TechRxiv
Powered by IEEE

Towards Optimal Parallelism-Aware Service Chaining and Embedding

This paper was downloaded from TechRxiv (https://www.techrxiv.org).

LICENSE

CC BY-SA 4.0

SUBMISSION DATE / POSTED DATE

30-11-2021 / 09-12-2021

CITATION

Zheng, Danyang; Shen, Gangxiang; Cao, Xiaojun; Mukherjee, Biswanath (2021): Towards Optimal Parallelism-Aware Service Chaining and Embedding. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.17099120.v1

DOI

10.36227/techrxiv.17099120.v1

Towards Optimal Parallelism-Aware Service Chaining and Embedding

Danyang Zheng, *Member, IEEE*, Gangxiang Shen*, *Senior Member, IEEE*, Xiaojun Cao, *Senior Member, IEEE*, and Biswanath Mukherjee, *Fellow, IEEE*

***Abstract*—Emerging 5G technologies can significantly reduce end-to-end service latency for applications requiring strict quality of service (QoS). With network function virtualization (NFV), to complete a client's request from those applications, the client's data can sequentially go through multiple service functions (SFs) for processing/analysis but introduce additional processing delay. To reduce the processing delay from the serially-running SFs, network function parallelism (NFP) that allows multiple SFs to run in parallel is introduced. In this work, we study how to apply NFP into the SF chaining and embedding process such that the latency, including processing and propagation delays, can be jointly minimized. We introduce a novel augmented graph to address the parallel relationship constraint among the required SFs. Considering parallel relationship constraints, we propose a novel problem called parallelism-aware service function chaining and embedding (PSFCE). For this problem, we propose a near-optimal maximum parallel block gain (MPBG) first optimization algorithm when computing resources at each physical node are enough to host the required SFs. When computing resources are limited, we propose a logarithm-approximate algorithm, called parallelism-aware SFs deployment (PSFD), to jointly optimize processing and propagation delays. We conduct extensive simulations on multiple network scenarios to evaluate the performances of our schemes. Accordingly, we find that (i) MPBG is near-optimal, (ii) the optimization of end-to-end service latency largely depends on the processing delay in small networks and is impacted more by the propagation delay in large networks, and (iii) PSFD outperforms the schemes directly extended from existing works regarding end-to-end latency.**

***Index Terms*—Network function virtualization, Network function parallelism, Parallelism-aware service function chaining and embedding, Approximation algorithm.**

I. Introduction

Network function virtualization (NFV) implements network functions (e.g., firewall, parental control) that run on traditional dedicated hardware to software-based modules, called virtual network functions (VNFs) or service functions (SFs) [1]–[3]. In the NFV paradigm, the client's NFV service request (NSR) includes the service source, destination, a set of SFs, and corresponding network resource demands (e.g., computing resource, bandwidth) [3]. To meet the client's request, the service provider can concatenate the required SFs into a service function chain (SFC) and embed it onto a shared physical network (PN) [4], [5]. The process of accommodating an NSR by composing and embedding an SFC onto a shared PN is referred to as service function chaining and embedding (SFCE). The physical forwarding path established by SFCE is called the service function path (SFP).

Recently, NFV techniques are applied in 5G networks to facilitate the low-lantecy service delivery [6]–[9], where 5G technologies are designed to significantly reduce latency (as much as 10x) [10]–[12]. Under such scenarios, the total processing delay from the serially-running SFs in an SFC may be comparable to the propagation delay and could be the bottleneck of optimizing the overall end-to-end latency for SFC delivery [13], [14]. To mitigate the impact caused by this bottleneck, network function parallelism (NFP) is introduced to run multiple SFs from the same request parallelly at one physical node (e.g., commercial/edge server) [13]. As a result, the processing delay of SFs working in parallel can be reduced from their processing delay sum (i.e., serially running SFs) to the highest processing delay among them (i.e., parallelly running SFs). According to [13], two SFs can be executed in parallel only if their operations do not conflict. For example, a flow monitor (FM) only monitors the client's data stream without any modifications, which can be operated with deep packet inspection (DPI) parallelly. On the contrary, both DPI and encryption might modify packets; thus, they cannot work in parallel. The constraint on whether two SFs can work in parallel or not is referred to as parallel relationship constraint.

When applying NFP to deliver SFC services with ultra-low latency requirements, we need to jointly optimize SFs' processing delay and the SFP propagation delay. Ideally, a physical node with enough computing resources can run many SFs in parallel to reduce SFs' processing delay, while SFP propagation delay can be reduced through embedding the required SFs along the shortest path connecting the service source and the destination. In practice, due to limited computing resources at each physical node, greedily minimizing SFs' processing delay may end up with increasing the SFP propagation delay. For example, a physical node with enough computing resources (e.g., a datacenter) may be geographically far away from both service source and destination, which may require a long physical routing path. In the literature, to reduce the latency of delivering a client's service, many existing works have focused on minimizing SFP length/propagation delay in SFCE, where SFs' processing delay is regarded as fixed or ignored [15]–[33]. The problem of how to apply NFP into SFCE to optimize the end-to-end latency is challenging.

In this work, we investigate how to efficiently apply NFP

D. Zheng, G. Shen, and B. Mukherjee are with Suzhou Key Laboratory of Advanced Optical Communication Network Technology in the School of Electronic and Information Engineering, Soochow University, Suzhou, 215006, China (e-mail: dyzheng@suda.edu.cn, shengx@suda.edu.cn, and bmukherjee@ucdavis.edu).

X. Cao is with the Department of Computer Science, Georgia State University, Atlanta, GA, 30302 USA (e-mail: cao@gsu.edu).

G. Shen is the corresponding author of this work.

into SFCE such that the end-to-end latency (including SFs' processing delay and the SFP propagation delay) of delivering parallelism-based services can be minimized. We introduce a novel augmented graph, called parallel graph (PG), to address the parallel relationship constraints. Considering parallel relationship constraints, we mathematically model the parallelism-aware service chaining and embedding (PSFCE) problem with the goal of jointly minimizing the SFs' processing delay and the SFP propagation delay. Next, we prove that the PSFCE problem is NP-hard under various network scenarios. For this NP-hard problem, we propose two efficient heuristic algorithms called maximum parallel block gain (MPBG) first optimization and parallelism-aware SFs deployment (PSFD) to optimize the end-to-end service latency. Meanwhile, we show that MPBG can achieve the optimal performance in some scenarios and PSFD is generally logarithm-approximate. We conduct extensive simulations to evaluate the performances of our proposed algorithms. Specifically, we show that (i) MPBG is near-optimal, (ii) the optimization of the end-to-end service latency largely depends on SFs' processing delay in small networks and is impacted more by the SFP propagation delay in large networks, (iii) to achieve latency-efficient service delivery in edge-cloud systems, short parallelism-aware SFCs (less than 10 SFs) should be deployed at the edge, while long parallelism-aware SFCs should be deployed in the cloud, and (iv) PSFD outperforms the schemes that are directly extended from existing works regarding the end-to-end latency.

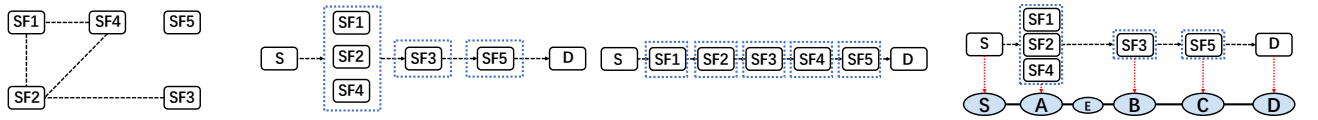
The rest of this paper is organized as follows. Section II summarizes related work, while Section III introduces network function parallelism and parallelism-based SFC (P-SFC). We formulate the problem of parallelism-aware service function chaining and embedding (PSFCE) in Section IV. In Sections V, VI, and VII, we analyze the NP-hardness of PSFCE and present novel analysis and algorithms to optimize it in various network scenarios. Section VIII analyzes experimental results. We conclude our work in Section IX.

II. Related Work

With network function virtualization (NFV), service providers can save operating expense (OpEx) and capital expenditure (CapEx) by flexibly implementing network functions as virtual modules [1]–[3]. Recently, NFV drew great attention from academia and industry [34]–[36]. When applying NFV techniques to emerging systems such as 5G networks, multi-access edge computing (MEC) systems, and large-scale deterministic networks, the QoS and service level agreements (SLAs) may require NFV services to be delivered within ultra-low latency [6]–[9], [33], [37], [38].

To satisfy ultra-low latency requirements, much work has been done to optimize SFP length/propagation delay [15]–[33]. When delivering service as a traditional SFC (linear logical structure), authors in [15] proposed a heuristic algorithm by applying the betweenness centrality technique to minimize the number of hops and hence the propagation delay. When the client has specific QoS requirements, authors in [16] formulated the QoS-aware and reliable traffic steering (QRTS) problem and proposed an approximate algorithm by applying the primal and dual technique. When the SFC is given a priori, authors in [17] developed SFC-constrained shortest-path schemes with the transformation of network graphs. To optimize propagation delay in the scenario of online/continuous learning, authors in [18], [19] investigated delivering a hybrid service function chain (HSFC). When the client identifies HSFC, authors in [18] proposed an optimal hybrid SFC embedding (Opt-HSFCE) algorithm, which optimizes propagation delay from the constructed SFP. When the HSFC is not given a priori, to optimize propagation delay, authors in [19] proposed a 2-approximation algorithm by applying graph-theory based techniques, called Eulerian circuit-based hybrid SFP optimization (EC-HSFP). Under latency limitation, the work in [21] proposed a heuristic algorithm to jointly optimize the resource utilization of both physical nodes and links. In [28], the authors proposed an architecture to reduce the latency of NFV systems with 5G techniques. The authors in [30] proposed a mixed integer linear program (MILP) and three heuristics to optimize the resource utilization for accommodating SFCs. In [33], the authors proposed a heuristic approach to deploy a given SFC under latency and mobility constraints.

To reduce SFs' processing delay, the authors in [13] proposed the technique of network function parallelism (NFP), which enables simultaneously executing multiple parallelable SFs. Based on that, works in [39]–[46] investigated how to apply NFP while considering dependencies among the required SFs. Here, dependencies represent executing orders of the required SFs. In [44], the authors proposed a heuristic algorithm, called parallelism-aware residual capacity first placement (PARC), to maximize the request acceptance while satisfying a specific latency constraint. In [45], the authors proposed a heuristic approach, called partial parallel chaining (PPC), to accommodate a P-NSR, where SFs are mutually parallelizable. That is, if SF1 is parallelizable with SF2 and SF3, then SF2 can co-execute with SF3 as well. In [46], the authors proposed a scheme, called delay-balanced parallelism, to balance the overhead and the processing delay of running the required SFs. However, the above works hardly take the parallel relationship constraints into account. Notably, the parallel relationship constraint identifies whether two SFs can work in parallel or not, and it is different from the dependency constraints. For example, firewall (FW) and DPI can be executed in either sequential order (i.e., FW to DPI or DPI to FW), but they cannot work in parallel as they both modify packet contents. As the flow monitor (FM) does not modify the traffic, FM can be parallelly executed with DPI or FW. That is, even though SF1 is parallelizable with SF2 and SF3, SF2 and SF3 may not be able to co-execute. The parallel relationship constraint is further illustrated in the subsequent Section. Meanwhile, the SFC is given a priori in the above works, so the SFC composition process is not taken into consideration. Compared to the above works, this work takes into account the parallel relationship constraints and study how to jointly compose and embed a parallelism-aware SFC (P-SFC) in diverse network scenarios such that the end-to-end service latency (including the processing delay and the propagation delay) is minimized.



