mathematics

MDPI

Article

Reinforcement Learning-Based Delay-Aware Path Exploration of Parallelized Service Function Chains

Zhongwei Huang [1], Dagang Li [1,2,*], Chenhao Wu [3] and Hua Lu [4]

[1] School of Computer Science and Engineering, International Institute of Next Generation Internet, Macau University of Science and Technology, Taipa 999078, Macao
[2] Zhuhai-M.U.S.T. Science and Technology Research Institute, Zhuhai 519031, China
[3] Faculty of Innovation Engineering, Macau University of Science and Technology, Taipa 999078, Macao
[4] Guangdong Communication & Network Institute, Guangzhou 510000, China
* Correspondence: dagang.li@ieee.org

Abstract: The parallel processing of the service function chain (SFC) is expected to provide better low-delay service delivery, because it breaks through the bottleneck of traditional serial processing mode in which service delay increases linearly with the SFC length. However, the provision of parallelized SFC (PSFC) is much more difficult due to the unique construction of PSFCs, inevitable parallelization overhead, and delay balancing requirement of PSFC branches; therefore, existing mechanisms for serial SFC cannot be directly applied to PSFC. After a comprehensive review of recent related work, we find that traffic scheduling mechanisms for PSFCs is still lacking. In this paper, a delay-aware traffic scheduling mechanism (DASM) for PSFCs is proposed. DASM first transforms PSFC into several serial SFCs by releasing the upstream VNF constraints so as to handle them independently while keeping their parallel relations. Secondly, DASM realizes delay-aware PSFC traffic scheduling based on the reinforcement learning (RL) method. To the best knowledge of the authors, this is the first attempt to address the PSFC traffic scheduling problem by transforming them into independent serial SFCs. Simulation results show that the proposed DASM outperforms the advanced PSFCs scheduling strategies in terms of delay balance and throughput.

Keywords: machine learning; reinforcement learning; parallelized service function chains; delay-aware traffic scheduling

MSC: 68T01

check for updates

Citation: Huang, Z.; Li, D.; Wu, C.; Lu, H. Reinforcement Learning-Based Delay-Aware Path Exploration of Parallelized Service Function Chains. *Mathematics* **2022**, *10*, 4698. https://doi.org/10.3390/math10244698

Academic Editor: Liangxiao Jiang

Received: 8 November 2022
Accepted: 8 December 2022
Published: 11 December 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Network function virtualization (NFV) technology separates network functions (NF) from proprietary hardware and moves data processing from proprietary hardware to virtual network functions (VNFs) running on general-purpose servers, enabling flexible function deployment and rapid network service upgrade. NFV effectively increases the flexibility of the operator's network and significantly lowers operating expenses and capital expenditures. VNFs are chained in a predefined order to provide services in the form of service function chain (SFC), such as VPN–traffic monitor (TM)–firewall (FW)–load balance (LB) [1]. These VNFs are the virtual representation of network functions (NFs) and need to be mapped to the underlying physical servers for execution. A software-defined network (SDN) controller with a global view is needed to guide traffic to traverse each VNF in the right order.

Traditionally, SFCs are constructed and processed in a serial way: each VNF is deployed, and traffic is routed through the VNFs in a predefined order, as shown in Figure 1. The end-to-end delay increases linearly with SFC length, which may become unacceptable for time-sensitive applications. For example, for end-to-end 5G services in a URLLC scenario, the delay for this type of services should be less than 5 ms or even less [2]. If the

delay constraint is exceeded, the service will fail, and the user experience will be impacted. In order to cope with this problem, previous research [1,3] has proposed the idea of SFC parallel processing or *parallelized SFCs (PSFCs)*, by parallelization the VNFs without dependency, so the end-to-end SFC delay can be significantly reduced, in some cases by more than 30% [1]. As demonstrated in Figure 1, the end-to-end SFC delay is reduced by 38 ms after parallelizing TM and FW.

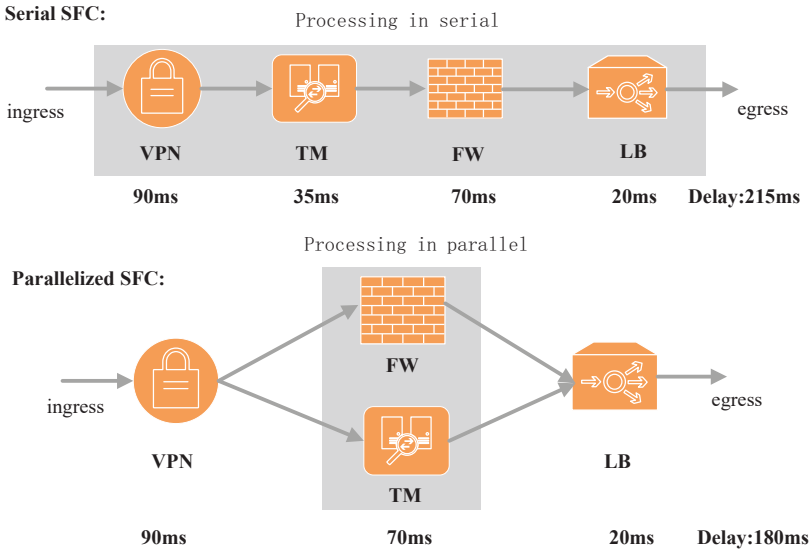

Figure 1. Serial and parallel SFC processing comparison adapted from [4]. The number below each VNF denotes the processing time of that VNF, which is derived from the tests conducted in [1]. Traffic arrives at the source node, passes through each VNF, and finally reaches the destination node.

