wei Zhao. 2019. Enabling Heterogeneous Network Function Chaining. *IEEE Transactions on Parallel and Distributed Systems* 30, 4 (apr 2019), 842–854. https://doi.org/10.1109/TPDS.2018.2871845

[9] David Dietrich, Ahmed Abujoda, Amr Rizk, and Panagiotis Papadimitriou. 2017. Multi-Provider Service Chain Embedding With Nestor. *IEEE Transactions on Network and Service Management* 14, 1 (mar 2017), 91–105. https://doi.org/10.1109/TNSM.2017.2654681

[10] Flavio Esposito, Maria Mushtaq, Michele Berno, Gianluca Davoli, Davide Borsatti, Walter Cerroni, and Michele Rossi. 2021. Necklace: An Architecture for Distributed and Robust Service Function Chains With Guarantees. *IEEE Transactions on Network and Service Management* 18, 1 (2021), 152–166. https://doi.org/10.1109/TNSM.2020.3036926

[11] Antonio Francescon, Giovanni Baggio, Riccardo Fedrizzi, Ramon Ferrusy, Imen Grida Ben Yahiaz, and Roberto Riggio. 2017. X–MANO: Cross–domain management and orchestration of network services. In *2017 IEEE Conference on Network Softwarization (NetSoft)*. 1–5. https://doi.org/10.1109/NETSOFT.2017.8004223

[12] Racha Gouareb, Vasilis Friderikos, and Aghvami. 2018. Virtual Network Functions Routing and Placement for Edge Cloud Latency Minimization. *IEEE Journal on Selected Areas in Communications* 36, 10 (oct 2018), 2346–2357. https://doi.org/10.1109/JSAC.2018.2869955

[13] LLC Gurobi Optimization. 2021. Gurobi Optimizer Reference Manual. http://www.gurobi.com

[14] Wajdi Hajji, Thiago A.L. Genez, Fung Po Tso, Lin Cui, and Iain Phillips. 2018. Dynamic Network Function Chain Composition for Mitigating Network Latency. In *Proceedings - IEEE Symposium on Computers and Communications*, Vol. 2018-June. Institute of Electrical and Electronics Engineers Inc., 316–321. https://doi.org/10.1109/ISCC.2018.8538646

[15] Ed. J. Halpern and Ed. C. Pignataro. 2015. RFC 7665 - Service Function Chaining (SFC) Architecture. *Internet Engineering Task Force (IETF) - Request for Comments: 7665* (2015), 487–492. https://tools.ietf.org/html/rfc7665http://ir.obihiro.ac.jp/dspace/handle/10322/3933

[16] Kalpana D. Joshi and Kotaro Kataoka. 2020. pSMART: A lightweight, privacy-aware service function chain orchestration in multi-domain NFV/SDN. *Computer Networks* 178 (2020), 107295. https://doi.org/10.1016/j.comnet.2020.107295

[17] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. 2011. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1765–1775. https://doi.org/10.1109/JSAC.2011.111002

[18] Whay C. Lee. 1995. Topology aggregation for hierarchical routing in ATM networks. *Computer Communication Review* 25, 2 (1995), 82–92. https://doi.org/10.1145/210613.210625

[19] J. Pei, P. Hong, K. Xue, and D. Li. 2018. Resource Aware Routing for Service Function Chains in SDN and NFV-Enabled Network. *IEEE Transactions on Services Computing* (2018), 1–1.

[20] C Pham, N Tran, S Ren, W Saad, and C Hong. 1 Jan.-Feb. 2020. Traffic-Aware and Energy-Efficient vNF Placement for Service Chaining: Joint Sampling and Matching Approach. *IEEE Transactions on Services Computing* 13, 1 (1 Jan.-Feb. 2020), 172–185. https://doi.org/10.1109/TSC.2017.2671867

[21] Chuan Pham, Nguyen H. Tran, Shaolei Ren, Walid Saad, and Choong Seon Hong. 2017. Traffic-aware and Energy-efficient vNF Placement for Service Chaining: Joint Sampling and Matching Approach. *IEEE Transactions on Services Computing*

(feb 2017), 1–1. https://doi.org/10.1109/tsc.2017.2671867

[22] Gang Sun, Yayu Li, Dan Liao, and Victor Chang. 2018. Service function chain orchestration across multiple domains: A full mesh aggregation approach. *IEEE Transactions on Network and Service Management* 15, 3 (sep 2018), 1175–1191. https://doi.org/10.1109/TNSM.2018.2861717

[23] X. Wang, X. Li, S. Pack, Z. Han, and V. C. M. Leung. 2020. STCS: Spatial-Temporal Collaborative Sampling in Flow-Aware Software Defined Networks. *IEEE Journal on Selected Areas in Communications* 38, 6 (2020), 999–1013. https://doi.org/10.1109/JSAC.2020.2986688

[24] Jin Y. Yen. 1971. Finding the K Shortest Loopless Paths in a Network . *Management Science* 17, 11 (jul 1971), 712–716. https://doi.org/10.1287/mnsc.17.11.712

[25] Zijun Zhang, Zongpeng Li, Chuan Wu, and Chuanhe Huang. 2017. A Scalable and Distributed Approach for NFV Service Chain Cost Minimization. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2151–2156. https://doi.org/10.1109/ICDCS.2017.210

[26] D. Zheng, C. Peng, X. Liao, L. Tian, G. Luo, and X. Cao. 2020. Towards Latency Optimization in Hybrid Service Function Chain Composition and Embedding. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 1539–1548. https://doi.org/10.1109/INFOCOM41043.2020.9155529

[27] Zhi Zhou, Qiong Wu, and Xu Chen. 2019. Online Orchestration of Cross-Edge Service Function Chaining for Cost-Efficient Edge Computing. *IEEE Journal on Selected Areas in Communications* 37, 8 (aug 2019), 1866–1880. https://doi.org/10.1109/JSAC.2019.2927070