(a) An example of PG. (b) Composed P-SFC 1. (c) Composed P-SFC 2. (d) An example of P-SFC embedding.

Fig. 1: Parallel graph (PG), different P-SFCs and P-SFC embedding process. S and D represent the source and destination.

III. Parallelisms in Network Function Virtualization

A. Network Function Parallelism (NFP)

Different network functions will perform diverse operations on client's data packets. These operations include writing, reading, dropping, and so on [13]. When two operations need to simultaneously change the client's packet content (e.g., dropping and dropping, writing and dropping), the relationship between these two operations is referred to as conflicting. When operations of multiple SFs are non-conflicting, network function parallelism (NFP) is introduced to allow these SFs running in parallel and simultaneously process the packets for the same client [13]. Parallelizable SFs are those with non-conflicting operations.

B. Parallel Graph (PG)

We use parallel graph (PG) to describe parallel relationship constraints among the required SFs. PG is defined as an undirected graph $PG = (V, E)$, where V represents the set of SFs, and E is the set of parallel relationship constraints among the SFs. Two SFs are parallelizable if and only if they are directly connected in the PG; otherwise, they are non-parallelizable. For example, in the PG of Fig. 1a, SF1 and SF2 can be executed in parallel, while SF5 is non-parallelizable with all other four SFs. The triangle in Fig. 1a (including SF1, SF2, and SF4) is a complete graph, and these three SFs can work in parallel. For example, SF1, SF2, SF3, SF4, and SF5 can be the functionalities of load balancing (LB), flow monitor (FM), gaming core (GC), network address translation (NAT), and DPI, respectively [13]. Since DPI needs to perform the operations of reading, writing, and dropping on diverse fields in packets, it cannot be executed in parallel with any other SFs. As FM only performs the operation of reading, it can parallelly execute with LB, GC, and NAT. Even though GC and LB both perform the operations of reading and writing, the packet fields that are executed by these two operations are different. Thus, GC and LB can co-execute with FW as the triangle (i.e., SF1, SF2, and SF4) shows. NAT needs to perform writing operations on multiple packet fields that are overlapped with GC and LB, so it can only be parallelly executed with FM.

C. Parallelism-based Service Function Chain (P-SFC)

A service function chain (SFC) represents the sequential execution order of the required SFs. When applying NFP to SFCE, we define the parallelism-based SFC (P-SFC) to specify both consecutive execution order and parallelisms among the required SFs. A P-SFC is composed of a set of consecutively-ordered *Parallel Blocks*, each of which consists of either a set of parallelizable SFs or one SF. Figs. 1b and 1c show two possible P-SFCs that are composed from the PG in Fig. 1a, where a blue dashed-line rectangle represents a parallel block. Specifically, the P-SFC in Fig. 1b contains three parallel blocks, where the first parallel block includes SF1, SF2, and SF4; while the second and third parallel blocks include SF3 and SF5, respectively. The P-SFC in Fig. 1c includes five parallel blocks, each of which contains one SF, i.e., five SFs will be sequentially executed in Fig. 1c. Note that the SFs assigned to the same parallel block must be parallelizable SFs, i.e., the PG sub-graph formed by the SFs in the same parallel block must be a complete graph. For example, in Fig. 1a, the sub-graph of SF1, SF2, and SF4 is a complete graph.

Lemma 1. *The sub-graph of SFs that can work in the same parallel block must be a complete graph in PG.*

D. Processing Delay and Propagation Delay

When parallelly executing multiple SFs, it is necessary to merge the results from these SFs to guarantee the correctness and uniqueness of the output [13]. It is worth noting that the SFs in the same parallel block will be embedded onto the same physical node in this work. Fig. 1d shows an example of embedding the P-SFC in Fig. 1b onto a PN, where the red-dotted arrow shows the embedding relationship between the parallel block and the physical node. To facilitate the description, the necessary notations are summarized in Table I. We use ξ_i to represent the i^{th} parallel block in the P-SFC. In Fig. 1d, ξ_1={SF1, SF2, SF4} is embedded onto the physical node A, ξ_2={SF3} is embedded onto physical node B, and ξ_3={SF5} is embedded onto physical node C. We use $\mathbb{L}_{\xi_i}$ to represent the processing delay of ξ_i, which is calculated from Eq. (1), where $\mathbb{L}_v$ is the processing delay of SF v.

$$\mathbb{L}_{\xi_i} = max_{v \in \xi_i} \quad \mathbb{L}_v \tag{1}$$

The processing delay of a P-SFC ($\mathbb{L}_{\mathbb{PC}}$) is the sum of its parallel blocks' processing delay. We use $\mathbb{B}$ to represent the set of parallel blocks including source and destination in a P-SFC, and overall processing delay is given by Eq. (2).

$$\mathbb{L}_{\mathbb{PC}} = \Sigma_{\xi_i \in \mathbb{B}} \quad \mathbb{L}_{\xi_i} \tag{2}$$

When ξ_i and ξ_j are embedded onto physical nodes n_i and n_j, there will be a physical forwarding path P_{n_i,n_j} composed of a series of physical links from node n_i to node n_j. For example, in Fig. 1d, the forwarding path $P_{A,B}$ includes $A \rightarrow E \rightarrow B$ carrying the traffic from parallel block ξ_1 to ξ_2. Accordingly, we define $\mathbb{L}_{P_{n_i,n_j}^{\xi_i,\xi_j}}$ to represent the propagation delay from physical node n_i that hosts ξ_i to physical node n_j that hosts ξ_j. We use $\mathbb{L}_{\mathbb{PD}}$ to represent propagation delay of the forwarding

path (i.e., the SFP in the physical network) for the P-SFC, which can be calculated by Eq. (3).

$$\mathbb{L}_{\mathbb{PD}} = \Sigma_{\xi_i \in \mathbb{B}} \quad \mathbb{L}_{P_{n_i,n_j}^{\xi_i,\xi_{i+1}}} \tag{3}$$

The overall processing delay and propagation delay of Fig. 1d are $\mathbb{L}_{\xi_1} + \mathbb{L}_{\xi_2} + \mathbb{L}_{\xi_3}$ and $\mathbb{L}_{\mathbb{P}_{S,A}^{s,\xi_1}} + \mathbb{L}_{P_{A,B}^{\xi_1,\xi_2}} + \mathbb{L}_{P_{B,C}^{\xi_2,\xi_3}} + \mathbb{L}_{P_{C,D}^{\xi_3,d}}$, respectively. The overall end-to-end service latency in Fig. 1d is the sum of $\mathbb{L}_{\mathbb{PC}}$ and $\mathbb{L}_{\mathbb{PD}}$.

IV. Parallelism-aware Service Function Chaining and Embedding (PSFCE)

A. Physical Network Model

The physical network (PN) is an undirected graph $PN = (N, L)$, where N represents the set of physical nodes, and each node can represent a datacenter, edge server, or point of presence (PoP), and L is the set of physical links that connect the physical nodes in the PN. Each physical node $n \in N$ has a certain amount of computing resources C_n and can host any type of SF. For a link $l_{m,n} \in L$ $(m, n \in N)$, it has a specific amount of available bandwidth $bw_{l_{m,n}}$ and has a propagation delay $\mathbb{L}_{l_{m,n}}$. A physical forwarding path $P_{m,n}$ is composed of a series of physical links from physical node m to n. We use $\mathbb{L}_{P_{m,n}}$ and $BW_{P_{m,n}}$ to represent the propagation delay and the bandwidth of $P_{m,n}$, respectively.

B. Parallelizable NFV Service Requests

The parallelizable NFV service request (P-NSR) is defined as a 4-tuple P-NSR $=< s, d, PG, BW >$, where s and d are the service source and destination; $PG = (V, E)$ represents the parallel graph; and BW is the bandwidth demand. For each SF $v \in V$, it requires a specific network function, a certain amount of computing demand C_v, and has a maximum processing delay $\mathbb{L}_v$.

C. Problem Formulation

Given a P-NSR, PSFCE is defined as: how to compose a P-SFC for a P-NSR and embed it onto a given PN such that (i) the end-to-end service latency (i.e., the sum of processing delay and propagation delay) of the constructed SFP is minimized; and (ii) the following constraints are satisfied. Table I lists the notations used in the problem formulation.

The objective function of the proposed PSFCE problem is shown as Eq. (4), which minimizes the overall service latency.

$$min \sum_{\xi_i \in \mathbb{B}} \mathbb{L}_{\xi_i} + \sum_{\xi_i \in \mathbb{B}} \sum_{\xi_j \in \mathbb{B}} \sum_{n_i \in N} \sum_{n_j \in N} \mathbb{L}_{P_{n_i,n_j}^{\xi_i,\xi_j}} \tag{4}$$

SF node embedding constraint: Eq. (5) shows whether an SF node v is assigned to the i^{th} parallel block or not, while Eq. (6) checks whether the i^{th} parallel block is mapped onto physical node m or not. Eq. (7) ensures that u and v can work in parallel if and only if there is an edge between u and v in the given PG. If ξ_i is embedded onto the physical node m, Eq. (8) requires that m must have enough computing resources to host it. Eq. (9) and Eq. (10) guarantee that each SF node v will be assigned to one parallel block, and each parallel block must be embedded onto one physical node. Eq. (11) calculates the processing delay of a parallel block.

TABLE I: Notation Table for Problem Formulation.