A lot of research efforts including our previous works [4,5] have developed architectures and techniques to facilitate the parallel processing of SFC. The provision of PSFC differs from that of standard serial SFC, and research on the parallel processing mechanism of SFC is still in its early stages, there are still some open issues:

- Current PSFC processing methods are based on a complex parallel architecture, which has high VNF dependence between parallel branches (e.g., two branches of PSFC in Figure 1 have interdependent VNFs of VPN and LB), resulting in complicated processing and limited optimization space for performance improvement.
- The end-to-end delay of PSFC is determined by the maximum branch. Delay difference caused by uncoordinated VNF processing between parallel branches of PSFC may cause packet accumulation in the subsequent node, which degrades the performance of PSFC.
- Existing parallel processing methods (e.g., reduction of the unreasonable parallelism of PSFC [4]) only consider VNF processing delay when balancing the parallel branches, but link delay can also greatly affect the total delay in the subsequent traffic scheduling process and should be also be considered.

These problems limit the applicability of PSFCs. Therefore, this paper aims to achieve a better parallel architecture to convert complex PSFCs into simplified forms so that they can be handled together with traditional serial SFCs in a unified way. We further propose a delay-aware traffic scheduling method to support delay balanced PSFCs traffic scheduling that considers all the major delay factors. Overall, the contributions of this paper are listed as follows:

- A comprehensive review of PSFC research in recent years is presented, as shown in Figure 2. We find that a traffic scheduling mechanism of PSFCs is still missing; therefore, a delay-aware traffic scheduling mechanism (DASM) for PSFC is proposed to fill this gap;

- DASM transforms PSFCs into multiple serial SFCs with reduced VNF dependency. This process relaxes half of the dependencies of parallel structures and reduces the PSFC's processing difficulties;
- To further maximize the benefits of PSFCs, a reinforcement learning (RL)-based PSFC scheduling algorithm is proposed, which can effectively minimize the total service delay by keeping the delay between parallel branches balanced.

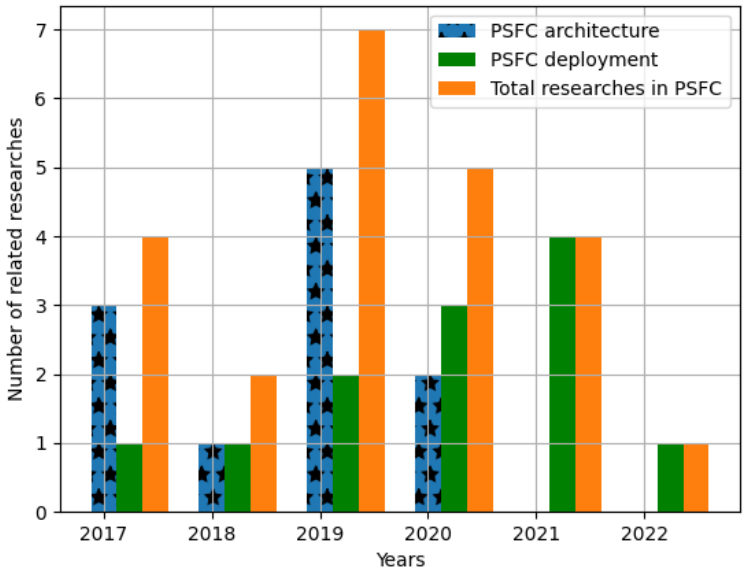

Figure 2. Statistics of studies on parallelized SFCs.

The rest of the paper is organized as follows. Section 2 gives a comprehensive overview of the research on PSFC, which is divided into the deployment and scheduling aspects. Section 3 introduces the serialization methodology of our scheduling mechanism, as well as the mathematical description. Section 4 describes the details of our Q-learning-based mechanism, we verified our proposed scheme by simulation in Section 5, and the full text is discussed and summarized in Section 6.

2. Systematic Review of Related Works

To date, many researchers have conducted extensive research on the deployment and scheduling of serial SFC. For example, Santos et al. [6] explored the deployment of SFC in large-scale network situations and suggested an RL technique that takes into consideration the availability of SFC, operational expenses, and energy usage. However, they still based this on the traditional serial SFC architecture. The instances can be deployed in parallel or distributively because the NFs in an NF chain are seen as logical nodes rather than fixed nodes. The problem of topology design is brought up by the fact that different NF chain topologies can express the same service request. In order to dynamically assign processing and transmission resources, Alhussein et al. [7] considered joint topology design, traffic routing, and NF placement for unicast NFV-enabled services. It is not easy to determine the appropriate resource usage for completing user requests in cloud or edge environments; hence, an effective resource prediction model might be crucial in resource management to accurately estimate the required resources. Tofighy et al. [8] proposed an ensemble CPU load prediction model that uses a Bayesian information criterion to select the best constituent model for each time slot based on the past consumption of cloud resources. More details in enabling scalable, resilient, and high-performance NFV/SFC deployments in next-generation networks can be found in a recent survey [9], in which Adoga et al. provided a thorough analysis of contemporary NFV and serial SFC implementation frameworks, covering topics including resource allocation and service orchestration, performance

optimization, resilience, and failure recovery. They also identified the main open research issues in present SFC research.

