Computer Networks 212 (2022) 109044

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Distributed federated service chaining: A scalable and cost-aware approach for multi-domain networks

Chen Chen [a], Lars Nagel [a], Lin Cui [b], Fung Po Tso [a,*]

[a] Department of Computer Science, Loughborough University, Loughborough, UK
[b] Department of Computer Science, Jinan University, Guangzhou, China

ARTICLE INFO

Keywords:
Multi-domain
Distributed orchestration
Service function chaining

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 Distributed Federated Service Chaining (DFSC), a framework for orchestrating and maintaining SFC placement in a distributed fashion while sharing only a minimal amount of domain information and control. First, a deployment cost minimization problem is formulated as an Integer Linear Programming (ILP) problem with fine-grained constraints for location and hardware dependencies. We show that this problem is NP-hard. Then, a placement algorithm is devised to use information only on inter-domain paths and border nodes. Our extensive experimental results demonstrate that DFSC efficiently optimizes the deployment cost, supports domain autonomy and enables faster decision-making. The results also show that DFSC finds solutions within a factor 1.15 of the optimal solution on average. Compared to a centralized approach in the literature, DFSC reduces the deployment cost by up to 20% and uses 70% less decision-making time.

1. Introduction

In recent years, due to the rise of edge computing and the Internet of Things (IoT), there is a significant increase of diverse applications such as augmented reality, gaming, face recognition [1–3]. These applications include not only traditional network functions such as firewall and load balancer but also machine learning components [4]. In practice, Service Function Chaining (SFC) has been introduced as a promising approach to deliver these complex end-to-end applications [5]. The deployment of service function chains consumes computing and network resources (e.g., CPU, memory, bandwidth) which significantly affect the revenue of service providers [6–8].

Service Providers are shifting network services from centralized clouds to distributed edge networks. When more and more edge service operators join the market, the network is becoming increasingly more fragmented across different administrative domains [9]. For this reason, we envision that future network services will span across public cloud data centers, ISP networks and edge networks. Fig. 1 shows an example with three administrative domains consisting of public clouds, ISP clouds and edge clouds. In this environment a jaywalk detection system is installed in form of a service function chain. Its purpose is to detect where people jaywalk and thus identify places where new crosswalks should be installed [10].

Realizing such applications requires a high degree of resource sharing and coordination across multiple administrative domains. However, existing SFC schemes are unable to support those because they either focus on a centralized resource orchestration [11–13] or only consider inter-cloud collaboration [14,15], both of which assume that all the topological and resource information are known and that a centralized orchestrator has control and visibility of all the domains. But network operators are known for restricting the exchange of detailed network information such as topology, resources and services [16,17] as doing so could mean revealing their own business-sensitive competitive advantages. As a result, network operators are likely to prefer managing their administrative domains using their own orchestrators.

Different management policies and the lack of detailed intra-domain information from other operators exacerbate the problem of placing service function chains in multi-domain networks. Existing works would

* Corresponding author.
E-mail address: p.tso@lboro.ac.uk (F.P. Tso).

https://doi.org/10.1016/j.comnet.2022.109044
Received 14 December 2021; Received in revised form 20 April 2022; Accepted 10 May 2022
Available online 18 May 2022

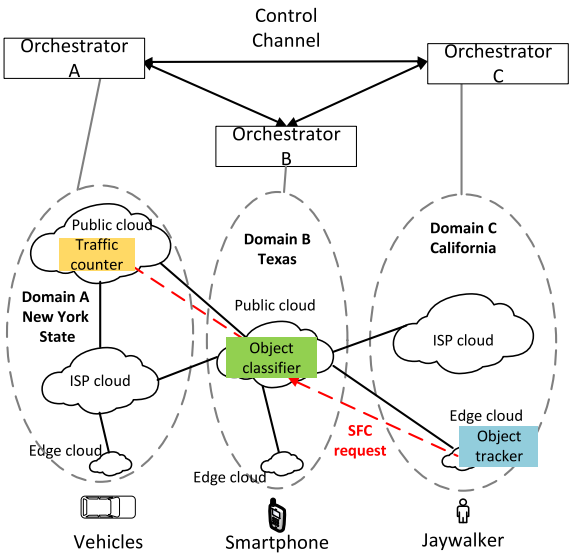


Fig. 1. An example for SFC in 3-tier (cloud-ISP-edge) architecture.

fail to find hosting nodes and routing paths for a chain because they would not have access to topology and resource information in other domains. When the network scale increases, centralized and inter-cloud based solutions may significantly prolong the convergence time as the search space increases exponentially. Orchestrating different domains such as public data centers, edge clouds and customer premises can be very complex due to the sheer volume of incoming requests and computational resources. A centralized approach lacks the ability to scale to such magnitudes.

Another problem is that certain network functions (NFs) are location and hardware dependent. In the scenario of Fig. 1, for example, the object tracker must be placed at Domain California if jaywalking is to be researched in California. Hence, there is a need to specify the domain. The object classifier, which is to recognize jaywalkers and cars, is likely to require GPUs for supporting a deep neural network (DNN). A cloud can be equipped with specialized hardware such as GPUs, which is why the object classifier should be placed at a public cloud rather than an edge cloud. Hence, there is also a need to specify the type of a cloud (i.e., public cloud, ISP cloud and edge cloud) which we will refer to as "tier" in this paper.

To overcome these challenges, we propose a federated SFC scheme which is scalable and preserves autonomy and privacy of each participating domain. It achieves this by distributing the decision-making process among multiple relevant domains along candidate paths through an aggregated network. This way, each domain can use its own placement algorithm (e.g., Algorithm 2) suited to its policies and resource capacities, and take full advantage of the underlying infrastructure. Domain privacy is preserved by using a minimal amount of network information. To the best of our knowledge, this is the first paper to jointly consider domain autonomy, privacy and scalability in the SFC deployment problem.

Furthermore, we formulate the SFC placement problem as an optimization problem aiming to minimize the monetary cost of using computing resources and traffic routing. As constraints we specify the affinity correlations of an SFC request with domains and tiers, respectively. We propose a placement algorithm called Distributed Federated Service Chaining (DFSC) which employs a k-shortest path algorithm [18] in an aggregated network graph. It therefore merely requires information about inter-domain paths and border nodes which can be obtained by using the Border Gateway Protocol (BGP). As illustrated in Fig. 1, the SFC request is partitioned into sub-requests by the ingress orchestrator (here: Orchestrator C) based on the aggregated graph. Then the sub-requests are assigned to Orchestrators A, B and C, respectively, which handle them within their own domains privately and autonomously. This has also the advantage that the process is parallelized and the decision-making time is significantly reduced.

We conducted extensive experiments in Network Simulator 3 [19] as well as in Mininet [20] and compared our scheme to the SFCO-AMD (SFC Orchestration Across Multiple Domains) approach [12] and the DistNSE (Distributed Network Service Embedding) approach [17]. SFCO-AMD and DistNSE are selected and implemented to demonstrate the efficiency of our distributed framework in terms of cost reduction and decision-making time. To further validate the proposed scheme and determine the optimality gap, we also compared our results to the optimum computed by the Gurobi solver [21]. Gurobi is a mathematical programming solver for linear programming, quadratic programming, integer linear programming and etc. The experiments demonstrate that the DFSC algorithm is within a factor 1.15 of the optimal solution on average while being several orders of magnitude faster than Gurobi. Compared with the SFCO-AMD and DistNSE approach, DFSC reduces the overall deployment cost (i.e., CPU cost, memory cost and traffic routing cost) by up to 20% and uses 70% less decision-making time on average.

In short, our contributions are as follows.

- An Integer Linear Programming (ILP) problem is formulated aiming to minimize the deployment cost incurred by CPU, memory and bandwidth usage. Constraints of this optimization problem include location and hardware dependencies that allow for meeting application requirements and specific network policies.
- We propose a distributed framework that distributes the decision-making process, including NF assignment and flow routing, among multiple orchestrators. Our placement scheme DFSC only requires the information of border nodes and inter-domain links. This preserves domain autonomy and privacy.
- In extensive experiments, DFSC is evaluated by comparing it with the optimal result as well as two existing works. It is shown that DFSC achieves less overall costs and a lower decision-making time than the existing works in two testbeds (i.e., NS3 and Mininet).