Notation	Definition
$\mathbb{B}$	Set of parallel blocks in the constructed P-SFC.
ξ_i, ξ_j	Parallel block $\xi_i, \xi_j \in \mathbb{B}$.
$\mathbb{L}_{\xi_i}$	Processing delay of ξ_i.
$\mathbb{L}_{\mathbb{PC}}$	Overall processing delay.
$\mathbb{L}_{\mathbb{PD}}$	Overall propagation delay.
N	Set of physical nodes.
V	Set of required SFs.
m, n	Physical node $m, n \in N$.
u, v	Service functions (SFs) $u, v \in V$.
C_n / C_v	Computing resources(demand) of $n(v)$.
$l_{u,v}$	Edge between u and v in the PG.
$l_{m,n}$	Physical link connecting m, n in PN.
$P_{m,n}$	Physical path from m to n.
BW	Bandwidth demand.
$\gamma_v^{\xi_i}$	=1 v is assigned to the i^{th} Parallel block; 0 otherwise.
$M_{n_i}^{\xi_i}$	=1 ξ_i is mapped onto n; 0 otherwise.
$P_{n_i,n_j}^{\xi_i,\xi_j}$	=1 P_{n_i,n_j} supports the traffic from ξ_i to ξ_j; 0 otherwise.
$BW_{P_{m,n}}$	Bandwidth of $P_{m,n}$.
$\mathbb{L}_{\mathbb{P}_{n_i,n_j}^{\xi_i,\xi_j}}$	Propagation delay from ξ_i to ξ_j along P_{n_i,n_j}.

$$\gamma_v^{\xi_i} = \begin{cases} 1, & v \text{ is assigned to the } i^{th} \text{ parallel block} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

$$M_{n_i}^{\xi_i} = \begin{cases} 1, & \xi_i \text{ is embedded onto physical node } n_i \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$$\gamma_v^{\xi_i} + \gamma_u^{\xi_i} \leq 1, \forall l_{u,v} \notin E, \forall \xi_i \in \mathbb{B} \tag{7}$$

$$M_{n_i}^{\xi_i} * \sum_{v \in V} \gamma_v^{\xi_i} * C_v \leq C_{n_i}, \forall \xi_i \in \mathbb{B}, \forall n_i \in N \tag{8}$$

$$\sum_{\xi_i \in \mathbb{B}} \gamma_v^{\xi_i} = 1, \forall v \in V \tag{9}$$

$$\sum_{n_i \in N} M_{n_i}^{\xi_i} = 1, \forall \xi_i \in \mathbb{B} \tag{10}$$

$$\mathbb{L}_{\xi_i} = max_{v \in V} \ \gamma_v^{\xi_i} * \mathbb{L}_v \tag{11}$$

SFP routing constraint: We use $P_{n_i,n_j}^{\xi_i,\xi_j}$ to denote whether P_{n_i,n_j} is the forwarding path from ξ_i to ξ_j as Eq. (12). Eq. (13) and Eq. (14) ensure that, if P_{n_i,n_j} is employed to transmit the traffic from ξ_i to ξ_j, then ξ_i and ξ_j must be embedded onto n_i and n_j, and P_{n_i,n_j} must have enough bandwidth. Eq. (15) and Eq. (16) require that (i) there must be a physical path starting from each parallel block (including source), and (ii) there must be a physical path ending at each parallel block (including destination). Eq. (17) computes the propagation delay of $P_{n_i,n_j}^{\xi_i,\xi_j}$.

$$\mathbb{P}_{n_i,n_j}^{\xi_i,\xi_j} = \begin{cases} 1, & \text{path from } n_i \text{ to } n_j \text{ employed} \\ & \text{to transmit traffic from } \xi_i \text{ to } \xi_j \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

$$\mathbb{P}_{n_i,n_j}^{\xi_i,\xi_j} \leq \frac{M_{n_i}^{\xi_i} + M_{n_j}^{\xi_j}}{2}, \forall n_i, n_j \in N, \forall \xi_i, \xi_j \in \mathbb{B} \tag{13}$$

$$\mathbb{P}_{n_i,n_j}^{\xi_i,\xi_j} * BW \leq BW_{P_{n_i,n_j}}, \forall n_i, n_j \in N_P, \forall \xi_i, \xi_j \in \mathbb{B} \tag{14}$$

$$\sum_{\xi_j \in \mathbb{B}, \xi_j \neq \xi_i} \sum_{n_i \in N} \sum_{n_j \in N} \mathbb{P}_{n_i,n_j}^{\xi_i,\xi_j} = 1, \forall \xi_i \in \{\mathbb{B} - \{d\}\} \quad (15)$$

$$\sum_{\xi_i \in \mathbb{B}, \xi_i \neq \xi_j} \sum_{n_i \in N} \sum_{n_j \in N} \mathbb{P}_{n_i,n_j}^{\xi_i,\xi_j} = 1, \forall \xi_j \in \{\mathbb{B} - \{s\}\} \quad (16)$$

$$\mathbb{L}_{P_{n_i,n_j}^{\xi_i,\xi_j}} = P_{n_i,n_j}^{\xi_i,\xi_j} * \mathbb{L}_{P_{n_i,n_j}}, \quad (17)$$

$$\forall \xi_i, \xi_j \in \mathbb{B}, \xi_i \neq \xi_j, \forall n_i, n_j \in N$$

V. NP-HARDNESS OF PSFCE

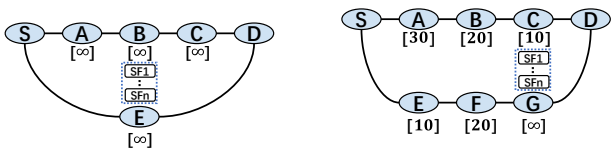
In this section, we analyze the NP-hardness of the proposed PSFCE problem in various network scenarios: (i) every physical node has enough computing resources to host the required SFs, and the PG is a complete graph, (ii) every physical node has enough computing resources to host the required SFs, and the PG is a non-complete graph, and (iii) every physical node has limited computing resources, and the required SFs cannot be hosted by one physical node.

A. Enough Computing Resources and Complete PG

With enough computing resources, any physical node can host all required SFs in a client's request. In practice, the length of an SFC is generally no more than 8 [47]. Therefore, with enough computing resources, it is possible that any physical node can host the SFs for a moderate SFC. In this case, SFP propagation delay can be optimized by embedding all required SFs onto the physical nodes along one bandwidth-aware shortest path from the service source to the destination. Since PG is a complete graph, all required SFs can work in parallel, and the optimally-constructed P-SFC only has one parallel block, whose processing delay is the highest processing delay among all SFs. As a result, the PSFCE problem in this scenario can be optimized by embedding and parallelly executing all SFs at one physical node along the bandwidth-aware shortest path connecting the service source to the destination. Fig. 2a shows an example of accommodating a P-NSR to a PN, where all nodes have enough computing resources while PG is a complete graph. In this case, all SFs can be executed in one parallel block, and this block is embedded onto one physical node along the bandwidth-aware shortest path, i.e., $S \rightarrow E \rightarrow D$. Similarly, when one physical node with enough computing resources is located along the bandwidth-aware shortest path as the physical node G in Fig. 2b, this shortest path is an optimal solution for PSFCE. Since the running time complexity of the bandwidth-aware shortest path algorithm is polynomial [48], the following theorem holds.

Theorem 1. *When each physical node has enough computing resources and PG is a complete graph, the PSFCE problem can be optimized within polynomial time.*

Lemma 2. *When one physical node along the bandwidth-aware shortest path has enough computing resources, the PSFCE problem can be optimized within polynomial time.*



(a) Optimal PSFCE solution in a PN with enough computing resources.

(b) Optimal PSFCE solution when the bandwidth-aware shortest path includes a node with enough computing resources.

Fig. 2: Optimal PSFCE solutions with enough computing resource. ∞ is used to present enough computing resources.

B. Enough Computing Resources and Non-Complete PG

When each physical node has enough computing resources, the SFP propagation delay can be optimized. However, since PG is a non-complete graph, optimizing SFs' processing delay and PSFCE will be NP-hard, as shown in Theorem 2.

Theorem 2. *When PG is a non-complete graph, the PSFCE problem is NP-hard.*

Proof. If each SF has the same processing delay, optimizing the P-SFC processing delay is equivalent to constructing the minimum number of parallel blocks in the P-SFC. Then, we can create the complement graph (PG_C) of PG, each edge in which represents the non-parallel relationship. When we label the SFs that are assigned to the same parallel block with the same color, minimizing the number of parallel blocks in the constructed P-SFC is equivalent to coloring the PG_C with the minimum number of colors, which is a well-known NP-hard problem [49]. Therefore, in this scenario, optimizing SFs' processing delay is NP-hard, and PSFCE is NP-hard. □

C. Limited Computing Resource

When computing resources at each physical node are limited, the physical nodes along the bandwidth-aware shortest path may not have enough computing resources to host all required SFs. In this case, propagation delay is determined by the complex routing process, and SFs' processing delay can not be optimized no matter whether PG is a complete graph or not. As a result, we need to jointly optimize propagation and processing delays, which is NP-hard, as shown in Theorem 3.

Theorem 3. *When each physical node has limited computing resources, PSFCE is NP-hard.*

Proof. We assume that every SF requests the same amount of computing resources, and the physical network only contains $|N| = |V|$ physical nodes, each of which has only enough computing resources to host one SF. As a result, SFs' processing delay is fixed as the sum of processing delay from all SFs, and optimizing SFP propagation delay is equivalent to finding the shortest path spanning all physical nodes in PN, which is the traveling salesman path problem (TSPP) [50]. As TSPP is a well-known NP-hard problem, PSFCE is also NP-hard in this scenario. □

From the above discussion, we can see that there are three important scenarios to optimize PSFCE. **(i)** When each

physical node has enough computing resources, and PG is a complete graph, PSFCE can be optimized by embedding and parallelly executing all SFs at one physical node along the bandwidth-aware shortest path. **(ii)** When each physical node has enough computing resources and PG is a non-complete graph, PSFCE becomes NP-hard. In this case, SFP propagation delay is solvable by applying the bandwidth-aware shortest path algorithm, and we propose an efficient maximum parallel block gain (MPBG) first optimization algorithm in Section VI to minimize SFs' processing delay. **(iii)** When computing resources at each physical node are limited, PSFCE remains NP-hard regardless of whether PG is a complete graph or not. To jointly optimize SFs' processing delay and SFP propagation delay, we propose and analyze a novel parallelism-aware SFs deployment (PSFD) algorithm in Section VII.

VI. PSFCE with Enough Computing Resources

When each physical node has enough computing resources to host the required SFs, the SFP propagation delay can be minimized by embedding all required SFs at physical nodes along the bandwidth-aware shortest path from the service source to the destination. However, minimizing the SFs' processing delay is challenging when PG is a non-complete graph, as proved by Theorem 2. To minimize SFs' processing delay, one needs to properly assign the required SFs to different parallel blocks and satisfy all parallel relationship constraints. Here, we propose parallel block gain (PBG) and maximum parallelism block gain (MPBG) first optimization algorithm to efficiently construct P-SFC while minimizing SFs' processing delay.

A. Parallel Block Gain (PBG)

A parallel block ξ_i is composed of a set of SFs that can work in parallel. The processing delay of a parallel block is denoted by $\mathbb{L}_{\xi_i}$, which depends on the in-block SF that has the highest processing delay. According to Eqs. (1) and (2), only the SF with the highest processing delay will determine the overall P-SFC processing delay for that parallel block, while the processing delay of other SFs are not counted, thus being saved. For a set of SFs, the maximum processing delay is the processing delay sum of these SFs (i.e., serially running these SFs). That is, minimizing the processing delay for a set of SFs is equivalent to maximizing the processing delay being saved by the SF parallelism process. Accordingly, we define the amount of processing delay being saved in a parallel block ξ_i as parallel block gain (PBG_{ξ_i}) in Eq. (18).

$$PBG_{\xi_i} = \sum_{v \in \xi_i} \mathbb{L}_v - \mathbb{L}_{\xi_i} \tag{18}$$

The parallel block gain for a P-SFC ($PBG_{\text{P-SFC}}$) is the sum of PBG_{ξ_i} from all parallel blocks as Eq. (19).