Different from the existing research, this paper focuses on the traffic scheduling of parallelized SFCs, which has not been well studied in previous works. We will summarize the related works in terms of SFC parallelized architecture and parallelized SFC deployment.

2.1. SFC Parallelized Architecture

A systematic review of SFC parallelized methods and their proposed architecture, evaluation tools, utilized techniques, advantages, and disadvantages is summarized in Table 1.

Zhang et al. [3] first proposed a serial and parallel hybrid SFC structure called ParaBox for processing packets between VNFs in parallel. The core functions of ParaBox include a dependency analysis model to determine whether NFs can be parallelized, a copy function to distribute the copies of data packets to multiple parallel branches, and a merge function to combine the output results. The architecture can effectively realize parallel processing of NF and provides an effective research basis for the parallel processing mechanism of SFC. Sun et al. [1] further studied the operation types of commonly used network functions (NFs) and designed an NF parallel processing mechanism based on these operation types, namely NFP. NFP realizes parallel processing of NFs without affecting the correctness of SFC. Similarly, Baek et al. [10] also parallelized NFs based on the dependency analysis of NFs as well.

Although SFC can be processed in parallel, existing parallelized SFC works focus on the strict execution order of VNFs in SFC. Ayoubi et al. [11] studied flexible SFC orchestration, and they reorganized the order of VNFs in SFC to better allocate computing and network resources for SFC provision. Chowdhary et al. [12] focused on the implementation of parallel SFC and introduced an NF parallelism for open-flow networks. They exploit the NF parallelism on top of open-flow rules to improve the performance of SFC in cloud networks. Liu et al. [2] proposed a new two-stage NF parallelization processing framework named TNP. They divided the deployment problem of PSFC into two sub-problems to address the SFC parallelization graph design and NFs mapping problems, respectively. Wang et al. [13] proposed the ParaNF architecture. They first proposed the delay-balanced SFC parallelism optimization problem to eliminate unnecessary parallelism. However, they only considered the VNF processing delay, which makes it unsuitable for real deployment situations. To reduce the redundant processing of modular NFs, Jiang et al. [14] proposed a new NF framework named SpeedyBox, which innovatively utilizes cross-NF runtime integration to improve the performance of PSFCs. A reliability evaluation approach and reliability optimization algorithm of SFC were proposed by Rui et al. [15]. Resource preemption, common-cause failure, failure recovery, and redundancy backup are some of the elements that are examined in regard to the composition relationship and reliability influencing factors of serial and parallelized SFCs.

Table 1. A systematic review of parallelized SFC architecture and methods.

Research	Years	Proposed Architecture	Performance Metrics	Evaluation Tools	Utilized Techniques	Advantages	Disadvantages
Zhang et al. [3]	2017	Parabox	delay, throughput	DPDK-enabled BESS	NF operations on packet headers and payload	the first study to explore parallel packet processing to reduce SFC latency	its NF parallelism detection remains preliminary and lacks a comprehensive analysis on NF action dependency, introduce larger resource overhead
Sun et al. [1]	2017	NFP	delay, load balance	DPDK in Linux containers	NF parallelism identification	light-weight packet copying, distributed parallel packet delivery, and load balanced packet merging	treats all CPU cores as equal, and allocates isolated CPU cores to different NFs
Hu et al. [16]	2018	NFcompass	overhead, load balance, throughput	DPDK, container	SFC re-organization, graph partition-based task scheduling	reduces aggregated processing overheads and coexistence interference overheads	uses a limited workload set that is not enough to represent the whole range of NFV domains
Engelmann et al. [17]	2018	-	reliability, resource efficiency	Monte Carlo simulations	backup VNFs	reliability up to 0.99999	considers the selection of multiple paths to carry the traffic flow of one SFC
Engelmann et al. [18]	2019	-	reliability, service success, and overhead	Monte Carlo simulations	erasure coding	provides higher service success while requiring less reliable path segments than backup protection	splitting large traffic into multiple sub-flows increases the network node occupation
Ayoubi et al. [11]	2018	Khaleesi	acceptance rate; revenue	CPLEX solver, FNSS tool	relaxed VNF orderings	studies the potential advantages of flexible structures over the rigid ones	ILP-based models are faced with high solution times
Chowdhary et al. [12]	2019	SFC-NFP	delay; throughput	DPDK platform with an OpenFlow switch	M/M/c queue model	the solution can seamlessly be integrated within an infrastructure using SDN	it requires the use of an OpenFlow switch, which may not be available in all networks
Liu et al. [2]	2019	TNP	delay; throughput	unspecified	heuristic algorithm	better utilizes the computing and link bandwidth resources in the substrate network	reliability is not taken into consideration