This work is an extension of our earlier investigation into SFC placement across multiple domains [22]. Rather than specifying domain and tier constraints for the whole chain, the new work allows different constraints for every network function. In other words, every network function carries a pair of domain and tier constraints. This enables tailored policies for network operators to assign a particular network function to a particular domain or tier. Moreover, we implemented the modified DFSC approach not only in the trace-driven NS3 simulator, but also in the network emulator Mininet. This allowed us to compare the NS3 results with those of Mininet which gives additional evidence that DFSC scales well and significantly reduces the decision-making time.

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

The domain autonomy and detailed network information are essential for SFC orchestration. Owing to the ever-increasing network domains and the proprietary privacy policies, it is prone to fail to get such detailed network information and the full control of all domains. Hence, the complexity of multi-domain management has significantly increased. Much of the existing approaches have partially considered the domain autonomy, privacy or scalability issues and proposed several solutions, aiming to optimize different metrics such as delay, throughput and cost. In this section, we summarized the related approaches that are applicable to the multi-domain environments.

2.1. Centralized SFC orchestration

Much of the literature proposed a centralized approach by considering different objectives such as optimal placement in light of end-to-end latency, efficient resource allocation and deployment cost.

Atefeh et al. [23] investigate the VNF placement problem for inter-DC elastic optical networks. This work aims to minimize the overall network cost with respect to the balanced resource utilization. However, this work largely ignores the domain autonomy and privacy for inter-DC networks. Toumi et al. [24] propose a deep reinforcement learning approach to solve the SFC placement for multi-domain networks. The proposed approach uses the linear physical programming to create rewards with respect to cost and latency. Nina et al. [25] study the service deployment problem across distributed edge clouds for a dynamic and mobile scenario. The proposed algorithm uses a multi-tier orchestration scheme, localized management scheme and protocols to federate multiple edge clouds. Sun et al. [12] have 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 resource availability information. Indeed, this work takes the multi-domain scenario into consideration. However, sharing such information remarkably compromises the privacy of domains. Joshi et al. [26] presented pSMART, a privacy-preserving SFC orchestration in the multi-domain environment. pSMART employs a binary query response method to coordinate multiple underlying domains. This work addresses the privacy issue, but it still requires the full control of all domains. Multi-SFC [27] studies the multi-domain SFC placement problem by enabling multiple NFV platforms. Multi-SFC uses the SFC segment, in which NFs are connected within a single domain/cloud/-platform. Gouareb et al. [11] 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 [28]. The author formulated an objective function to minimize service cost and resource consumption. They modeled 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. Xiang et al. [29] proposed an efficient cross-domain query algorithm to preserve the privacy of domains. By abstracting the resource availability into resource state, only minimal network information is exposed. To solve the multi-domain SFC problem, Bhamare et al. [30] proposed an ILP problem aiming to minimize inter-cloud traffic and response time. Furthermore, they compared the results with a simple greedy approach. Ghaznavi et al. [31] proposed an approach to place NF instances in multiple hosts. Indeed, the traffic workload is distributed among the instances that network resources are utilized more efficiently. However, the approach still employs a centralized orchestrator.

These approaches rely on global information visibility and full control of the multiple domains by employing a global centralized orchestrator. Therefore, such approaches are not directly applicable for the multi-domain SFC problem.

2.2. Distributed SFC orchestration

There are also a few works employing distributed approaches. Esposito et al. [14] proposed the Necklace architecture 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 every single node. Also, sharing the same host node among multiple SFCs leads to a slower convergence time which negatively impacts the decision-making time. Godfrey et al. [32] leverage a pre-computed set of nodes to exchange messages in a distributed manner and, hence, the communication overheads between multiple orchestrators are reduced. Lin et al. [33] propose a novel optimization algorithm based on column generation method to tackle the SFC placement problem for multi-domain networks. The proposed algorithm requires no intra-domain information such as domain topology and network resources. Liu et al. [34] propose a distributed approach to guarantee fair competition among multiple domains. Also, the approach balances the load by migrating the deployed SFs. Huang et al. [35] reported a scalable approach via federated reinforcement learning. This work employs federated learning into the global model training with the available resources. Zhang et al. [15] proposed a distributed approach by partitioning a cost minimization problem into multiple subproblems. However, this work fails to take multi-domain into consideration and, hence, cannot handle the inter-domain communication and information exchange. Antevski et al. [36] proposed an approach to federate multiple domains by employing distributed ledger technologies. However, it takes around 5 s for a single service federation without considering the deployment time. In [17], Ahmed proposed DistNSE, a distributed framework for service chain embedding across multi-domains. DistNSE employs a bidding mechanism to partition SFCs. However, DistNSE chooses to enumerate all the paths between the peering nodes in each domain to find an optimal mapping solution. Therefore, this approach requires excessive time to make decisions which significantly influences the scalability.

Although these approaches pave the way towards flexible and scalable SFC placement, they still treat the domain autonomy, privacy and scalability separately. Unlike aforementioned works, the design of DFSC jointly considers these factors and provides fast decision-making and good scalability for large-scale networks.

3. System model

This section provides more details about the network and service chain model considered for the distributed federated service chaining.

3.1. Physical network

The network is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_V\}$ is the set of all nodes and $\mathcal{E} = \{e_1, e_2, \ldots, e_E\}$ is the set of all edges or network links. 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 node v, there are several fields $\{v.cpu, v.mem, v.domain, v.tier\}$, $v.cpu$ and $v.mem$ denote the maximum CPU and memory capacity of node v. We use $v.domain$ to denote the domain id of the node, and $v.tier$ to denote the tier of the node (i.e., tier 1, tier 2 and tier 3). Each link in $\mathcal{E}$ has a capacity of network bandwidth. By $\mathcal{P} = \{p_1, p_2, \ldots, p_P\}$ 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 a 7-tuple $r = \{src, dst, \mathcal{N}, \Psi^{tr}, l^{td}, T_t, D_d\}$ where src and dst are the ingress and egress node and $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$ is the set of NFs in the 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^{tr} denotes the required traffic rate, and l_{td} denotes the maximum tolerated delay. D_d and T_t are the set of domain and tier constraints, respectively.

Table 1
Symbols and Variables.

Symbols and Variables	Description
Physical network	
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Physical network graph.
$\mathcal{V} = \{v_1, v_2, \dots, v_V\}$	All nodes in the network.
$\mathcal{E} = \{e_1, e_2, \dots, e_E\}$	All physical links in the network.
$\mathcal{P} = \{p_1, p_2, \dots, p_P\}$	All paths in the network.
C_v^{cpu}, C_v^{mem}	Remaining CPU and memory capacity in node v.
C_p^{bw}	Remaining bandwidth capacity on path p.
SFC graph	
src, dst	Ingress and egress point of the SFC.
$\mathcal{N} = \{n_1, n_2, \dots, n_N\}$	All NFs in the SFC request.
$c_{n_i}^{cpu}$, $c_{n_i}^{mem}$	Amount of CPU and memory required by NF n_i.
Ψ^{tr}	Required traffic rate.
l^{td}	Maximum tolerated delay.
l^{d}	End-to-end delay of a given SFC.
$D_d^{n_i}$	The domain constraint.
$T_t^{n_i}$	The 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	
$y_v^{n_i}$	Indicator if NF n_i is hosted on node v.
$y_p^{n_i,n_j}$	Indicator if traffic from NF n_i to n_j is routed through path p.

3.3. SFC request affinity

As aforementioned, network operators could have demands for specific constraints on the domain and the tier due to their operational concerns such as specific hardware or location. We define domain and tier constraints by employing the idea of affinity or anti-affinity rules as follows.