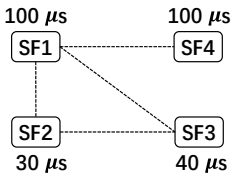
$$PBG_{\text{P-SFC}} = \Sigma_{\xi_i \in \mathbb{B}}\ PBG_{\xi_i} \tag{19}$$

The P-SFC processing delay $\mathbb{L}_{\mathbb{PC}}$ can be calculated by Eq. (20), where $\mathbb{L}_{\text{SFC}}$ is the processing delay of executing all required SFs without any parallelisms (i.e., the traditional SFC).

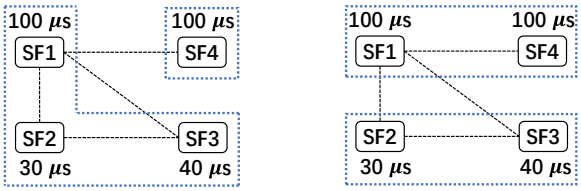
$$\mathbb{L}_{\mathbb{PC}} = \mathbb{L}_{\text{SFC}} - PBG_{\text{P-SFC}} \tag{20}$$

B. Maximum Parallel Block Gain (MPBG) first optimization

Maximizing PBG_{ξ_i} for a parallel block ξ_i will maximize parallelism(s) in this block, leading more SFs to operate in parallel, which can in turn increase the parallel block gain for P-SFC as shown in Eq. (19). As $\mathbb{L}_{\text{SFC}}$ is fixed for a given P-NSR, Eq. (20) shows that minimizing the overall P-SFC processing delay is equivalent to maximizing PBG of P-SFC (i.e., $PBG_{\text{P-SFC}}$). To optimize P-SFC processing delay, we need to properly construct parallel blocks $\xi \in \mathbb{B}$ such that the PBG of the constructed P-SFC in Eq. (19) is maximized.



(a) A given parallel graph (PG).



(b) ξ_1={SF1,SF2,SF3} ξ_2={SF4}. (c) ξ_1={SF1,SF4} ξ_2={SF2,SF3}.

Fig. 3: Options to construct parallel blocks.

The PG in Fig. 3a identifies four SFs and their parallel relationship constraints, while the number beside each SF node represents its processing delay. There are multiple options to construct parallel blocks. Fig. 3b first selects the largest complete sub-graph in PG to construct $\xi_1 = \{SF1, SF2, SF3\}$ and then the remaining SF4 will be alone in ξ_2. As a result, the PBG for ξ_1 and ξ_2 is 70 μs and 0, respectively. In total, $PBG_{\text{P-SFC}}$ in Fig. 3b is 70 μs. Fig. 3c identifies another way to construct parallel blocks via maximizing the PBG value of each parallel block. By doing so, parallel block ξ_1 will include SF1 and SF4, which has PBG of 100 μs, and parallel block ξ_2 includes SF2 and SF3, which has PBG of 30 μs. Overall, $PBG_{\text{P-SFC}}$ of the scheme in Fig. 3c is 130 μs.

Algorithm 1 MPBG algorithm

```
Input: P-NSR;
Output: P-SFC;
Initialize index i = 1;
while V is not empty do
    Find set of SFs (V^sub) that maximizes Eq. (18);
    Construct ξ_i by assigning the SFs in V^sub to ξ_i;
    Set V = V − V^sub, and i = i + 1;
end while
Set ξ_0 = s and ξ_last = d;
Formulate P-SFC by concatenating all parallel blocks in
their index orders (i.e., B = {ξ_0,ξ_1,...,ξ_last});
return P-SFC;
```

In fact, optimizing $PBG_{\text{P-SFC}}$ in Eq. (19) is NP-hard, which can be proved via reductions from the maximum clique prob-

lem (MCP) or weighted maximum clique problem (WMCP) [51], [52]. Thanks to the importance of MCP and WMCP problems in the field of data science, many existing works have proposed fast and exact optimization techniques [53], [54]. Based on the above examples in Fig. 3, to maximize $PBG_{\text{P-SFC}}$ and in turn reduce the processing delay of P-SFC, we propose an efficient heuristic algorithm, namely, maximum parallel block gain (MPBG) first optimization algorithm, which greedily constructs the parallel blocks with large PBG values, as shown in Alg. 1. In Lines 4-8, when there exist SFs that are not yet assigned to any parallel block, MPBG will repeat: (i) find a set of SFs (V^{sub}) that maximize Eq. (18) by recursively calculating the maximum PBG value of each parallel block that the SFs can construct based on PG; (ii) construct i^{th} parallel block (ξ_i) by using the SFs in V^{sub}; and (iii) remove set of SFs in V^{sub} from V. At last, MPBG sets $\xi_0 = s$, $\xi_{last} = d$, and formulates P-SFC by concatenating all constructed parallel blocks.

C. Bound Analysis of MPBG Algorithm

An essential property of ξ_i constructed by applying MPBG in the i^{th} iteration is described in Lemma 3.

Lemma 3. *Each parallel block (ξ_i) constructed by MPBG includes the SFs whose sub-graph in PG is a maximal clique.*

Proof. We prove this lemma by contradictions. We assume that there exists a parallel block (ξ_i') that is the super set of ξ_i. We define the SF that is included in ξ_i' but not in ξ_i as v'. If $\mathbb{L}_{v'} > \mathbb{L}_{\xi_i}$, then Eq. (21) holds, which contradicts the fact that PBG_{ξ_i} is the maximum value found.

$$\sum_{v\in\xi_i'} \mathbb{L}_v - \mathbb{L}_{\xi_i'} = \sum_{u\in\xi_i} \mathbb{L}_u > \sum_{v\in\xi_i} \mathbb{L}_v - \mathbb{L}_{\xi_i} \tag{21}$$

If $\mathbb{L}_{v'} \leq \mathbb{L}_{\xi_i}$, then $\mathbb{L}_{\xi_i'} = \mathbb{L}_{\xi_i}$, and Eq. (22) holds, which contradicts that $PBG_{\xi_i^{\text{PBG}}}$ is the maximum value found.

$$\sum_{v\in\xi_i'} \mathbb{L}_v = \sum_{v\in\{\xi_i\cup\{v'\}\}} \mathbb{L}_v > \sum_{u\in\xi_i} \mathbb{L}_u \tag{22}$$

As a result, there does not exist a parallel block (ξ_i') that is the superset of the parallel block constructed by MPBG. And the parallel block ξ_i constructed by MPBG includes the SFs that construct a maximal clique in PG. □

With Lemma 3, we then prove that MPBG can achieve optimal performance in the following scenarios.

Theorem 4. *With enough computing resources, when there exist only two maximal cliques in PG, MPBG achieves optimal performance.*

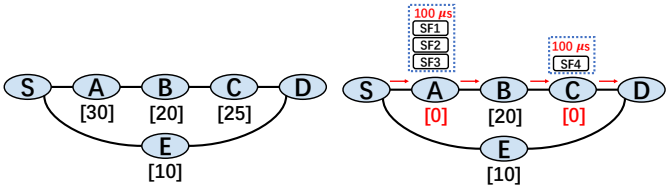
Proof. As proved by Lemma 3, the first parallel block ξ_1 constructed by MPBG is not the subset of any other possible parallel block. In other words, the set of SFs in ξ_1 is, in fact, the set of vertices that construct one of the maximal cliques in PG. Similarly, the parallel block constructed in the next iteration will be composed of the SFs in the remaining maximal clique of the PG. As only two maximal cliques exist in the PG, MPBG optimizes the PBG value in Eq. (19). Hence, MPBG achieves optimal performance. □

We list some examples where there exist only two different maximal cliques in a PG. For instance, when the complement graph of PG (PG_C) is a path, complete binary (ternary) tree, star, or complete bipartite graph, there exist only two maximal cliques in the PG. Additionally, when PG is one of the graphs listed in Lemma 4, the optimality of MPBG also holds.

Lemma 4. *With enough computing resources, when the complement graph of PG is a trivial graph, circle, wheel, or complete multi-partite graph, the MPBG algorithm achieves optimal performance.*

VII. PSFCE with Limited Computing Resources

When each physical node has limited computing resources, optimizing the PSFCE problem is different from and more challenging than in Section VI. This is because (i) one cannot flexibly create a parallel block as there might not exist a physical node with enough computing resources to host it, and (ii) the physical nodes along the bandwidth-aware shortest path might not have enough computing resources to accommodate the P-NSR. To demonstrate these, we use the PG in Fig. 3a as the request example. We assume that SF1, SF2, and SF3 require 10 Gb computing resources while SF4 needs 25 Gb computing resources. Fig. 4a is the physical network, where the number in the square bracket represents the available computing resources at each physical node. Due to limited computing resources at each physical node, none of the physical nodes can host the parallel block ξ={SF1, SF4} that requires 10 + 25 = 35 Gb computing resources, even though this ξ maximizes PBG in Eq. (19). Hence, one may accommodate the PG in Fig. 3b by composing ξ_1={SF1, SF2, SF3}, ξ_2={SF4} and embedding them onto physical nodes A and C in Fig. 4b, where physical nodes C and D now have no computing resources left. Similarly, due to the limited computing resources at physical node E, we cannot route the forwarding path along the shortest path $S \rightarrow E \rightarrow D$.



(a) A physical network with limited computing resources.

(b) Embedding result in limited computing resources network.

Fig. 4: Parallel block construction and embedding processes are constrained by limited computing resources.

The above example shows that the limited computing resources impact how to construct parallel blocks and how to deploy (i.e., embed and route) parallel blocks in the physical network. Here, to optimize PSFCE, we have to jointly take the parallel block construction, physical node embedding, and routing process into account. As optimizing processing delay and propagation delay are interrelated, it is essential to identify physical node candidates that can facilitate the joint optimization of both delays.

A. Parallelism-aware Betweenness Centrality (PBC)

The parallel block gain (PBG) in Section VI is defined as the processing delay saved from the parallel blocks when computing resources are enough at each physical node. Now, there might not exist a physical node n with enough computing resources to host the parallel block constructed from the maximum PBG strategy. Accordingly, to evaluate the processing delay impact of a physical node, we define PBG'_n in Eq. (23), where V^{sub} is a set of parallelizable SFs that can be jointly hosted at n and $\mathbb{L}_{V^{sub}}$ is processing delay of the parallel block constructed by the SFs in V^{sub}. This PBG'_n measures how much a physical node n can facilitate saving processing delay by considering (i) the number of parallelizable SFs that n can host and (ii) the processing delay of the parallel block constructed by the SFs that are co-hosted at n.

$$PBG'_n = \frac{|V^{sub}|}{\mathbb{L}_{V^{sub}}} \tag{23}$$
$$\Sigma_{v \in V^{sub}} C_v \leq C_n, l_{u,v} \in E, \forall u, v \in V^{sub}$$

Similarly, to optimize SFP propagation delay, we need the scheme to measure how much propagation delay a physical node n might introduce when it is selected to host a parallel block. Here, we define potential propagation delay ($\mathbb{L}_{PPD_n}$) to evaluate the propagation delay when selecting n to construct SFP as Eq. (24), where Δ is the set of physical nodes that have been added into SFP. Initially, $\Delta = \{s, d\}$.

$$\mathbb{L}_{PPD_n} = min_{n_i, n_j \in \Delta} \quad \mathbb{L}_{P_{n,n_i}} + \mathbb{L}_{P_{n,n_j}}, \quad n_i \neq n_j \tag{24}$$

Traditionally, to optimize traffic latency, betweenness centrality (BC) measures the node centrality based on the number of shortest paths passing the physical node. Likewise, we propose parallelism-based latency betweenness centrality (PBC) in Eq. (25) to measure the node centrality of how much a physical node can facilitate deploying the parallel blocks with low processing and propagation delays. Note that α and β are two coefficient values to balance the optimization priority between processing delay and propagation delay. A physical node with a high PBC value means it enables great parallelisms and yields a low potential latency (i.e., the sum of potential SFs' processing delay and potential propagation delay).

$$PBC_n = max_{V^{sub} \subset V} \frac{|V^{sub}|}{\alpha * \mathbb{L}_{V^{sub}} + \beta * \mathbb{L}_{PPD_n}} \tag{25}$$

B. Parallelism-aware SFs Deployment (PSFD) Algorithm

Based on the technique of parallelism-based latency betweenness centrality, now we propose the parallelism-aware SFs deployment (PSFD) algorithm to efficiently accommodate a given P-NSR on the physical network while jointly taking the parallel block construction, physical node embedding, and routing process into account.