Table 1. *Cont.*

Research	Years	Proposed Architecture	Performance Metrics	Evaluation Tools	Utilized Techniques	Advantages	Disadvantages
Wang et al. [13]	2019	ParaNF	delay; balance	KVM VMs pktgen-dpdk	heuristic algorithm	ParaNF can reduce latency by up to 47% over traditional SFC while maintaining the nearly line-speed packet processing performance	new branches with a processing delay slightly larger than the maximum processing delay of the other branches were not allowed
Jiang et al. [14]	2019	Speedybox	delay; eliminate redundancy	table-based Match/ Action technique	cross-NF runtime consolidation	low-latency NFV framework eliminates the redundancy without sacrificing modularization	relies on dedicated hardware applications and requires strict update delay
Zhang et al. [19]	2019	HybridSFC	delay throughput multi-server	BESS Pktgen-dpdk KVM VMs	heuristic algorithm	spans multiple machines to achieve high performance and programmability to improve SFC performance with manageable overheads	the modeling and the specification of parallel SFC in the controller are not studied
Xie et al. [20]	2020	FlexChain	delay accept flow	VL2 Fat-Tree BCube	heuristic algorithm	parallel VNFs are placed on the same node, thus preventing the loss in copy/merge latency on different nodes	may not be applicable in resource-constrained scenarios

Earlier SFC parallel processing architectures such as ParaBox and NFP mainly consider the scenario where the entire SFC is deployed on a single server, which greatly reduces the flexibility of the system. Zhang et al. [19] proposed a parallelization framework, i.e., HybridSFC, to accelerate the operation of SFC across multi-core servers. To better investigate traffic-level parallelism, HybirdSFC splits a sequential SFC into many SFC pieces (i.e., sub-SFC is applied to a part of the traffic flow) and parallelizes NF processing across multiple cores/servers to achieve NF-level parallelism. Similarly, Xie et al. [20] proposed a more adaptable and effective SFC parallel system, called FlexChain, which enables the placement of SFCs on many servers, obviating the need for additional bandwidth between servers. Due to the NP-hard nature of the problem, they propose a parallel-aware approximate deployment algorithm with performance guarantees, as well as an efficient heuristic algorithm especially suitable for large-scale data center networks.

2.2. Parallelized SFC Deployment

Both serial and parallelized SFC need to be mapped to the underlying network nodes or servers to provide the resources for processing. This process becomes an SFC deployment problem. The deployment problem of SFC is an NP-hard problem. The additional packet replication and resource consumption in parallelized SFC further complicates the problem. A systematic review of parallelized SFC deployment studies, their contributions, performance metrics, evaluation tools, utilized techniques, advantages, and disadvantages is summarized in Table 2. We divided the related works into three categories according to their optimization goals.

Table 2. A systematic review of PSFC deployment and scheduling methods.

Research	Years	Contribution	Performance Metrics	Evaluation Tools	Utilized Techniques	Advantages	Disadvantages
Sun et al. [21]	2019	request splitting; SFC deployment in DCNS	delay; load balance; acceptance rate	unspecified	heuristic algorithm	alleviates the problem of hash collision of elephant flows	splitting large traffic into multiple sub-flows increases the network node occupation
Bao et al. [22]	2020	VNF placement of PSFCs	delay; overhead; load balance	OpenStack, ONOS, and XOS	Prune and Plant (P&P)	eliminates the NP-hardness	two-stage optimization may lead to coordination difficult
Cai et al. [5]	2020	VNF deployment of PSFCs	delay; resource utilization; acceptance rate	Matlab	match theory	a match theory-based method leads to high resource efficiency	multiple algorithms result in higher algorithmic complexity
Luo et al. [23]	2020	VNF deployment of PSFCs	delay; acceptance rate	C/C++	ILP; heuristic algorithm	use the heuristic algorithm to search for a near-optimal solution	the computationally expensive DP-based approach may not be practical for large-scale networks
Rui et al. [15]	2020	VNF migration of PSFCs	VNF migration; reliability	OpenStack	Petri net model	VNF migration reduces the impact of resource preemption and common-cause failures on service reliability	the difficulty increases when solving high-order systems
Cai et al. [4]	2021	VNF deployment; scheduling of PSFCs	delay; parallelism degree; resource utilization, acceptance rate	Matlab	reinforcement learning	ML-based method enables an automatic decision-making	facing the problem of parameters adjustment and training
Wang et al. [24]	2021	VNF Placement & sub-SFC backup	delay; availability; resource utilization	BESS, DPDK, and Alevin	heuristic algorithm	guarantees a high availability (99.999%)	the numbers of working and backup sub-SFCs are fixed
Lin et al. [25]	2021	NF instance assignment of PSFCs	delay; parallelism degree	C++	heuristic algorithm	investigated the optimal partial parallelism problem	did not involve the partial parallel chain embedding

Table 2. *Cont.*

Research	Years	Contribution	Performance Metrics	Evaluation Tools	Utilized Techniques	Advantages	Disadvantages
Lin and Chou et al. [26]	2021	dependency-aware NF instance assignment	delay; load balance	unspecified	scoring-based mechanism	avoids detouring any parallel segment and further balances the loading of instances	only ensures that parallel subpaths are of roughly the same length
Zheng et al. [27]	2022	parallelism-aware VNF embedding	delay; resource utilization	JAVA	two-stage heuristic algorithm	near-optimal performance when computing resources are enough	still focuses on service function chaining embedding problem
Our proposed method	to date	converts PSFs to multiple serial SFCs; PSFCs scheduling	delay; delay balance; resource utilization; acceptance rate	Pycharm, python	reinforcement learning	the proposed method searches for a unified solution for parallelized SFCs	faces the problem of parameters adjustment and training