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

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

The conference version provides coarse-grained domain and tier constraints for the SFC granularity. In other words, every SFC request carries a pair of domain and tier constraints. Every network function in the request shares the same constraint. Unlike the conference version, in this paper, we consider the granularity of network functions to enable flexible and fine-grained orchestration. Every network function carries a pair of domain and tier constraints.

4. Problem description

In geo-distributed multi-domain networks, service providers search for solutions to deploy SFC requests under constraints. The selection of the nodes 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, there will be resource costs caused by consuming computing resources in clouds, and traffic routing costs incurred by the usage of bandwidth while steering traffic through the chain.

Given an SFC request, our goal is to minimize the overall deployment cost. Let C be the deployment cost per second, C_R the resource cost per second, and C_T the traffic routing cost per second. Then the goal is to minimize the overall deployment cost.

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

We assume that these costs are constant during the time of deployment.

4.1.1. Resource cost

For the resource costs of an SFC request, we consider CPU and memory. The resource cost C_R is defined as

$$C_R = \sum_{n_i \in \mathcal{N}} \sum_{v \in \mathcal{V}} (\beta_d^{cpu} c_{n_i}^{cpu} y_v^{n_i} + \beta_d^{mem} c_{n_i}^{mem} y_v^{n_i}) \tag{4}$$

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, $y_v^{n_i}$ is an indicator variable which is 1 if NF n_i is deployed on node v, 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 usage of network links [37,38]. 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 cost of routing one unit of traffic through 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. $y_p^{n_i,n_j}$ is an indicator variable which is 1 if path p is used between NFs n_i and n_j, and 0 otherwise. In particular, 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.

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

The problem is subject to several constraints. The CPU and memory requirements must not exceed the remaining CPU capacity C_v^{cpu} and memory capacity C_v^{mem} on the node v.

$$\sum_{n_i \in \mathcal{N}} c_{n_i}^{cpu} y_v^{n_i} \leq C_v^{cpu}, \quad \forall v \in \mathcal{V} \tag{7}$$

$$\sum_{n_i \in \mathcal{N}} c_{n_i}^{mem} y_v^{n_i} \leq C_v^{mem}, \quad \forall v \in \mathcal{V} \tag{8}$$

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

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

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

$$\sum_{v \in \mathcal{V}} y_v^{n_i} \leq 1 \tag{10}$$

The end-to-end delay l^d of the path for SFC request must meet the constraint of the maximum tolerated delay l^{td}.

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

Finally, we phrase the domain and tier constraints. If NF n_i includes a tier affinity constraint specifying tier t, n_i must be deployed at tier t.

$$if \quad T_t^{n_i} = 1, \quad \sum_{v \in \mathcal{V}^t} y_v^{n_i} = 1 \tag{12}$$

where $\mathcal{V}^t$ is the set of nodes belonging to tier t.

If NF n_i includes a tier anti-affinity constraint specifying tier t, n_i must not be deployed at tier t.

$$if \quad T_t^{n_i} = -1, \quad \sum_{v \in \mathcal{V}^t} y_v^{n_i} = 0 \tag{13}$$

If NF n_i includes a domain affinity constraint specifying domain d, n_i must be deployed at domain d.

$$if \quad D_d^{n_i} = 1, \quad \sum_{v \in \mathcal{V}^d} y_v^{n_i} = 1 \tag{14}$$

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

If NF n_i includes a domain anti-affinity constraint specifying domain d, n_i must not be deployed at domain d.

$$if \quad D_d^{n_i} = -1, \quad \sum_{v \in \mathcal{V}^d} y_v^{n_i} = 0 \tag{15}$$

Problem 1 can be proven to be NP-hard. For this we demonstrate that the Multiple Knapsack Problem (MKP), which is known to be NP-hard [39], can be reduced to this problem. The input of the MKP problem is a set $\mathcal{M}$ of items, where the weight of item $m_i \in \mathcal{M}$ are $m_i.weight$. A set $\mathcal{K}$ of knapsacks, where the capacity of $k_j \in \mathcal{K}$ are $k_j.cap$. The profit of mapping m_i to k_j is f_{ij}. The objective of the MKP problem is to maximize the overall profit.

For the reduction, we map each item m_i to an NF n_i with $m_i.weight = c_{n_i}^{cpu}$. Each knapsack k_j is considered as a node v_j with $k_j.cap = v_j.cpu$. Then, we assume the memory resources are infinite on the node v_j. Consider the special case that each SFC consists of only one NF. The profit f_{ij} is the negative of the deployment cost of assigning n_i to v_j. Assume that there are sufficient nodes to host all NFs, meaning that no nodes are overloaded. Thus, the MKP problem becomes a special case of Problem 1. The objective is to minimize the overall deployment cost, or maximize the total profit. Therefore, the MKP problem can be reduced to the SFC placement problem and, hence, the SFC placement problem is NP-hard. Similarly, we could also assume $k_j.cap = v_j.mem$ and the CPU resources are infinite on the node v_j. In this case, the MKP problem is still reducible to the Problem 1.

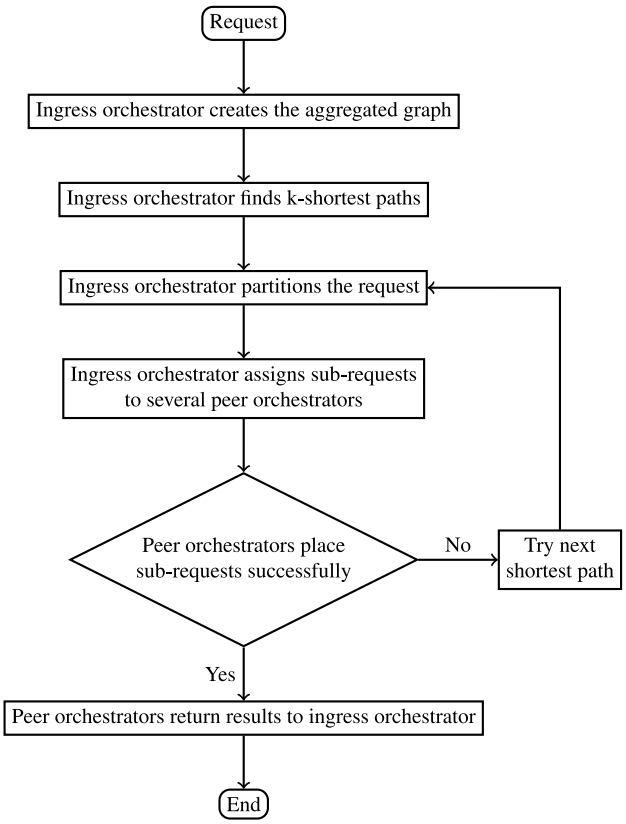
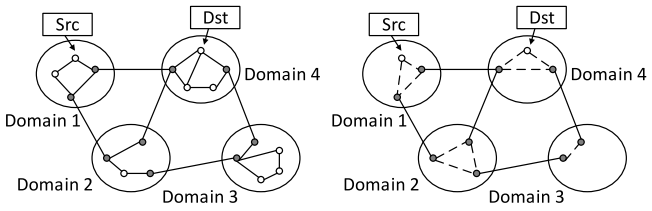


Fig. 2. The workflow of DFSC.

5. Distributed federated service chaining

In this section, we elaborate on the distributed federated service chaining algorithm. The proposed algorithm is designed to find solutions for Problem 1. This algorithm aims to distribute the decision-making process to multiple domain orchestrators and preserve the autonomy of domains. A global orchestrator is not required. The required information of the proposed algorithm includes inter-domain links and border nodes. Fig. 2 shows the workflow of our proposed algorithm. In this work, we assume that each domain has a domain orchestrator which is aware of all the intra-domain information but has only limited information about other domains. When an SFC request arrives at a domain, the ingress orchestrator partitions the request into sub-requests based on domain constraints, the resource cost parameters β_{cpu}^d and β_{mem}^d. The network functions in the request are assigned one by one to the domain with the lowest resource cost which fulfills the domain constraints. Every sub-request is the set of network functions assigned to a domain. Then, sub-requests are delivered to several peer orchestrators. In the example shown in Fig. 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, Orchestrators A and B are referred to as peer orchestrators.