To begin, PSFD initializes a node set Δ with $\{s, d\}$. Then, PSFD repeats the following operations until all SFs are satisfied: (i) update PBC value of each physical node; (ii) select the node δ whose PBC value is highest and add it to Δ; (iii) create a parallel block that includes the set of SFs (V^{sub}) and maximizes the PBC value at δ, and embeds it onto δ; and (iv) remove the set of embedded SFs from V. Note that, to further utilize the available computing resources remaining in the selected physical nodes, if a physical node n is already in Δ, the $\mathbb{L}_{PPD}$ value of n is counted as 0. Next, the PSFD algorithm repeats the following operations until all the parallel blocks are embedded: (i) find physical node x in Δ which has the smallest propagation delay to the SFP endpoint node n_e; and (ii) add x to $\mathbb{S}$, set $n_e = x$, and remove x from Δ. At last, the PSFD algorithm constructs the SFP by connecting each pair of adjacent physical nodes in $\mathbb{S}$ and formulates the corresponding P-SFC.

Algorithm 2 PSFD Algorithm

1: **Input:** $P\text{-}NSR$, PN;
2: **Output:** SFP, P-SFC;
3: Initialize Δ = {s,d};
4: **while** $V \neq \emptyset$ **do**
5: Update PBC value of each physical node based on Eq. (25), if a node n is already in Δ, $\mathbb{L}_{PPD_n}$ is 0;
6: Select physical node δ with highest PBC value, and add δ to Δ;
7: Create a parallel block including the set of SFs (V^{sub}) that maximize the PBC value of δ, and embed the SFs in the constructed parallel block onto δ;
8: Remove the set of SFs in V^{sub} from V;
9: **end while**
10: Initialize $\mathbb{S} = \emptyset$, set the SFP endpoint node n_e as s, add n_e to SFP;
11: **while** $\Delta \neq \emptyset$ **do**
12: Find node x in Δ with the smallest $\mathbb{L}_{P_{x,n_e}}$;
13: Add x to $\mathbb{S}$, set $n_e = x$, and $\Delta = \Delta - x$;
14: **end while**
15: Add d to $\mathbb{S}$, construct the SFP by connecting each pair of adjacent nodes in $\mathbb{S}$, and form the corresponding P-SFC;
return SFP, P-SFC;

C. Bound Analysis of PSFD Algorithm

Next, we prove that the PSFD algorithm is logarithm-approximate. Table III lists the notations used in the proof.

TABLE II: Notation Table for Approximation Proof.

Notation	Definition
δ_i	Physical node selected in i^{th} iteration.
$V^{sub}_{\delta_i}$	Set of SFs instantiated at physical node δ_i.
V^{left}_i	Set of unsatisfied SFs in i^{th} iteration.
$\mathbb{L}_{PPD_{\delta_i}}$	Propagation delay in i^{th} iteration.
$\mathbb{L}_{V^{sub}_{\delta_i}}$	Processing delay in i^{th} iteration.
SFP^{OPT}_i	Optimal SFP in i^{th} iteration.
$\|\zeta_i\|$	Number of physical nodes in SFP^{OPT}_i.
$\|V\|$	Number of SFs required by P-NSR.
k	Minimum number of SFs satisfied among any iteration.
$\|T\|$	Total number of iterations to satisfy the P-NSR.

Theorem 5. *When α and β in Eq. (25) are both 1, the PSFD algorithm is logarithm-approximate.*

Proof. Eq. (26) shows the relationships between the number of satisfied SFs and the unsatisfied SFs in adjacent iterations (i.e., iterations i and $i+1$).

$$|V_i^{left}| = |V_{\delta_i}^{sub}| + |V_{i+1}^{left}| \tag{26}$$

As PSFD picks the physical node with the highest PBC value, PBC_{δ_i} is greater than the average PBC value of the physical nodes in the optimal SFP. Then, Eq. (27) holds.

$$\frac{|V_{\delta_i}^{sub}|}{\mathbb{L}_{PPD_{\delta_i}} + \mathbb{L}_{V_{\delta_i}^{sub}}} \geq \frac{\sum_{m \in SFP_i^{OPT}} \frac{|V_m^{sub}|}{\mathbb{L}_{PPD_m} + \mathbb{L}_{V_m^{sub}}}}{|\zeta_i|} \tag{27}$$

For a physical node m in SFP_i^{OPT}, to maximize the PBC value, the sum of $\mathbb{L}_{PPD_m}$ and $\mathbb{L}_{V_m^{sub}}$ is not greater than the latency of the SFP_i^{OPT} as shown in Eq. (28).

$$\mathbb{L}_{PPD_m} + \mathbb{L}_{V_m^{sub}} \leq \mathbb{L}_{\text{SFP}_i^{OPT}}, \forall m \in SFP_i^{OPT} \tag{28}$$

Since different physical nodes can provide the same SF instance(s) to satisfy the P-NSR, Eq. (29) holds.

$$\sum_{m \in SFP_i^{OPT}} |V_m^{sub}| \geq |V_i^{left}| \tag{29}$$

When combining Eqs. (27) - (29), we have Eq. (30).

$$\mathbb{L}_{PPD_{\delta_i}} + \mathbb{L}_{V_{\delta_i}^{sub}} \leq |V_{\delta_i}^{sub}| * \frac{\mathbb{L}_{\text{SFP}_i^{OPT}} * |\zeta_i|}{|V_i^{left}| * |\zeta_i|} \tag{30}$$

According to Eq. (26), Eq. (31) holds.

$$\frac{|V_{\delta_i}^{sub}|}{|V_i^{left}|} \leq \frac{1}{|V_i^{left}|} + \frac{1}{|V_i^{left}| - 1} + ... + \frac{1}{|V_{i+1}^{left}|} \leq \sum_{\tau=|V_{i+1}^{left}|}^{|V_i^{left}|} \frac{1}{\tau} \tag{31}$$

When summing up all the iterations, we have Eq. (32), which combines Eqs. (30) and (31).

$$\sum_{i=1}^{|T|} \mathbb{L}_{PPD_{\delta_i}} + \mathbb{L}_{V_{\delta_i}^{sub}} \leq \sum_{\tau=|V_1^{left}|}^{|V_{|T|}^{left}|} \frac{1}{\tau} * \mathbb{L}_{\text{SFP}_i^{OPT}} \tag{32}$$

According to the properties of the harmonic series, Eq. (33) hold.

$$\sum_{i=1}^{|T|} \frac{1}{\tau} = \frac{1}{|V|} + \frac{1}{|V| - |V_{\delta_1}^{sub}|} + ... + \frac{1}{|V_{\delta_{|T|}}^{left}|}$$

$$\leq \frac{1}{|V|} + \frac{1}{|V| - k} + \frac{1}{|V| - 2 * k} + ... + 1 \leq \frac{ln(V)}{k} + 1 \tag{33}$$

Since the SFP_1^{OPT} includes all the required SFs, for any SFP_i^{OPT} $(i > 1)$, Eq. (34) and Eq. (35) hold.

$$\mathbb{L}_{\text{SFP}_1^{OPT}} > \mathbb{L}_{\text{SFP}_i^{OPT}}, \ \forall i > 1 \tag{34}$$

$$\sum_{i=1}^{|T|} \mathbb{L}_{PPD_{\delta_i}} + \mathbb{L}_{V_{\delta_i}^{sub}} \leq (\frac{ln(|V|)}{k} + 1) * \mathbb{L}_{\text{SFP}_1^{OPT}} \tag{35}$$

Based on Eq. (24), the propagation delay of the created SFP is less than the sum of $\mathbb{L}_{PPD_{\delta_i}}$. Thus, Eq. (36) holds.

$$\mathbb{L}_{\text{SFP}_{PSFD}} < \sum_{i=1}^{|T|} \mathbb{L}_{PPD_{\delta_i}} + \mathbb{L}_{V_{\delta_i}^{sub}} \leq (\frac{ln(|V|)}{k} + 1) * \mathbb{L}_{\text{SFP}_1^{OPT}} \tag{36}$$

Therefore, the proposed PSFD algorithm achieves a logarithm-approximation performance. □

VIII. Performance Evaluation

A. Simulation Settings

To evaluate the performances of the proposed schemes, we conduct our experiments in three network scenarios: (i) 24-node-43-link USNET [55] (as Fig. 5), (ii) edge-cloud system [56], and (iii) random network. If not otherwise specified, the parameters are generated in terms of the state-of-the-art simulation settings as follows [13], [15], [19], [57], [58]. Each physical node's amount of computing resources are randomly set in a discrete-uniform range [20, 100] gigabit (Gb). Each physical link has a bandwidth in a discrete-uniform range [5, 10] gigabit per second (Gbps). The propagation delay of a physical link is randomly set in a uniform range [10, 50] microseconds (μs). In P-NSR, the number of SF nodes is randomly set in a discrete-uniform range [5, 20], edge density of PG is randomly set in a uniform range [0, 1], and bandwidth demand is random in a discrete-uniform range [1, 5] Gbps. Each SF node requires computing resources in a discrete-uniform range [10, 25] Gb and needs the maximum processing delay in a uniform range [10, 30] μs. The service source and destination of the P-NSR are randomly selected from the PN. The coefficient values α and β in PBC calculations are both set to 1. It is worth noting that the edge density of a PG is estimated by the ratio between the number of edges existing in this PG and the maximum number of edges that this PG can have (i.e., the number of edges in the complete graph for this PG). We evaluate the performances of the proposed algorithms in terms of (i) propagation delay, (ii) processing delay, (iii) end-to-end latency (i.e., the sum of overall processing delay and propagation delay), (iv) number of accepted requests, and (v) resource utilization ratio.