2.2.1. Studies Aiming to Reduce Service Delay

Baek et al. [10] proposed a PSFC deployment mechanism in a distributed network in which VNFs in an SFC are deployed on the same node as much as possible to reduce link delay. In order to solve the problem of long queuing delay caused by elephant flow and mouse flow on the same path, Sun et al. [21] first split the large flow into multiple sub-flows and copied the original SFC into several sub-SFCs, i.e., parallelized flow. They propose a heuristic algorithm for online PSFC orchestration, which determines the VNFs and virtual link mapping to servers and corresponding physical links. Bao et al. [22] proposed a two-stage Prune and Plant (P&P) method to realize the optimized parallelization and deployment of PSFC. First, the dependencies of VNFs were described by a directed acyclic graph (DAG). In the pruning stage, the DAG was transformed into a serial-parallel graph (SP graph), and the NP-hardness is eliminated while maintaining the parallelism of VNFs. In the plant phase, the optimal deployment position of VNFs in the SP-graph is found to reduce the total latency. Cai et al. [5] proposed an architecture to support SFC parallel processing and designed an SFC parallel processing algorithm to parallelize two NFs which can effectively reduce the service delay by more than 30%. Furthermore, a service chain deployment algorithm based on optimal matching theory was proposed to form the optimal matching of VNFs and nodes, which can effectively improve the utilization of node resources. Luo et al. [23] also studied the deployment of PSFCs. First, they designed an SFC parallel algorithm based on VNF dependencies to convert the original SFC into a parallelized SFC. Then, the deployment problem was modeled as an integer linear programming problem and a heuristic method ParaSFC was proposed to obtain an approximate optimal solution. According to the experimental findings, ParaSFC may accommodate more SFC deployment requests on networks with limited resources while parallelizing all installed p-SFCs to minimize their average service latency by roughly 15%.

2.2.2. Studies Aiming to Reduce the Delay Difference and Parallelism Optimization

Although the parallel processing of the SFCs can reduce the service delay to a certain extent, the Uncoordinated processing speed of sub-SFCs will bring non-negligible delay differences and packets accumulation between parallel branches, furthermore, the end-to-end delay of PSFCs is determined by the longest branch. To this end, our previous work [4] studied the parallelism optimization problem of PSFCs and proposed a parallelism optimization algorithm, i.e., POA. POA can effectively reduce unreasonable parallelism. They further studied the optimal scheduling problem of multiple PSFCs and proposed a Q-learning-based PSFC scheduling method. Compared with traditional heuristic algorithms, it can effectively reduce service delay by more than 30% and improve the resource utilization rate by more than 25%. The parallel processing of network functions will bring non-negligible resource consumption and overhead, especially in the case of distributed deployment of NFs. Lin et al. [25] studied the optimal parallel processing of distributed NFs under different SFC situations (i.e., long function time, long transmission time, long processing time), and proposed a partially parallel SFC processing algorithm, i.e., PPC. In this, NFs are processed in parallel only when PPC can provide latency reduction. The experimental results show that the algorithm has improved performance compared with SFC serial processing and SFC fully parallel processing, and significantly reduces service latency by more than 35%. Lin et al. [26] studied the optimal embedding of PSFC and took the VNF dependency into consideration. They proposed a VNF embedding method based on the scoring principle. The scores of VNFs include the order score of functions and the score of dependencies between NFs. Zheng et al. [27] introduced a new augmented graph method to address the parallel relationship constraints between SFs and proposed the parallelism-aware SFC embedding problem. To address this problem, they proposed two approximately optimal SFC deployment algorithms for when resources are either sufficient or limited, which jointly optimize processing and propagation delay.

2.2.3. Studies Aiming to Ensure Service Reliability or Availability

Engelmann et al. [17,18] studied the problem of end-to-end service reliability in parallel service flows. They divided larger flows into smaller sub-flows, and to avoid hash collision issues in the same link, they analyzed the number of backup VNFs required by parallelized SFCs to achieve the required reliability in both a centralized and distributed manner. Wang et al. [24], studied the problem of parallelized SFC deployment in data center networks and considered the availability guarantees and resource optimization of SFCs. They separate an SFC into numerous sub-SFCs and offer a backup sub-SFC model to assure the right processing of services when errors occur instead of attempting to parallelize NF in an SFC. They also created three placement methods and a hybrid placement methodology (HPA). The proposed method reduced 40% of the consumption and reduced 30% of the SFC latency while maintaining a high availability (99.999%), according to their simulation.

It can be seen that there is a relatively mature body of research on the deployment of parallelized SFCs, reducing service delay and delay difference, parallelism optimization, and ensuring service reliability or availability by SFCs parallelization. However, only Cai et al. [4] have studied the scheduling of parallelized SFCs, but their proposed Q-learning-based scheduling will face the exponential explosion problem in large-scale scenarios.

3. Proposed Method for PSFC Serialization

3.1. Representation of PSFCs

In this study, a directed acyclic graph (DAG) is used to model a PSFC. The directed edges in the DAG reflect the packet forwarding direction between two VNFs, and the nodes in the DAG represent the VNFs of an SFC. A serial SFC graph can be converted into a PSFC graph (PSG) in accordance with the relationship between VNFs [4]. Figure 3a displays possible PSG combinations. The SFC VNFs can either be fully parallelized using several VNFs or concatenated into a serial SFC graph. PSG can also be organized with multiple balanced and unbalanced VNFs in different branches. The packets need to be replicated and sent to multiple parallel branches for parallel processing, which is performed by the copy and merge node as shown in Figure 3a. The total number of parallel branches that need a duplicate of the packet [4] is defined as the parallelism degree of PSG, indicated by Φ. For example, the parallelism degree of the configurations presented Figure 3a are 1, 3, 2, and 2, respectively.