The ingress orchestrator builds an aggregated graph including inter-domain links and aggregated nodes by employing the full-mesh aggregation [40]. Then, the ingress orchestrator employs the k-shortest path algorithm [18] in the aggregated graph and decides how to assign the sub-requests to different domains. The k-shortest path algorithm provides several candidate paths, resulting in exploiting path diversity at the cost of only a slight increase in decision-making time. Finally, every 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.



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

Fig. 3. Topology aggregation.

5.1. Constructing aggregated graph

Privacy is always a paramount concern in multi-domain networks [26]. Hence, we assume that only border nodes (nodes connected to an inter-domain link) are visible to other peer orchestrators. The boundary of each domain consists of the border nodes in a single domain. This can be achieved by the Border Gateway Protocol [17]. In Fig. 3(a), gray nodes represent the border nodes, and the link that traverses two domains is an inter-domain link. Thus, the intra-domain network connectivity and topology are regarded as confidential information of each domain. In this setup, domain internal information is well preserved.

The aggregated graph consists of inter-domain links, border nodes and logical intra-domain links. As shown in Fig. 3(b), the gray nodes represent border nodes. The boundary of every domain is the set of border nodes that located at the edge of the domain. Then, the solid and dotted lines denote the inter-domain and logical links, respectively. The white nodes represent the ingress and egress nodes of the SFC request. The aggregated graph is derived from the physical network by employing full-mesh aggregation. 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 the logical link are temporarily set to 0 in the aggregation phase because such information is regarded as confidential. In the aggregation graph, there is enough information for the ingress orchestrator to make the first-phase decision.

5.2. Distributed federated service chain placement

In this subsection, we explain that how to find solutions with the distributed framework. The pseudo code of DSFC algorithm is shown in Algorithm 1. First, the aggregated graph is constructed by the ingress orchestrator when an 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 [18]. 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. The DFSC algorithm is initialized in line 1 in Algorithm 1. Then, lines 3–15 and lines 21–26 are executed on the ingress orchestrator while lines 16–20 are 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 lines 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 the 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 returns the result in lines 21–26.

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** $\Psi^{tr} > C_p^{bw}$ **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** $n_i \in \mathcal{N}$ **do**
11: Find the available domain d with the lowest
12: resource cost;
13: **if** d cannot meet the domain constraint for n_i **then**
14: Try next domain;
15: **end if**
16: Assign the NF to d;
17: **end for**
18: **for** each domain d on path p **do**
19: Call **Algorithm 2** on peer orchestrators;
20: **if Algorithm 2** fails **then** break;
21: **end if**
22: **end for**
23: Check the end-to-end delay;
24: **if** All NF are assigned **then**
25: **return** Routing path, hosted nodes, the
26: deployment cost;
27: **end if**
28: **end while**
29: **return** Deployment failure.

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** $\Psi^{tr} > C_p^{bw}$ **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: **if** v cannot meet the tier constraint for n_i **then**
11: Try next node on p;
12: **end if**
13: Assign the NF to v;
14: **end for**
15: **if** all NFs are assigned **then**
16: **return** p^d and $\mathcal{V}^h$;
17: **end if**
18: **end while**
19: **return** Intra-domain deployment failure.

Time complexity: In algorithm 1, line 4 runs the k-shortest path algorithm which has the time complexity of $O(k \cdot |\mathcal{V}^a|^3)$ [18], where k is the number of paths, and $\mathcal{V}^a$ is the cardinality of the aggregated nodes in the graph. Line 5 has the time complexity of $O(logk)$ because of linear time loops. Line 10 has the time complexity of $O(n)$ due to linear time loops. Line 16 runs a loop over traversed domains whose

time complexity is $O(|d|)$. As a result, the time complexity of algorithm 1 is $O(k \cdot |\mathcal{V}^a|^3)$.

Algorithm 2 describes the intra-domain deployment which is simultaneously executed on each peer orchestrator. In line 3, k-shortest path algorithm is employed to find candidate paths within a single domain. Subsequently, in lines 4–11 DFSC searches for the first available cloud to host the NF. Meanwhile, the tier and resource constraints are checked. When all assigned NFs in this domain are 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 messages to the ingress orchestrator.

Time complexity: In algorithm 2, line 3 runs the k-shortest path algorithm which has the time complexity of $O(k \cdot |\mathcal{V}^d|^3)$ [18], where k is the number of paths, and $\mathcal{V}^d$ is the cardinality of the nodes in domain d. Line 4 has the time complexity of $O(logk)$. Line 8 has the time complexity of $O(|N^d|)$ because of linear time loops. Hence, the time complexity of algorithm 2 is $O(k \cdot |\mathcal{V}^d|^3)$.

5.3. Service embedding protocol

To exchange the deployment information, we devise a communication protocol with two messages sent and received among the orchestrators.

The ingress orchestrator assigns sub-requests to peer orchestrators by sending the *subchain_msg* messages. The message contains the assigned sub-chain, the path obtained from k-shortest paths, the required bandwidth and the tier constraints. Finally, we use the Pickle package to serialize the message into a text-based file named *sub_chain.txt*.

```
import pickle
#sub-chain message
def subchain_msg(self, sub_chain, path, required_bw,
    tier_con):
   subchain_msg = [sub_chain, path, required_bw,
       tier_con]
   with open('dfsc/sub_chain.txt', 'wb') as fp:
      pickle.dump(subchain_msg, fp)
```

After intra-domain deployment, the peer orchestrator sends the result back to the ingress orchestrator with the message *subchain_result_msg*. This message contains two variables. *succ_flag* indicates whether the intra-domain deployment succeeds. *sub_result* contains the host nodes and the routing path in this domain. Finally, we use the Pickle package to serialize the message into a text-based file named *sub_chain_result.txt*.

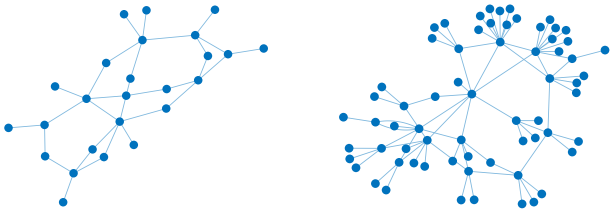
```
import pickle
#sub-chain deployment result
def subchain_result_msg(self, succ_flag, host_node,
    route_path):
   sub_result = [host_node, route_path]
   sub_result_msg = [succ_flag, sub_result]
   with open('dfsc/sub_chain_result.txt', 'wb') as fp:
      pickle.dump(sub_result_msg, fp)
```

6. Evaluation

In this section, we conducted trace-driven simulations and emulations to evaluate the benefits of the proposed DFSC approach. The experiments use real-world NF instance prices.

Table 2
Parameter settings.

Description	Values
Number of domains for Agis	5
Number of domains for Internode	9
CPU capacity in cloud	1000
CPU capacity in ISP cloud	500
CPU capacity in edge cloud	200
Memory capacity in cloud	2000 GB
Memory capacity in ISP cloud	1000 GB
Memory capacity in edge cloud	400 GB
Bandwidth capacity in inter-domain link	1000 Mbps
Bandwidth capacity in intra-domain link	2000 Mbps
Link delay	[2,5] ms
Description	**Values**
CPU cost parameter	[0.001, 0.005] $/s
Memory cost parameter	[0.001, 0.004] $/GBs
CPU demand of NF n	[1,10]
Memory demand of NF n	[3, 30] GB
Traffic rate requirement	[100, 500] kbps
Traffic routing cost parameter	[0.02, 0.05] $/Mb
Maximum tolerated delay	[0, 100] ms



(a) Topology Agis (b) Topology Internode

Fig. 4. Topologies in the experiment.