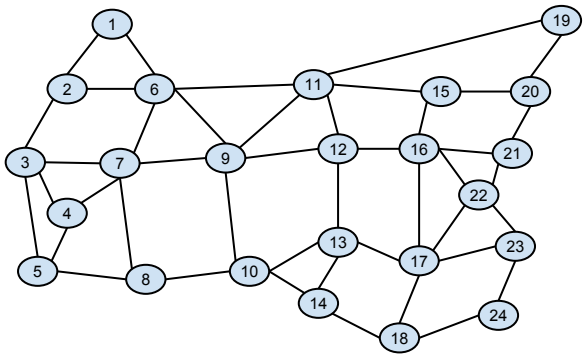
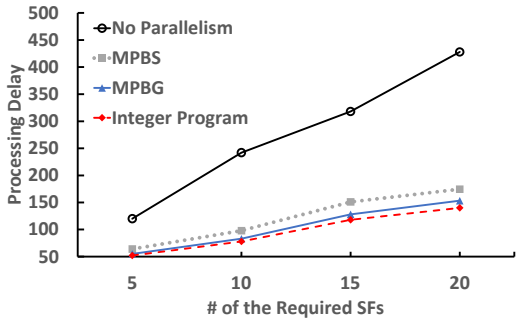
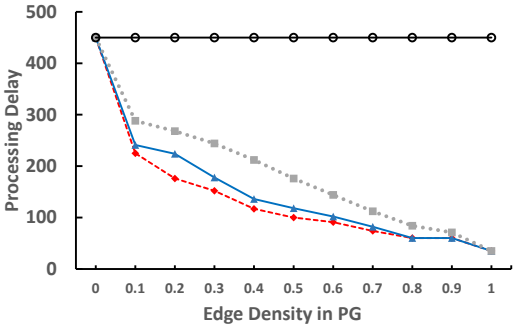


Fig. 5: Topology of 24-node-43-link USNET [55].



(a) Processing delay vs. $|V|$.

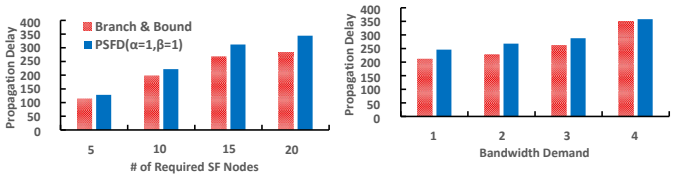


(b) Processing delay vs. edge density.

Fig. 6: Impact of SF parallelisms.

B. Performance Bound Analysis

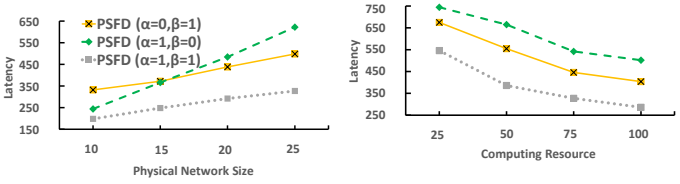
To demonstrate the impact of SF parallelisms, we set each physical node with enough computing resources. As a result, propagation delay can be optimized by embedding all SFs onto one physical node along the bandwidth-aware shortest path. To minimize processing delay, we implement the integer program model under the constraints in Eqs. (5) - (11) to obtain the optimal processing delay of a P-SFC. We also implement the maximum parallel block size (MPBS) first optimization algorithm, which replaces Eq. (18) as $PBG_\xi = |\xi|$ (i.e., maximizes the size of each parallel block). In Fig. 6, we show the performances of the integer program, MPBG, MPBS, and no parallelism (i.e., $\mathbb{L}_{\text{SFC}}$ in Eq. (20)), which are denoted by red rhombus-dashed, blue triangle-solid, grey square-dotted, and dark circled-solid curves, respectively. In Fig. 6a, when increasing the number of SFs, all schemes need more processing delay to construct the P-SFC. This is because more required SFs likely construct more parallel blocks, thereby increasing the processing delay. As MPBG creates the parallel block that saves the most processing delay at each iteration, it has near-optimal performance. In Fig. 6b, when fixing the number of required SFs as 20 and increasing the edge density in PG, all three proposed schemes need less processing delay, except that no-parallelism scheme always needs the same processing delay. Note that, when edge density is 0 (i.e., no parallelism) and 1 (i.e., all SFs can work in parallel), integer programming, MPBG and MPBS have the same performance. When edge density is high (i.e., more than 0.6), MPBG has a similar performance as the integer program. High edge densities increase the connectivities of PG and reduce the number of maximal cliques in PG. Therefore, with a limited number of maximal cliques in PG, MPBG, achieves near-optimal performance. As shown in Fig. 3, assigning more parallelizable SFs in one parallel block reduces processing delay to construct P-SFC, but may not maximize PBG of P-SFC in Eq. (19). Thus, when the edge density is 0.4, MPBG outperforms MPBS by as much as 58% in our examples.



(a) Propagation delay vs. $|V|$. (b) Propagation delay vs. BW demand.

Fig. 7: Impact of routing.

To show the impact of routing, each physical node is set to host only one SF. As a result, the optimization problem of PSFCE becomes finding the best path connecting the service source and destination while hosting all required SFs. We implement the branch-and-bound (B&B) method to obtain the optimal path with the least propagation delay. Fig. 7 shows the performances of B&B and PSFD (α=1,β=1), which are denoted by the red and blue bars, respectively. In Fig. 7a, when increasing the number of required SF nodes, both schemes need more propagation delay to construct the SFP. This is because more required SFs lead to more physical nodes in the SFP, thereby increasing the propagation delay. Similarly, when fixing the number of SF nodes as 15 and increasing the bandwidth demand as shown in Fig. 7b, both the B&B method and PSFD need more propagation delay. From Figs. 7a and 7b, we see that the PSFD algorithm achieves near-optimal performance when the number of required SFs is small or the bandwidth demand is high. According to Eq. (24), when $|V| = 1$, PSFD can construct the optimal SFP by applying the PBC technique. Similarly, when increasing the bandwidth demand, the number of bandwidth-aware shortest paths reduces. The SFP created by applying the PBC technique is likely the same as the one created by the B&B method. Thus, with higher bandwidth demand, the performance of PSFD is closer to the optimal.



(a) Latency vs. PN size. (b) Latency vs. computing resources.

Fig. 8: Joint impact of SF parallelism and routing process.

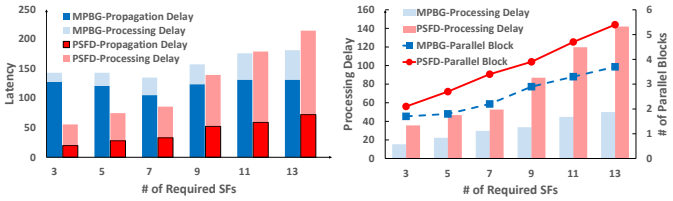
We conduct experiments to evaluate the joint impact of SF parallelisms and routing by varying the physical network size (i.e., number of physical nodes and links in the network) and computing resources at each physical node in Fig. 8. Here, we compare the performances of PSFD (α=1,β=0), PSFD (α=0,β=1), and PSFD (α=1,β=1); where PSFD (α=1,β=0) fo-

cuses on the optimization of SFs' processing delay and PSFD (α=0,β=1) concentrates on optimizing SFP propagation delay. In Fig. 8, performance of PSFD (α=0,β=1), PSFD (α=1,β=0), and PSFD (α=1,β=1) are denoted by the yellow-solid, green-dashed, and grey-dotted curves, respectively. In Fig. 8a, when increasing the physical network size, all schemes require more latency to construct SFP. This is because the service source is likely far away from the destination in a larger network, which increases SFP propagation delay. As PSFD (α=1,β=1) takes optimization of processing delay and propagation delay into account, the physical node selected to embed the parallel block by PSFD (α=1,β=1) will yield the smallest sum of processing and propagation delays. Interestingly, when comparing PSFD (α=1,β=0) with PSFD (α=0, β=1) in Fig. 8a, we can see that PSFD (α=1, β=0) is better in smaller PNs and PSFD (α=0, β=1) is better in larger PNs. This indicates that the optimization of the end-to-end service latency largely depends on SFs' processing delay in small networks, while end-to-end service latency is impacted more by SFP propagation delay in large networks. In Fig. 8b, when fixing the number of required SFs as 15 and increasing the computing resources at each physical node, all schemes need less latency. This is because, with more computing resources, more parallelisms are enabled to save processing delay, and fewer physical nodes are required in SFP to host all SFs, which can save on propagation delay. When investigating the joint impact of SF parallelism and routing processes, the proposed PSFD (α=1,β=1) algorithm outperforms PSFD (α=1,β=0) and PSFD (α=0,β=1) by an average of 52% and 36%, respectively, in our examples.

C. *Performance Evaluation in Edge-Cloud Systems*

In this section, we evaluate the performances of the proposed schemes in edge-cloud systems. Similar to the network topology in [56], the cloud is directly connected to the clients' access network via fiber links, while edge servers are distributed around clients. The propagation delay from a client to the cloud is in a uniform range [50, 100] μs, and the propagation delay between edge servers and clients is in a uniform range [5, 15] μs. Servers in the cloud have high-performance computing hardware and the processing delay of running an SF is in a uniform range [5, 15] μs. The edge servers own low-performance computing hardware and the processing delay of running an SF is in a uniform range [10, 30] μs. Additionally, the cloud has enough computing resources to accommodate any P-NSR, while the edge has the limited number of servers in a discrete-uniform range [20, 40]. The edge servers are interconnected with a 60% probability. Notably, for each request, it is accommodated (i) in the cloud (with enough computing resources) by MPBG and (ii) at the edge (with limited computing resources) by PSFD.

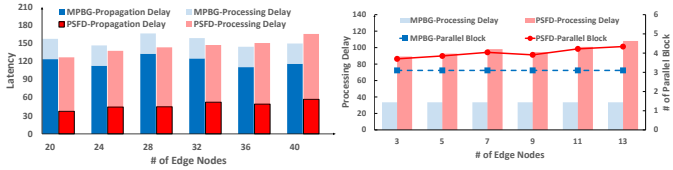
Figs. 9 and 10 show the performances of MPBG and PSFD regarding the impact of the SFC length and the number of edge nodes, respectively. In Figs. 9 and 10, the dark (light) blue and red bars represent the propagation (processing) delay of applying MPBG and PSFD to accommodate the request, respectively, while the blue-dashed and red-solid curves represent the number of parallel blocks created by MPBG and PSFD, respectively.



(a) Latency vs. # of required SFs. (b) $\mathbb{L}_{\mathbb{PC}}$ vs. # of required SFs.

Fig. 9: Impact of SFC length on MPBG and PSFD.