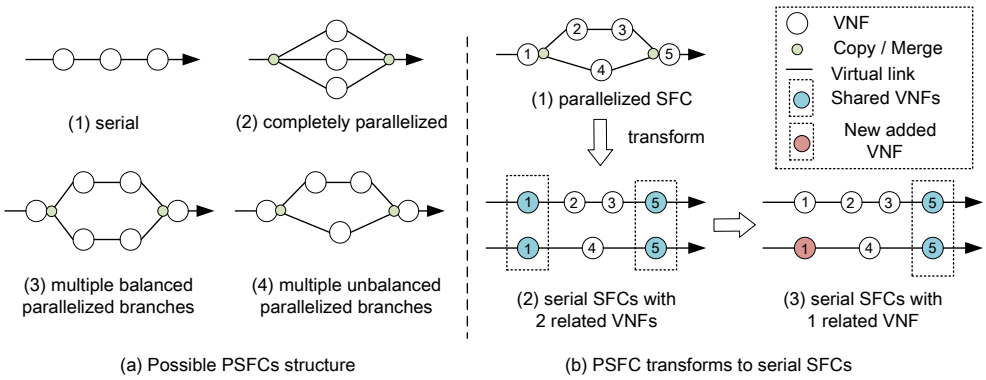

Figure 3. Possible PSG structures and PSFC transformation.

3.2. Transforming PSFC into Multiple SFCs

In this paper, we jointly consider PSFCs as well as serial SFCs in our system in order to find a unified solution. PSFCs are transformed into multiple parallel branches according to the degree of parallelism. As shown in Figure 3b, the PSFC is transformed into two branches. Different from normal independent SFCs, the branches transformed from a PSFC have VNF dependencies between them. For example, for non-parallelized VNF1 and VNF5 of the two parallel branches, they use a shared VNF for traffic processing. Data replication and merging operations are also required in addition to simple traffic routing, that is, the

traffic must be copied (performed by copy function) and sent to multiple branches after being processed by VNF1, and the traffic of different branches must be merged before sent to the next non-parallelized VNF, i.e., VNF5.

Reducing the related VNFs: There are two ways to deal with these two branches. In the first case, both VNF1 and VNF5 can be deployed in the same server and use the shared VNF (indicated by blue circles) to process the traffic, as shown in Figure 3b(2). In this case, no additional VNF instance in the physical server will be consumed in comparison to the original PSFC, but these two branches having two related VNF dependencies makes the scheduling problem more complicated and difficult to coordinate. It is worth mentioning that it is also possible to utilize the VNF in other servers for non-parallelized VNFs, although using the newly instantiated VNF instead of sharing the VNF will inevitably bring VNF instantiation cost and additional server running cost (if it were originally in an off state). We noticed that because there are no upstream VNFs before VNF1, it is feasible to deploy VNF1 of the two branches on different servers and sent the independent traffic to downstream VNFs. On the contrary, there are three upstream VNFs (i.e., VNF2, VNF3, and VNF4) before VNF5, so the arriving traffic must be merged before VNF5 to guarantee the correctness of the traffic. Because of this reason, VNF5 of the two SFCs must be deployed in the same server and use the shared instantiated VNF. Overall, the deployment of the two SFCs will consume one additional instantiated VNF (i.e., VNF1 as indicated by the red circle) in a physical server.

As shown in Figure 3b(3), although the additional instantiation cost of one VNF is introduced, the problem is much simplified compared to the first case by reducing the number of related VNFs from two to one. Moreover, these two parallel branches can be scheduled separately and regarded as two SFCs with the same destination, i.e., the last VNF of two SFCs must be scheduled to the same server.

3.3. Traffic Scheduling of PSFCs

For the ease of the following discussion, a schematic diagram of PSFC traffic scheduling is shown in Figure 4, showing a PSFC consisting of VNF1, VNF2, VNF3, VNF4, and VNF5, and the processing delay is shown below each VNF. It is assumed that VNF2, VNF3, and VNF4 can be processed in parallel, based on any of the previous parallelized analysis methods [1,3]. Because VNF processing delays are different, in order to achieve delay balance, VNF2 and VNF4 are in the same parallel branch and VNF3 is in another branch. The traffic enters from the ingress, passes through each parallel branch in turn, traverses each VNF, and then reaches the service destination node at the egress. The PSFC can be transformed into two branches, including branch 1: VNF1-VNF2-VNF4-VNF5 and branch 2: VNF1-VNF3-VNF5. The underlying network topology consists of several servers and switches, as well as links between them. The instantiated VNF types that each server can carry are shown in the figure. We present two examples to elaborate the two traffic scheduling methods discussed above, where white and orange represent the idle and occupied VNF, respectively, as shown in Figure 4.

Traffic scheduling scheme 1: as shown in Figure 4a, both VNF1 and VNF5 use a shared VNF deployed in the same server. Specifically, VNF1 is deployed in S1, VNF2 is deployed to S2, VNF3 is deployed to S4, VNF4 is deployed to S5, and VNF5 is deployed to S7. The numbers between the nodes represent the link delay. The traffic path of serial branch 1 is represented by a blue dotted line, and the traffic path of serial branch 2 is represented by a red dotted line. At this time, the total end-to-end delay of branch 1 is the sum of VNF processing delay and link delay, which is $\underbrace{20+15+18+15}_{processing\ delay}+\underbrace{15+15+25+20+10}_{link\ delay}$ = 153 ms. Similarly, the end-to-end delay of branch 2 is 155 ms.