6.1. Experimental setup

6.1.1. Infrastructure

We implemented DFSC and SFCO-AMD in the Network Simulator 3 (NS3) and the Mininet emulator. In contrast to NS3, Mininet uses real CPU, memory and a real hardware clock which provides a deeper insight into the decision-making time. Note that many network testbeds such as [41] can also be used to implement our algorithms. Also, by analyzing the performance difference of NS3 and Mininet, we could comprehensively verify the performance of DFSC. The experiments are conducted on a server with 105 GB RAM and an Intel(R) Xeon(R) E5645 processor with 24 cores. We use two US backbone network topologies from the Internet Topology Zoo [42] which are depicted in Fig. 4. We use them because they include backbone, customer and transit networks. We divide the Agis and Internode topology into 6 and 9 domains, respectively

Domain orchestrators are implemented as applications both in NS3 and Mininet. In NS3, we first implement an "orchestrator" application. Then, we create orchestrator nodes based on the number of domains. Each orchestrator runs the application. These applications exchange messages via C++ variables. In Mininet, orchestrators are implemented as Python scripts. We also create orchestrator nodes based on the number of domains. Then, each node runs an orchestrator script. The messages among orchestrators are serialized into text-based meta files. Then, other orchestrators read the messages from these meta files. This is achieved by using the Pickle package of Python.

6.1.2. Real cost data

The cost settings are selected from a set of price ranges based on Amazon instance prices [43]. The cost for one CPU per second ranges from 0.001$ to 0.005$. The cost for one GB of memory per second

Table 3
Processing time of VNFs.

VNF	Processing time
Firewall	120 μs
IDS	160 μs
Caching	83 μs
Flow monitor	200 μs
Load balancer	647.5 μs
WAN-opt.	200 μs

ranges from 0.001$ to 0.004$. The cost for sending one Mb of data over one link ranges from 0.02$ to 0.05$. Moreover, the maximum tolerated end-to-end delay is randomly selected between 0 ms and 100 ms.

6.1.3. System parameter

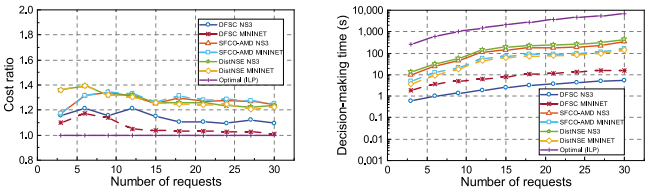
All the parameter settings are listed in Table 2. The SFC requests are derived from .pcap files of the CAIDA traces [44]. The source and target nodes of an SFC request are uniformly selected at random from the set of nodes in the network. The chain length, that is the number of NFs in a request, varies from 3 to 6. The number of cores in one cloud, ISP cloud and edge cloud is 1000, 500, and 200, respectively. The memory capacity of one cloud, ISP cloud and edge cloud is 2000 GB, 1000 GB and 400 GB, respectively. The bandwidth capacities for inter-domain and intra-domain links are set to 1000 Mbps and 2000 Mbps, respectively. The propagation delay is randomly selected between 2 ms and 10 ms for every link. For each NF, the number of CPU cores is randomly selected from {1, 2, …, 10}. The memory requirement in GB is randomly selected from {1, 2, …, 20}. Then, we set up some simulation parameters according to some similar papers [45,46]. Also, the domain and tier constraints are randomly created. The value k for k-shortest path is set to 8 because of empirical studies. The reason behind this setup is that acceptance does not significantly increase when the value of k is greater than 8. Each SFC request contains of a varying number of network functions. The network functions are randomly chosen from a set of network functions from Table 3. The processing time of network functions conforms to commercial instances as given in [47].

6.1.4. Benchmarks

To benchmark the performance of DFSC with the optimal solution, we use an ILP solver Gurobi to obtain the offline optimum of the cost minimization problem by solving Problem 1. We first compare DFSC, SFCO-AMD and DistNSE with the optimum to study the optimality gap in the topology Agis. Then, we compare DFSC with SFCO-AMD and DistNSE in the topology Internode. SFCO-AMD partitions the SFC requests into sub-requests by using a centralized orchestrator. Then, sub-requests are deployed in different domains. DistNSE uses a bidding mechanism in a distributed manner, aiming to optimize the deployment cost.

6.2. Evaluation results for the topology Agis

In this subsection, we compare the performance of optimum, DFSC, SFCO-AMD and DistNSE to study the optimality gap in the topology Agis as shown in Fig. 4(a). The topology consists of 25 nodes and 30 links. We divide the nodes into 6 domains based on the geo-distance. Then we randomly created different tiers for nodes. To evaluate the performance of Algorithm 1, we compare it with the optimal solution from Gurobi. Specifically, given an SFC request, the solver searches for hosting clouds and traffic routing paths that minimize the cost. In this experiment, we test a different number of SFC requests ranging from 3 to 30 with a step size 3. Subsequently, we calculate the average cost per request. Finally, we compare the optimum with DFSC and SFCO-AMD by plotting the cost ratio of the overall deployment cost.



(a) The cost ratio (b) The decision-making time

Fig. 5. Comparison of the cost ratio and decision-making time for the topology Agis.

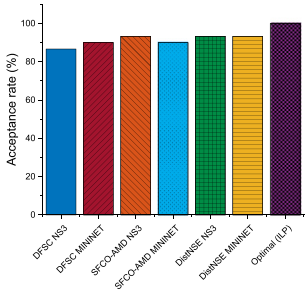
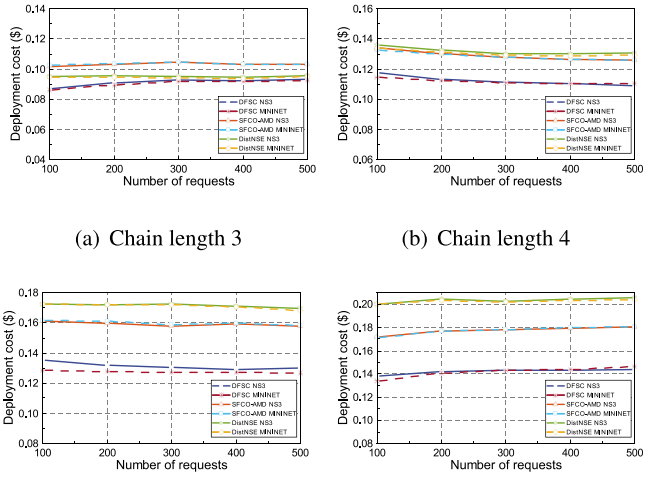


Fig. 6. The acceptance rate for the topology Agis.



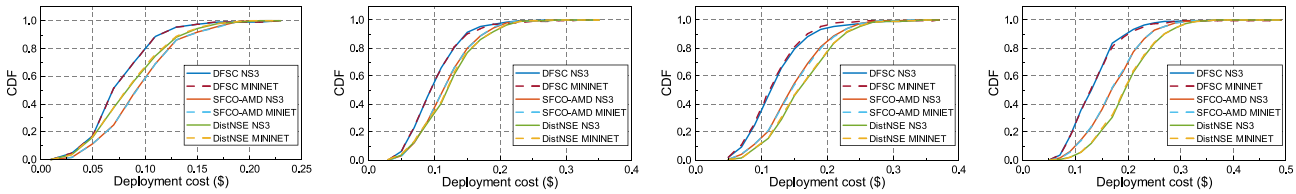
(a) Chain length 3 (b) Chain length 4

(c) Chain length 5 (d) Chain length 6

Fig. 7. Number of requests vs. deployment cost for the topology Internode.