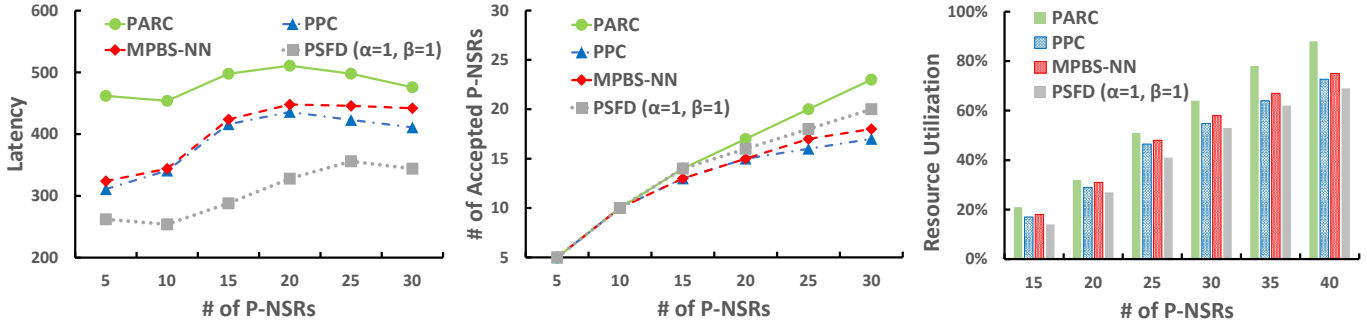
When increasing the number of required SFs, both algorithms need more service latency to deliver the P-NSR, as shown in Fig. 9a. For PSFD at the edge, both propagation and processing delays increase with the number of required SFs as more parallel blocks are created. This is because (i) more parallel blocks likely lead to a higher processing delay according to Eq. (2), and (ii) more edge servers are needed to host parallel blocks, leading to a higher propagation delay for connections. Interestingly, for MPBG in the cloud, the processing delay increases with the number of required SFs while the propagation delay fluctuates. This can be explained as follows. More SFs are likely forming more parallel blocks, leading to higher processing delay. As the source and destination nodes are randomly located, the propagation delay fluctuates. Fig. 9b further verifies the above observations and analysis. With enough computing resources and high-performance hardware in the cloud, MPBG creates fewer parallel blocks and needs much lower processing delay compared to running PSFD at the edge. Overall, deploying services at the edge by PSFD is better when the number of required SFs is smaller (less than 10), while deploying services in the cloud by MPBG is more latency-efficient when the number of required SFs is larger. Since the SFC length is no more than 8 in practice [47], most P-NSRs should be deployed at the edge to achieve latency-efficient service delivery.



(a) Latency vs. # of edge nodes. (b) $\mathbb{L}_{\mathbb{PC}}$ vs. # of edge nodes.

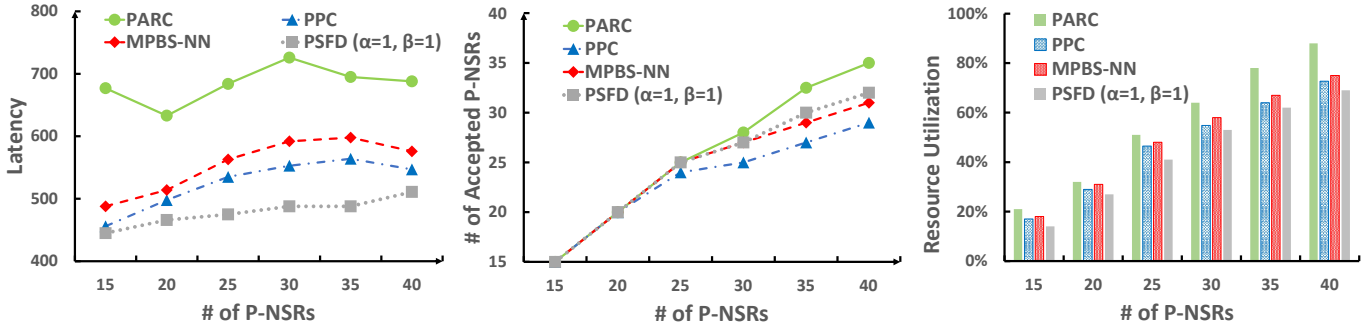
Fig. 10: Impact of # of edge nodes on MPBG and PSFD.

In Fig. 10, when increasing the number of edge nodes, the latency of deploying services by MPBG in the cloud varies, while the latency of deploying services by PSFD at the edge increases. Again, as the source and destination nodes are randomly selected from the edge nodes, the propagation delay of deploying services by MPBG in the cloud fluctuates. Enlarging the size of the edge network likely makes source and destination far away from each other, leading to a higher propagation delay in PSFD. Since the same set of SFs is required, the processing delay of deploying services by MPBG in the cloud remains the same. When enlarging the network size, as different set of edge servers are employed to host SFs in a P-NSR, the processing delay of deploying services by PSFD at the edge fluctuates.



(a) Latency vs. # of P-NSR. (b) Acceptance vs. # of P-NSR. (c) Resource utilization vs. # of P-NSR.

Fig. 11: Performance in 24-node-43-link USNET.



(a) Latency vs. # of P-NSR. (b) Acceptance vs. # of P-NSR. (c) Resource utilization vs. # of P-NSR.

Fig. 12: Performance in 40-node-180-link mesh network.

D. Performance Evaluation with Online Traffic

In this section, we evaluate PSFD when P-NSRs are generated in an online fashion and will stay in the network once accommodated. We use 24-node-43-link USNET and 40-node-180-link randomly-generated mesh network topology in our experiments. Note that the random network is generated based on JAVA language by first generating a tree with 40 nodes and then adding links to ensure the existence of 180 links in the network. The computing resources of each physical node and bandwidth resources of each physical link are randomly set in discrete-uniform ranges [100, 300] Gb and [50, 100] Gbps, respectively. The number of required SFs is set to {5, 10, 15, 20, 25} with an even distribution [18], [19], [44].

We implement and improve the proposed MPBS algorithm, partial parallel chaining (PPC) [45], and the parallelism-aware residual capacity first placement (PARC) algorithm [44] to compare with PSFD. We name the algorithm improved from MPBS as MPBS with nearest-neighbor (MPBS-NN). MPBS-NN is implemented by (i) selecting the node with the highest betweenness centrality value from source to the destination; (ii) creating and embedding the largest parallel block that this physical node can host; (iii) repeating steps (i) and (ii) until all SFs are embedded; and (iv) applying the nearest-neighbor algorithm to connect the source with the destination going through all physical nodes that host parallel block(s). We extend PPC by (i) generating all possible size vectors to indicate all possible P-SFCs; (ii) selecting one with the least processing delay; (iii) if PN can accommodate the constructed P-SFC, creating the corresponding parallelism-aware layered graph, otherwise, denying the request; (iv) if the parallelism-aware layered graph is created by step (iii), finding the shortest path in the constructed layered graph as the SFP. The PARC is extended by (i) randomly generating an SFC based on P-NSR, and setting the source s as the endpoint n_e; (ii) creating the P-SFC with least processing latency by exhaustive search; (iii) embedding as many parallel blocks as possible to physical node δ with the highest residual computing resources, connecting δ to n_e, and updating n_e as δ; (iv) repeating step (iii) until all SFs are embedded; and (v) connecting n_e with the destination to formulate the SFP. Note that, if the extended algorithms cannot accommodate an incoming P-NSR, it will be denied and counted as unaccepted. We evaluate the performances of PSFD, MPBS-NN, PPC, and PARC regarding: (i) service end-to-end latency, (ii) number of accepted P-NSRs, and (iii) resource utilization ratio.

Figs. 11 and 12 present the performances of PSFD, MPBS-NN, PPC, and PARC in USNET and 40-node network, respectively. The latency and acceptance performances of PARC, MPBS-NN, PPC, PSFD (α=1, β=1) are denoted by the green circled-solid, red rhombus-dashed, blue triangle-dashed, and grey square-dotted curves, respectively. The resource utilization performances of those algorithms are represented by the green bar, red bar, blue bar, and grey bar, respectively.

In Fig. 11a, when increasing the number of P-NSRs, the performances of all schemes show the trend of increasing at first and then decreasing. When the algorithms are embedding more P-NSRs to the PN, the network resources of some key links or nodes are running out. Then, the algorithm needs to accommodate a P-NSR via a longer SFP, thus increasing the average service latency. As the network resources

are running out, all schemes begin accommodating the P-NSRs with fewer SFs while denying the P-NSRs with many SFs. As the SFP generated for a small number of SFs is generally short, the average service latency of all SFPs later decreases. Even though the PARC algorithm creates the P-SFC with the least processing latency using exhaustive search, the embedding policy of PARC will accommodate the parallel block on the physical node with many residual computing resources, which likely will introduce detour routing processes, thus leading to high service latency. MPBS-NN creates large parallel blocks and embeds the block on the physical node with high betweenness centrality value to reduce both processing and propagation delays. When the system load is low, MPBS-NN accommodates P-NSRs with a relatively lower service latency than PARC does. When more P-NSRs are accommodated, the network resources of the links and nodes with high betweenness centrality value runs out, and MPBS-NN has to employ longer routing paths for creating SFPs, leading to a sharp increase in average service latency. PPC generates the P-SFC with the least processing delay and accommodates it by establishing a layered graph to reduce the propagation latency. It outperforms PARC and MPBS-NN according to the service latency. However, the processing delay and propagation delay are independently optimized. A large parallel block might be deployed far away from the others, leading to a relatively higher overall latency than PSFD. PSFD jointly takes processing delay, propagation delay, and computing resources usage (i.e., V^{sub}) into account and can dynamically pick the better-fit physical node to reduce the service latency. In Figs. 11b and 11c, PARC has the highest acceptance and resource utilization. Meanwhile, PPC has the worst acceptance. This is because PPC greedily creates large parallel blocks to optimize processing delay, and there might not exist a physical node with enough computing resources to host such large blocks. Interestingly, MPBS-NN accepts fewer P-NSRs but has higher resource utilization compared to PSFD. This is because the static path-selection method (i.e., selecting the node with the highest betweenness centrality value) will quickly fill capacities of key links and nodes, which will lead the later P-NSRs to be accommodated via detours, thus leading to higher service latency and resource utilization.

Fig. 12 shows the performances of PSFD, MPBS-NN, PPC, and PARC in the 40-node mesh network. Similarly, PSFD outperforms MPBS-NN, PPC, and PARC in service latency, while PARC has the highest acceptance and resource utilization. It is worth noting that all schemes in this larger network have higher acceptance than USNET. Higher connectivities mitigate the pressure of running out of bandwidth resources for key links because more physical links can be used for substitutions. Numerically, in our examples, the average acceptance ratios for all four schemes are 66.7% and 80.6% in USNET and 40-node networks, respectively. In addition, the performance gap between PSFD and MPBS-NN/PPC/PARC in Fig. 11a and Fig. 12a becomes smaller. Regarding the service latency in our examples, PSFD, on average, outperforms MPBS-NN by 32% and 15%, PPC by 29.4% and 10.1%, and PARC by 58% and 42% in USNET and 40-node network, respectively.

IX. Conclusion

This paper comprehensively investigated how to minimize the latency in the parallelism-aware service function chaining and embedding (PSFCE) problem. When each physical node has enough computing resources to host all required SFs, SFP propagation delay can be optimized by embedding the required SFs along the bandwidth-aware shortest path. We have proposed the maximum parallel block gain (MPBG) first optimization algorithm to efficiently create a parallelism-based SFC (P-SFC) with a low processing delay. When computing resources at each physical node are limited such that the required SFs have to be accommodated by multiple physical nodes, we proposed the parallelism-aware SF deployment (PSFD) algorithm to jointly optimize processing and propagation delays. Through thorough analysis, we showed that PSFD is in general logarithm-approximate. For different network settings, we have shown that PSFD can effectively optimize the end-to-end latency and outperform the schemes directly extended from existing works. Additionally, we had the following findings: (i) MPBG achieves near-optimal performance when computing resources are enough at every physical node, (ii) when the computing resources are limited, the end-to-end latency largely depends on the optimization of SFs' processing delay in small networks, and (iii) to achieve latency-efficient, short P-SFCs (with less than 10 SFs) should be deployed at the edge, while long P-SFCs should be deployed in the cloud. Future work should investigate the end-to-end service latency optimization problem of P-SFC composition and embedding when considering SF dependencies and parallel relationships for latency-deterministic network scenarios.