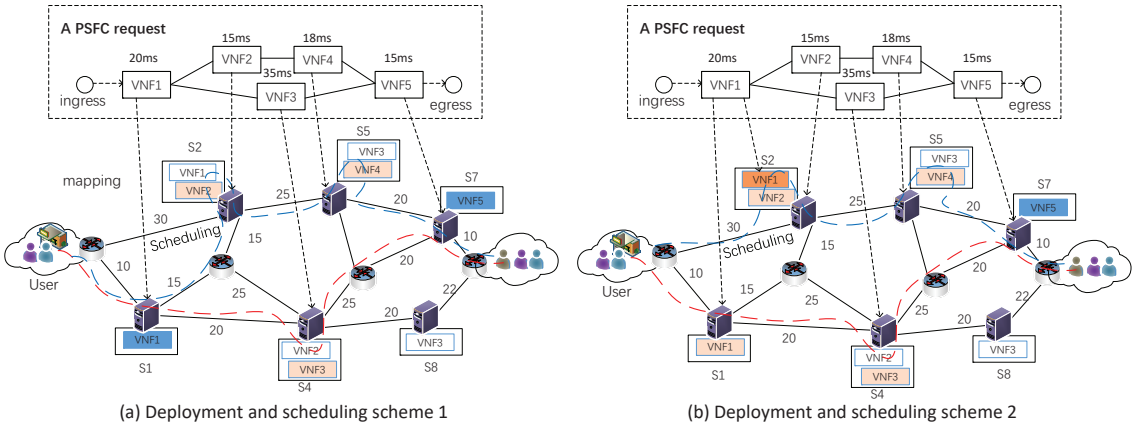

Figure 4. Traffic scheduling of PSFC. The white, blue, orange, and red boxes represent the idle VNFs, shared VNFs, occupied VNFs, and additional VNFs, respectively.

Traffic scheduling scheme 2: as shown in Figure 4b, now the two parallel branches are processing independently. VNF1 is deployed in both in S1 and S2, VNF2 is deployed to S2, VNF3 is deployed to S4, VNF4 is deployed to S5, and only VNF5 deployed to S7 utilizes the shared VNF to merge the parallelized flow from the two branches. The end-to-end delay of PSFC in this case can be obtained similarly, where branch 1 and branch 2 have end-to-end delays of 153 ms and 155 ms, respectively.

Although these two scheduling strategies result in the same end-to-end delay of PSFC which is determined by the maximum delay of two parallel branches (i.e., 155 ms), the first scheme is more difficult to implement with two related VNF among the two branches, whereas scheme 2 transforms the PSFC into two independent branches, which is much easier to implement, and the system can support massive service both in parallelized and serial SFCs. However, it can not be ignored that scheme 2 uses one additional VNF (i.e., VNF1) to support the independent processing of the two branches, which will introduce the additional VNF instantiation cost and running cost of server.

Minimizing the delay difference of parallel branches: Although the service end-to-end delay depends on the longest branch in PSFCs, the delay difference between two parallel branches is also crucial. Large delay difference will introduce queuing delay and packet accumulation at the merge node [4]. The delay difference of two examples above is 2 ms which is relatively small and does not affect the processing efficiency of the PSFC. Note that, in addition to S4, VNF3 can also be executed on S5 and S8. The total end-to-end delay of branch 2 when VNF3 is executed on S5 and S8 will be $\underbrace{20+15+18+15}_{processing\ delay}+\underbrace{10+15+15+25+20+10}_{link\ delay}=165$ ms and $\underbrace{20+15+18+15}_{processing\ delay}+\underbrace{10+20+20+22}_{link\ delay}=142$ ms, respectively, and the delay difference comparing to branch 1 (which is 153 ms) becomes 12 ms and 11 ms, which is much larger. The current strategy is obviously optimal. The delay difference of PSFC under different scheduling strategies and their total delay are summarized in Table 3. It can be seen from the table that different traffic scheduling strategies have great differences in service delay and optimizing the traffic scheduling strategy of SFC is particularly important for the service quality.

Table 3. Performance of different scheduling policies, where o_i represents the flow ingress and d_i represents the flow egress node.

Policies	Branch 1 Delay	Branch 2 Delay	Differences	e2e Delay	If Optimal
o_i-S1-S4-S7-d_i	153 ms	155 ms	2 ms	155 ms	yes
o_i-S1-S4-S5-S7-d_i	153 ms	165 ms	12 ms	165 ms	no
o_i-S1-S4-S8-d_i	153 ms	142 ms	11 ms	153 ms	no

3.4. Real-World Case Study

In this subsection, we provide a real-world case study top obtain a better understanding of our proposed method, as shown in Figure 5a. Take the real-world service of Figure 1 as an example, the number of each VNF can be expressed as the processing time of VNF, which is derived from the real tests [1]. Because TM and FW are independent NF, the original serial SFC is converted into parallel service graph to improve the service processing speed. In the parallelized service graph, TM and FW process in parallel, receive the traffic from VPN, and then send the traffic to the LB.