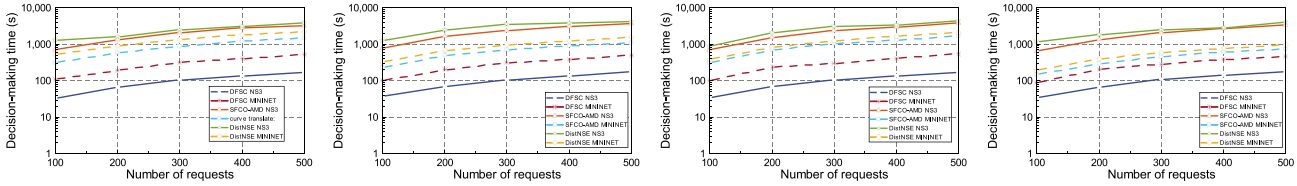
6.2.1. Efficiency of the deployment cost for the topology Agis

We first examine the cost ratio achieved by DFSC, SFCO-AMD and DistNSE against the optimum, which is the ratio of the deployment cost achieved by DFSC, SFCO-AMD or DistNSE to that achieved by the offline minimum cost of the ILP solver. Fig. 5(a) demonstrates the cost ratio of DFSC, SFCO-AMD and DistNSE over a varying number of requests. Our key observation is that: (1) The DFSC achieves only 15% more deployment cost on average than the optimum, demonstrating the effectiveness of our proposed algorithm. (2) DFSC always outperforms SFCO-AMD and DistNSE from state-of-the-art literature. (3) As the number of requests increases, the cost ratio of DFSC only varies slightly, indicating that DFSC is very stable against a varying number of requests. (4) The results from NS3 and Mininet show similar trends, verifying the stable performance of DFSC. The cost of NS3 and Mininet for all algorithms are slightly different in some cases. The rationale is that the end-to-end latency can be different in NS3 and Mininet. Consequently, the placement decision in NS3 and Mininet can be different as the end-to-end latency is a constraint in Algorithm 1.



(a) Chain length 3 (b) Chain length 4 (c) Chain length 5 (d) Chain length 6

Fig. 8. CDF of deployment cost for the topology Internode.



(a) Chain length 3 (b) Chain length 4 (c) Chain length 5 (d) Chain length 6

Fig. 9. Number of requests vs. decision-making time for the topology Internode.

6.2.2. Time efficiency for the topology Agis

We continue to analyze the efficiency of DFSC in terms of decision-making time. In this paper, the decision-making time is the time spent by the system to execute the algorithm. Fig. 5(b) shows the total decision-making time over different number of requests. The results are shown in Fig. 5(b), which demonstrates that: (1) Either in NS3 or Mininet, DFSC is at least 70% faster than SFCO-AMD and DistNSE, and at least two orders of magnitude faster than the ILP solver, indicating the effectiveness of DFSC. (2) As the number of requests rises, the decision-making time of DFSC increases slightly while that of SFCO-AMD, DistNSE and ILP solver increase dramatically. This observation suggests that in practice, DFSC can efficiently handle substantial requests.

6.2.3. Acceptance rate for the topology Agis

Then, we compare the acceptance rate of different approaches in Fig. 6. The acceptance rate is the ratio of the number of successfully deployed requests to that of total arrived requests. DistNSE performs the best because DistNSE not only uses all the paths between ingress and ingress pairs, but also allows bid for last selected sub-requests. However, we observe that DFSC is only outperformed by 2%–3% compared with DistNSE. This suggests that DFSC remarkably outperforms the SFCO-AMD in terms of deployment cost and decision-making time at the cost of 2% degradation in the acceptance rate.

6.3. Evaluation results for the topology Internode

In this subsection, we compare the performance of DFSC approach with SFCO-AMD and DistNSE in the Internode topology as shown in Fig. 4(b). The topology consists of 66 nodes, 78 links and 9 domains. Since the ILP solver has difficulties to identify feasible solutions in a reasonable amount of time, we did not implement the ILP solver for the Internode topology.

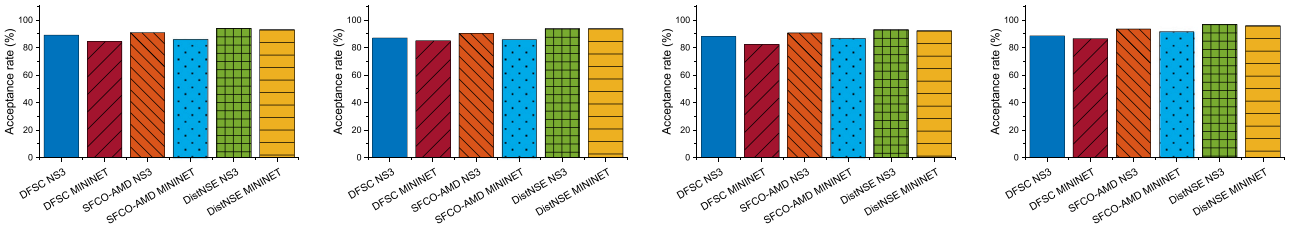
6.3.1. Efficiency of the deployment cost for the topology Internode

Fig. 7 demonstrates the deployment cost of DFSC, SFCO-AMD and DistNSE in NS3 and Mininet. We observe that DFSC always outperforms SFCO-AMD and DistNSE over different chain lengths, demonstrating the efficiency of DFSC. DFSC results in deployment costs in the range of 0.08–0.09$ for chain length 3, 0.11–0.12$ for chain length 4, 0.12–0.13$ for chain length 5, and 0.13–0.14$ for chain length 6. On average, the deployment costs given by DFSC are 12%–20% less than that of SFCO-AMD. Also, DFSC outperforms DistNSE by 20% on average. The rationale is that DFSC simultaneously considers the resource and routing cost. However, DistNSE uses a bid mechanism and only allows bid for last selected sub-requests, which inhibits cost optimization. Similarly, SFCO-AMD partitions the SFC request without comparing the resource cost among domains which increases the deployment cost. Since the DFSC algorithm always outperforms the SFCO-AMD and DistNSE in terms of deployment cost, the effectiveness of our DFSC approach is proved.

Fig. 8 demonstrates the distribution (CDF) of the deployment cost with 500 requests. DFSC NS3 and DFSC Mininet outperform SFCO-AMD and DistNSE in NS3 and Mininet. This implies that DFSC effectively reduces the deployment cost as it strikes a nice balance between the traffic routing cost and resource costs. Also, we observe that the DFSC Mininet distribution only slightly deviates from that of NS3. It suggests that DFSC has stable performance in different testbeds.

6.3.2. Time efficiency for the topology Internode

Then, we investigate the decision-making time achieved by DFSC, SFCO-AMD and DistNSE. Fig. 9 shows the total decision-making time over different number of requests. We observe that DFSC NS3 performs the best followed by DFSC Mininet in every case. Our key observation is that the decision-making time of DFSC increases much slower than that of SFCO-AMD and DistNSE in each case. In NS3, the decision-making time of DFSC ranges from 33 s to 160 s on average. However, the decision-making time of SFCO-AMD ranges from 10 to 50 min on average in NS3. In Mininet, the decision-making time of the proposed DFSC approach ranges approximately from 90 s to 7 min. In contrast, the decision-making time of SFCO-AMD and DistNSE ranges from 5 min to 25 min in Mininet. In other words, DFSC spends only 1.1 s to process one request in the worst case, which suggests that DFSC handles SFC requests timely. This is primarily because DFSC employs a distributed approach to run the process on multiple orchestrators. Another factor is that DFSC employs the k-shortest path algorithm instead of finding all paths between the ingress and egress pair. However, SFCO-AMD and DistNSE strive to find all the candidate paths between ingress and egress pairs which significantly increases the decision-making time. Although DistNSE runs in a distributed manner, it uses a bidding mechanism which also increases the decision-making time. This result indicates that DFSC has better scalability while processing a large number of requests. Another key observation is that the gap between DFSC and SFCO-AMD in Mininet is smaller than that in NS3. The rationale is that, as multiple domain orchestrators communicate with others by



(a) Chain length 3 (b) Chain length 4 (c) Chain length 5 (d) Chain length 6

Fig. 10. Acceptance rate for the topology Internode.

serializing the messages into the disks in Mininet, DFSC consumes more time to make decisions than that in NS3. In NS3, although DFSC was also implemented in a distributed manner, the communication among orchestrators is via global variables in the memory which is faster than via disks. Our observation remains unchanged because DFSC is still 70% faster than SFCO-AMD in Mininet. Overall, the experiments in NS3 and Mininet both confirm the efficacy of DFSC.