References

[1] "ETSI GS NFV-MAN 001: Network functions virtualisation (NFV); management and orchestration," *v. 1.1.1*, Dec. 2014.

[2] J. Halpern and C. Pignataro, "Service function chaining (SFC) architecture," https://tools.ietf.org/html/rfc7665, 2015.

[3] P. Quinn and T. Nadeau, "Problem statement for service function chaining," https://tools.ietf.org/html/rfc7498, 2015.

[4] G. Li, Y. R. Yang, F. Le, Y. Lim, and J. Wang, "Update algebra: Toward continuous, non-blocking composition of network updates in SDN," in *Proc. IEEE INFOCOM*, 2019, pp. 1081–1089.

[5] G. Jung, P. Rahimzadeh, Z. Liu, S. Ha, K. Joshi, and M. A. Hiltunen, "Virtual redundancy for active-standby cloud applications," in *Proc. IEEE INFOCOM*, 2018, pp. 1916–1924.

[6] Y. Wu, H. Huang, C. Wang, and Y. Pan, *5G-Enabled Internet of Things*. CRC Press, 2019.

[7] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "NFV and SDN key technology enablers for 5G networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2468–2478, 2017.

[8] A. A. Ahmed and A. A. Alzahrani, "A comprehensive survey on handover management for vehicular ad-hoc network based on 5G mobile networks technology," *Trans. Emerg. Telecommun. Technol.*, vol. 30, no. 3, 2019.

[9] E. Yaacoub and M. Alouini, "A key 6G challenge and opportunity - connecting the base of the pyramid: A survey on rural connectivity," *P. IEEE*, vol. 108, no. 4, pp. 533–582, 2020.

[10] L. Ruan, M. P. I. Dias, and E. Wong, "Towards self-adaptive bandwidth allocation for low-latency communications with reinforcement learning," *Opt. Switch. Netw.*, vol. 37, p. 100567, 2020.

[11] H. Pang, C. Zhang, F. Wang, J. Liu, and L. Sun, "Towards low latency multi-viewpoint 360 interactive video: A multimodal deep reinforcement learning approach," in *Proc. IEEE INFOCOM*, 2019, pp. 991–999.

[12] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing: A key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.

[13] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling network function parallelism in NFV," in *Proc. ACM SIGCOMM*, 2017, pp. 43–56.

[14] I. Tomkos, D. Klonidis, E. Pikasis, and S. Theodoridis, "Toward the 6G network era: Opportunities and challenges," *IT Prof.*, vol. 22, no. 1, pp. 34–38, 2020.

[15] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 643–655, 2019.

[16] R. Yu, G. Xue, and X. Zhang, "QoS-aware and reliable traffic steering for service function chaining in mobile networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2522–2531, 2017.

[17] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest path and maximum flow problems under service function chaining constraints," in *Proc. IEEE INFOCOM*, 2018, pp. 2132–2140.

[18] D. Zheng, C. Peng, X. Liao, and X. Cao, "Towards optimal hybrid service function chain embedding in multi-access edge computing," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6035–6045, 2020.

[19] D. Zheng, C. Peng, X. Liao, L. Tian, G. Luo, and X. Cao, "Towards latency optimization in hybrid service function chain composition and embedding," in *Proc. IEEE INFOCOM*, 2020, pp. 1–10.

[20] A. Hmaity, M. Savi, L. Askari, F. Musumeci, M. Tornatore, and A. Pattavina, "Latency- and capacity-aware placement of chained virtual network functions in FMC metro networks," *Opt. Switch. Netw.*, vol. 35, 2020.

[21] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, "Latency-aware VNF chain deployment with efficient resource reuse at network edge," in *Proc. IEEE INFOCOM*, 2020, pp. 1–10.

[22] X. Shang, Y. Huang, Z. Liu, and Y. Yang, "Reducing the service function chain bakcup cost over the edge and cloud by a self-adapting scheme," in *Proc. IEEE INFOCOM*, 2020, pp. 1–10.

[23] G. Sallam and B. Ji, "Joint placement and allocation of virtual network functions with budget and capacity constraints," in *Proc. IEEE INFOCOM*, 2019, pp. 523–531.

[24] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *Proc. IEEE INFOCOM*, 2019, pp. 514–522.

[25] T. Kuo, B. Liou, K. C. Lin, and M. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1562–1576, 2018.

[26] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE INFOCOM*, 2015, pp. 1346–1354.

[27] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *Proc. IEEE INFOCOM*, 2018, pp. 486–494.

[28] J. Baranda, G. Avino, J. Mangues-Bafalluy, L. Vettori, R. Martínez, C. F. Chiasserini, C. Casetti, P. Bande, M. Giordanino, and M. Zanzola, "Automated deployment and scaling of automotive safety services in 5G-transformer," in *Proc. IEEE NFV-SDN*, 2019, pp. 1–2.

[29] A. Sheoran, X. Bu, L. Cao, P. Sharma, and S. Fahmy, "An empirical case for container-driven fine-grained VNF resource flexing," in *Proc. IEEE NFV-SDN*, 2016, pp. 121–127.

[30] T.-M. Nguyen, M. Minoux, and S. Fdida, "Optimizing resource utilization in NFV dynamic systems: New exact and heuristic approaches," *Comput. Netw.*, vol. 148, pp. 129–141, 2019.

[31] A. Sheoran, P. Sharma, S. Fahmy, and V. Saxena, "Contain-ed: An NFV micro-service system for containing E2E latency," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 5, pp. 54–60, 2017.

[32] Z. Luo, C. Wu, Z. Li, and W. Zhou, "Scaling geo-distributed network function chains: A prediction and learning framework," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1838–1850, 2019.

[33] D. Harutyunyan, N. Shahriar, R. Boutaba, and R. Riggio, "Latency and mobility-aware service function chain placement in 5G networks," *IEEE Trans. Mob. Comput.*, pp. 1–1, 2020.

[34] B. Yi, X. Wang, K. Li, S. K. Das, and M. Huang, "A comprehensive survey of network function virtualization," *Comput. Netw.*, vol. 133, pp. 212–262, 2018.

[35] Cisco, "Cisco NFV solution: Enabling rapid service innovation in the era of virtualization," https://www.cisco.com/c/dam/global/shared/assets/pdf/sp04/nfv-solution.pdf, 2014.

[36] Huawei-Industrial-Base, "White paper - Huawei observation to NFV," https://www.huawei.com/ilink/en/download/HW_399662, 2014.

[37] J. Krolikowski, S. Martin, P. Medagliani, J. Leguay, S. Chen, X. Chang, and X. Geng, "Joint routing and scheduling for large-scale deterministic IP networks," *Comput. Commun.*, vol. 165, pp. 33–42, 2021.

[38] B. Martini, F. Paganelli, P. Cappanera, S. Turchi, and P. Castoldi, "Latency-aware composition of virtual functions in 5g," in *Proc. IEEE NetSoft*, 2015, pp. 1–6.

[39] W. Bao, D. Yuan, B. B. Zhou, and A. Y. Zomaya, "Prune and plant: Efficient placement and parallelism of virtual network functions," *IEEE Trans. Computers*, vol. 69, no. 6, pp. 800–811, 2020.

[40] J. Cai, Z. Huang, J. Luo, Y. Liu, H. Zhao, and L. Liao, "Composing and deploying parallelized service function chains," *J. Netw. Comput. Appl.*, vol. 163, p. 102637, 2020.

[41] X. Lin, D. Guo, Y. Shen, G. Tang, and B. Ren, "DAG-SFC: minimize the embedding cost of SFC with parallel VNFs," in *Proc. ACM ICPP*, 2018, pp. 15:1–15:10.

[42] R. Wang, J. Luo, F. Dong, and D. Shen, "ParaNF: Enabling delay-balanced network function parallelism in NFV," in *Proc. IEEE CSCWD*, 2019, pp. 392–397.

[43] Y. Zhang, Z. Zhang, and B. Han, "Hybridsfc: Accelerating service function chains with parallelism," in *Proc. IEEE NFV-SDN*, 2019, pp. 1–7.

[44] S. Xie, J. Ma, and J. Zhao, "Flexchain: Bridging parallelism and placement for service function chains," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 1, pp. 195–208, 2021.

[45] I. Lin, Y. Yeh, and K. C. Lin, "Toward optimal partial parallelization for service function chaining," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2033–2044, 2021.

[46] D. Shen, H. Liu, R. Wang, F. Dong, and F. Li, "Paragraph: Subgraph-Level network function composition with delay balanced parallelism," *IEEE Access*, vol. 8, pp. 199 308–199 322, 2020.

[47] S. Kumar, C. Obediente, M. Tufail, S. Majee, and C. Captari, "Service function chaining use cases in data centers," https://datatracker.ietf.org/doc/html/draft-kumar-sfc-dc-use-cases-02, 2014.

[48] B. Zhang and H. Mouftah, "Fast bandwidth-constrained shortest path routing algorithm," *IEE Proc. Commun.*, vol. 153, no. 5, pp. 671–674, 2006.

[49] W. Sun, J. Hao, X. Lai, and Q. Wu, "Adaptive feasible and infeasible TABU search for weighted vertex coloring," *Inf. Sci.*, vol. 466, pp. 203–219, 2018.

[50] J. Hromkovic, *Algorithmics for Hard Problems - Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics, Second Edition*. Springer, 2004.

[51] C. M. Li and Z. Quan, "An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem," in *Proc. AAAI*, 2010.

[52] P. S. Segundo, F. Furini, and J. Artieda, "A new branch-and-bound algorithm for the maximum weighted clique problem," *Comput. Oper. Res.*, vol. 110, pp. 18–33, 2019.

[53] H. Jiang, C. Li, and F. Manyà, "An exact algorithm for the maximum weight clique problem in large graphs," in *Proc. AAAI*, 2017, pp. 830–838.

[54] H. Jiang, C. Li, Y. Liu, and F. Manyà, "A two-stage MaxSAT reasoning approach for the maximum weight clique problem," in *Proc. AAAI*, 2018, pp. 1338–1346.

[55] Y. Wu, M. Tornatore, S. Ferdousi, and B. Mukherjee, "Green data center placement in optical cloud networks," *IEEE Trans. Green Commun. Netw.*, vol. 1, no. 3, pp. 347–357, 2017.

[56] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "VNF placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, p. 69, 2019.

[57] I. Jang, D. Suh, S. Pack, and G. Dán, "Joint optimization of service function placement and flow distribution for service function chaining," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2532–2541, 2017.

[58] Z. Zhou, Q. Wu, and X. Chen, "Online orchestration of cross-edge service function chaining for cost-efficient edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1866–1880, 2019.