6.3.3. Acceptance rate for the topology Internode

Then, we continue to investigate the acceptance rate of DFSC, SFCO-AMD and DistNSE. Fig. 10 demonstrates the acceptance rate over a varying number of chain lengths. The DFSC approach is outperformed by 2%–3% on average. The SFCO-AMD and DistNSE employ an algorithm to find all the candidate paths between source and target pairs, resulting in accommodating more SFC requests. In summary, DFSC achieved a competitive acceptance rate that is 2%–3% less than SFCO-AMD and DistNSE.

Overall, DFSC performs the best in terms of deployment cost. This is because DFSC balances the resource cost and routing cost. DistNSE achieves 20% more deployment cost than DFSC as it only allows bid for last selected sub-requests. Also, DFSC outperforms SFCO-AMD by 12%–20% because SFCO-AMD fails to jointly consider the resource and routing cost. Similarly, DFSC significantly reduces the decision-making time by up to 70% as it runs a k-shortest path algorithm in a distributed manner. SFCO-AMD uses a centralized approach which remarkably increases the decision-making time. Although DistNSE also runs in a distributed manner, it uses a bidding mechanism which increases the decision-making time. Finally, SFCO-AMD and DistNSE outperform DFSC by 2%–3% in terms of acceptance rate because SFCO-AMD and DistNSE both search all the candidate paths in the network, which enables accommodating more SFC requests.

7. Conclusion

In this paper, we study the SFC placement problem with multiple administrative domains aiming to minimize the deployment cost and the total decision-making time. Then, DFSC is proposed to address this problem. DFSC employs a distributed approach to preserve domain autonomy, privacy, and improve the scalability. Instead of collecting network information and requiring the full control of domains, DFSC employs the full-mesh aggregation approach and k-shortest path algorithm to locally generate the placement solution. Therefore, such distributed approach makes DFSC scalable and autonomy-preserving. The extensive results prove that the gap between DFSC and the optimum is 15% on average. In addition, the execution time is several orders of magnitude faster than the ILP solver. Also, the DFSC approach achieves lower deployment costs at the cost of 2%–3% degradation in the acceptance rate. Moreover, DFSC could reduce the decision-making time by up to 70% compared with SFCO-AMD and DistNSE. Hence, the DFSC approach can be employed for cost-aware and scalability-sensitive framework. More other factors can be investigated in the future, e.g., the number of distributed domains.

CRediT authorship contribution statement

Chen Chen: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing. **Lars Nagel:** Conceptualization, Methodology, Supervision. **Lin Cui:** Conceptualization, Methodology. **Fung Po Tso:** Conceptualization, Methodology, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

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 Innovate UK grant 106199-47198; the Chinese National Research Fund (NSFC) No. 62172189 and 61772235; Science and Technology Program of Guangzhou No. 202002030372; Natural Science Foundation of Guangdong Province No. 2020A1515010771. Chen is partially supported by the China Scholarship Council.

References

[1] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, X. Fu, Delay-aware virtual network function placement and routing in edge clouds, IEEE Trans. Mob. Comput. 20 (2) (2021) 445–459, http://dx.doi.org/10.1109/TMC.2019.2942306.

[2] D. Zheng, C. Peng, X. Liao, L. Tian, G. Luo, X. Cao, Towards latency optimization in hybrid service function chain composition and embedding, in: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 2020, pp. 1539–1548, http://dx.doi.org/10.1109/INFOCOM41043.2020.9155529.

[3] L. Cui, F.P. Tso, S. Guo, W. Jia, K. Wei, W. Zhao, Enabling heterogeneous network function chaining, IEEE Trans. Parallel Distrib. Syst. 30 (4) (2019) 842–854, http://dx.doi.org/10.1109/TPDS.2018.2871845.

[4] I. Lujic, V.D. Maio, K. Pollhammer, I. Bodrozic, J. Lasic, I. Brandic, Increasing traffic safety with real-time edge analytics and 5G, in: Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking, in: EdgeSys '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 19–24, http://dx.doi.org/10.1145/3434770.3459732.

[5] J. Halpern, C. Pignataro (Eds.), RFC 7665 - Service Function Chaining (SFC) Architecture, Internet Engineering Task Force (IETF) - Request for Comments: 7665, 2015, pp. 487–492.

[6] B. Farkiani, B. Bakhshi, S.A. MirHassani, T. Wauters, B. Volckaert, F. De Turck, Prioritized deployment of dynamic service function chains, IEEE/ACM Trans. Netw. 29 (3) (2021) 979–993, http://dx.doi.org/10.1109/TNET.2021.3055074.

[7] J. Castañeda, S.E. Pomares Hernández, S. Yangui, J.C. Pérez Sansalvador, L.M. Rodríguez Henríquez, K. Drira, VNF-based network service consistent reconfiguration in multi-domain federations: A distributed approach, J. Netw. Comput. Appl. 195 (2021) 103226, http://dx.doi.org/10.1016/j.jnca.2021.103226.

[8] X. Wang, X. Li, S. Pack, Z. Han, V.C.M. Leung, STCS: Spatial-temporal collaborative sampling in flow-aware software defined networks, IEEE J. Sel. Areas Commun. 38 (6) (2020) 999–1013, http://dx.doi.org/10.1109/JSAC.2020.2986688.

[9] L. Cui, F. Tso, W. Jia, Federated service chaining: Architecture and challenges, IEEE Commun. Mag. 58 (3) (2020) 47–53.

[10] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, S. Sinha, Real-time video analytics: The killer app for edge computing, Computer 50 (10) (2017) 58–67.
[11] R. Gouareb, V. Friderikos, Aghvami, Virtual network functions routing and placement for edge cloud latency minimization, IEEE J. Sel. Areas Commun. 36 (10) (2018) 2346–2357, http://dx.doi.org/10.1109/JSAC.2018.2869955.
[12] G. Sun, Y. Li, D. Liao, V. Chang, Service function chain orchestration across multiple domains: A full mesh aggregation approach, IEEE Trans. Netw. Serv. Manag. 15 (3) (2018) 1175–1191, http://dx.doi.org/10.1109/TNSM.2018.2861717.
[13] H. Chen, X. Wang, Y. Zhao, T. Song, Y. Wang, S. Xu, L. Li, MOSC: a method to assign the outsourcing of service function chain across multiple clouds, Comput. Netw. 133 (2018) 166–182, http://dx.doi.org/10.1016/j.comnet.2018.01.020.
[14] F. Esposito, M. Mushtaq, M. Berno, G. Davoli, D. Borsatti, W. Cerroni, M. Rossi, Necklace: An architecture for distributed and robust service function chains with guarantees, IEEE Trans. Netw. Serv. Manag. 18 (1) (2021) 152–166, http://dx.doi.org/10.1109/TNSM.2020.3036926.
[15] Z. Zhang, Z. Li, C. Wu, C. Huang, A scalable and distributed approach for NFV service chain cost minimization, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 2151–2156, http://dx.doi.org/10.1109/ICDCS.2017.210.
[16] A. Francescon, G. Baggio, R. Fedrizzi, R. Ferrusy, I.G. Ben Yahiaz, R. Riggio, X–MANO: Cross–domain management and orchestration of network services, in: 2017 IEEE Conference on Network Softwarization (NetSoft), 2017, pp. 1–5, http://dx.doi.org/10.1109/NETSOFT.2017.8004223.
[17] A. Abujoda, P. Papadimitriou, 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., 2016, http://dx.doi.org/10.1109/COMSNETS.2016.7439948.
[18] J.Y. Yen, Finding the K shortest loopless paths in a network , Manage. Sci. 17 (11) (1971) 712–716, http://dx.doi.org/10.1287/mnsc.17.11.712.
[19] ns-3 | a discrete-event network simulator for internet systems, https://www.nsnam.org/.
[20] Mininet: An instant virtual network on your laptop (or other PC) - Mininet, http://mininet.org/.
[21] Gurobi, Gurobi Optimizer Reference Manual, 2021.
[22] C. Chen, L. Nagel, L. Cui, F.P. Tso, Distributed federated service chaining for heterogeneous network environments, in: Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing, in: UCC '21, Association for Computing Machinery, New York, NY, USA, 2021.
[23] A. Khatiri, G. Mirjalily, Z.-Q. Luo, Balanced resource allocation for VNF service chain provisioning in inter-datacenter elastic optical networks, Comput. Netw. 203 (2022) 108717.
[24] N. Toumi, M. Bagaa, A. Ksentini, On using deep reinforcement learning for multi-domain SFC placement, in: 2021 IEEE Global Communications Conference (GLOBECOM), 2021, pp. 1–6, http://dx.doi.org/10.1109/GLOBECOM46510.2021.9685367.
[25] N. Slamnik-Krijestorac, G.M. Yilma, M. Liebsch, F.Z. Yousaf, J. Marquez-Barja, Collaborative orchestration of multi-domain edges from a connected, cooperative and automated mobility (CCAM) perspective, IEEE Trans. Mob. Comput. (2021) 1, http://dx.doi.org/10.1109/TMC.2021.3118058.
[26] K.D. Joshi, K. Kataoka, pSMART: a lightweight, privacy-aware service function chain orchestration in multi-domain NFV/SDN, Comput. Netw. 178 (2020) http://dx.doi.org/10.1016/j.comnet.2020.107295.
[27] A. Huff, G. Venâncio, V.F. Garcia, E.P. Duarte, Building multi-domain service function chains based on multiple NFV orchestrators, in: 2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2020, pp. 19–24, http://dx.doi.org/10.1109/NFV-SDN50289.2020.9289849.
[28] D. Dietrich, A. Abujoda, A. Rizk, P. Papadimitriou, Multi-provider service chain embedding with nestor, IEEE Trans. Netw. Serv. Manag. 14 (1) (2017) 91–105, http://dx.doi.org/10.1109/TNSM.2017.2654681.
[29] Q. Xiang, X. Tony Wang, J. Jensen Zhang, H. Newman, Y. Richard Yang, Y. Jace Liu, Unicorn: Unified resource orchestration for multi-domain, geo-distributed data analytics, Future Gener. Comput. Syst. 93 (2019) 188–197, http://dx.doi.org/10.1016/j.future.2018.09.048.
[30] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, H.A. Chan, Optimal virtual network function placement in multi-cloud service function chaining architecture, Comput. Commun. 102 (2017) 1–16, http://dx.doi.org/10.1016/j.comcom.2017.02.011.
[31] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, R. Boutaba, Distributed service function chaining, IEEE J. Sel. Areas Commun. 35 (11) (2017) 2479–2489, http://dx.doi.org/10.1109/JSAC.2017.2760178.
[32] G. Kibalya, J. Serrat-Fernandez, J.-L. Gorricho, D.G. Bujjingo, J. Serugunda, A multi-stage graph aided algorithm for distributed service function chain provisioning across multiple domains, IEEE Access 9 (2021) 114884–114904, http://dx.doi.org/10.1109/ACCESS.2021.3104841.
[33] R. Lin, S. Yu, S. Luo, X. Zhang, J. Wang, M. Zukerman, Column generation based service function chaining embedding in multi-domain networks, IEEE Trans. Cloud Comput. (2021) 1, http://dx.doi.org/10.1109/TCC.2021.3084999.
[34] Y. Liu, H. Zhang, D. Chang, H. Hu, GDM: A general distributed method for cross-domain service function chain embedding, IEEE Trans. Netw. Serv. Manag. 17 (3) (2020) 1446–1459, http://dx.doi.org/10.1109/TNSM.2020.2993364.
[35] H. Huang, C. Zeng, Y. Zhao, G. Min, Y. Zhu, W. Miao, J. Hu, Scalable orchestration of service function chains in NFV-enabled networks: A federated reinforcement learning approach, IEEE J. Sel. Areas Commun. 39 (8) (2021) 2558–2571, http://dx.doi.org/10.1109/JSAC.2021.3087227.
[36] K. Antevski, C.J. Bernardos, Federation of 5G services using distributed ledger technologies, Internet Technol. Lett. 3 (6) (2020) e193, http://dx.doi.org/10.1002/itl2.193.
[37] Z. Zhou, Q. Wu, X. Chen, Online orchestration of cross-edge service function chaining for cost-efficient edge computing, IEEE J. Sel. Areas Commun. 37 (8) (2019) 1866–1880, http://dx.doi.org/10.1109/JSAC.2019.2927070.
[38] C. Pham, N. Tran, S. Ren, W. Saad, C. Hong, Traffic-aware and energy-efficient vNF placement for service chaining: Joint sampling and matching approach, IEEE Trans. Serv. Comput. 13 (1) (2020) 172–185, http://dx.doi.org/10.1109/TSC.2017.2671867.
[39] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, Berlin, Germany, 2004.
[40] W.C. Lee, Topology aggregation for hierarchical routing in ATM networks, Comput. Commun. Rev. 25 (2) (1995) 82–92, http://dx.doi.org/10.1145/210613.210625.
[41] W. Nakimuli, G. Landi, R. Perez, M. Pergolesi, M. Molla, C. Ntogkas, G. Garcia-Aviles, J. Garcia-Reinoso, M. Femminella, P. Serrano, F. Lombardo, J. Rodriguez, G. Reali, S. Salsano, Automatic deployment, execution and analysis of 5G experiments using the 5G EVE platform, in: 2020 IEEE 3rd 5G World Forum (5GWF), 2020, pp. 372–377, http://dx.doi.org/10.1109/5GWF49715.2020.9221060.
[42] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, IEEE J. Sel. Areas Commun. 29 (9) (2011) 1765–1775, http://dx.doi.org/10.1109/JSAC.2011.111002.
[43] Amazon, EC2 on-demand instance pricing – Amazon web services, 2021, https://aws.amazon.com/ec2/pricing/on-demand/.
[44] CAIDA, The CAIDA UCSD passive-2019. URL https://data.caida.org/datasets/passive-2019.
[45] j. Pei, P. Hong, K. Xue, D. Li, Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system, IEEE Trans. Parallel Distrib. Syst. 30 (10) (2018) 2179–2192, http://dx.doi.org/10.1109/TPDS.2018.2880992.
[46] C. Pham, N.H. Tran, S. Ren, W. Saad, C.S. Hong, Traffic-aware and energy-efficient vNF placement for service chaining: Joint sampling and matching approach, IEEE Trans. Serv. Comput. (2017) 1, http://dx.doi.org/10.1109/tsc.2017.2671867.
[47] V. Eramo, E. Miucci, M. Ammar, F.G. Lavacca, An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures, IEEE/ACM Trans. Netw. 25 (4) (2017) 2008–2025, http://dx.doi.org/10.1109/TNET.2017.2668470.

Chen Chen received his B.S. degree from the Department of Electrical Engineering, Xidian University and his M.S. degree from the Department of Electrical Engineering, TU Dresden. Currently, he is pursuing his Ph.D. at Loughborough University. His research interests are broadly in areas of Software Defined Network, Cloud and Edge computing, network resource orchestration, in-network computing, distributed system and service chaining.

Lars Nagel is a lecturer at Loughborough University since October 2017. He received his computer science diploma from the Technical University Munich and his Ph.D. from the University of Durham. He worked for Trium Analysis Online and was a postdoctoral research fellow at Mainz University. His main research interests are randomized and distributed algorithms, especially for scheduling and load balancing. Application areas of his work include networks, cloud and high-performance computing and storage systems.

Lin Cui (Member, IEEE) received the Ph.D. degree from the City University of Hong Kong in 2013. He is currently a Professor with the Department of Computer Science, Jinan University, Guangzhou, China. He has broad interests in networking systems, with focuses on cloud data center networking, software defined networking, NFV, programmable networking, and distributed systems.

Fung Po Tso (Senior Member, IEEE) received the B.Eng., M.Phil., and Ph.D. degrees from the City University of Hong Kong in 2006, 2007, and 2011 respectively. He is currently a Senior Lecturer with the Department of Computer Science, Loughborough University. His research interests include: network policy management, network measurement and optimization, service chaining, data center networking, software defined networking, and edge computing.