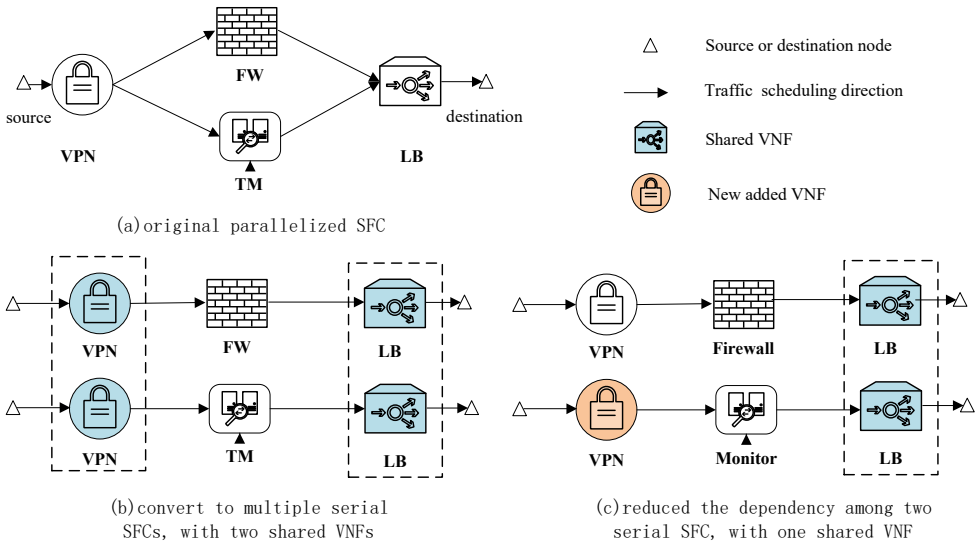

Figure 5. Case study of real-world parallelized SFC.

In our proposed approach, the parallelized SFC will first be converted into two parallel branches (-VPN-FW-LB- and -VPN-TM-LB- shown in Figure 5b) where VPN and LB can use the shared VNF. These two branches still have two interdependent VNFs. According to the analyses in Section 3.2, to reduce the dependency of the two serial SFC, we can relax the VPN dependency by adding a new VPN, so the PSFC will be converted to -VPN-TM-LB- and -VPN-FW-LB-, where only the LB uses the shared VNF. In this case, these two branches only have one interdependent VNFs, as shown in Figure 5c.

In the original parallelized graph, in order to maintain delay balance, the delay of the two parallel branches is only determined by one VNF and two links, e.g., the processing delay of FW and its connected two links, whereas in Figure 5c, the delay of two parallel branches are determined by two VNFs and three links. More space of delay optimization and traffic scheduling make it easier to balance delays during actual traffic scheduling process.

In the following section, we will introduce in detail how to implement the scheduling of the PSFCs based on the RL method and ensure the delay balance between the parallel branches.

3.5. Mathematical Formulation

Assuming that the set of SFC requests is S, $s_i \in S$ indicates the ith SFC request, the jth VNF of s_i is presented by $f_{ij} \in F$. Define a binary variable $x_{ij}^k = \{0,1\}$, with 1 indicating the jth VNF of s_i is processed by the server k, and 0 otherwise. Another variable $y_{j,j+1}^{k,k'} = \{0,1\}$ is also introduced, where 1 indicates the traffic flow from jth VNF instance on server k will be sent to $j+1$th instance on server k', and 0 otherwise.

In the traditional serial SFCs, the total end-to-end delay of SFCs is the sum of VNF processing delay, transmission delay, and link propagation delay:

$$D_{e2e} = \sum(\phi_{ij} + \tfrac{v_i}{b_l})x_{ij}^k + \sum D_{l(k,k')}y_{j,j+1}^{k,k'}, i \in S, j, j+1 \in F, k, k' \in N, \tag{1}$$

where ϕ_{ij} indicates the processing delay of jth VNF of s_i, $D_{l(k,k')}$ represents the link delay between server k and k', v_i is the data size of s_i, b_l is link transmissibility, which is often defined by link bandwidth of link l connected server k and k'; therefore, $\frac{v_i}{b_l}$ represents the data transmission delay.

Theorem 1. *Assuming there are p parallel branches in total in a PSFC. The service end-to-end delay D_{e2e} of the PSFC is determined by the parallel branch p^* with the maximum delay, called the critical branch with critical delay D_{e2e}^c, with the delay of other branches being D_{e2e}^p and $D_{e2e}^p \leq D_{e2e}^c$.*

Based on Theorem 1, the end-to-end delay of a PSFC is defined as:

$$\begin{aligned} &D_{e2e} = D_{e2e}^c = \max_p\{\sum_{j\in p}(\phi_{ij} + \tfrac{v_i}{b_l})x_{ij}^k + \sum_{j,j+1\in p} D_{l(k,k')}y_{j,j+1}^{k,k'}\}, \\ &i \in S, j, j+1 \in F, k, k' \in N. \end{aligned} \tag{2}$$

In traditional serial SFCs, the sole objective is to minimize to total end-to-end delay by efficient traffic routing strategy, whereas in PSFCs, the delay mismatch between parallel branches could bring packet accumulation due to the different processing speed, resulting in performance degradation [4]. Furthermore, resources of non-critical parallel branches that do not determine the service end-to-end delay can be allocated to other services, which may improve the resource utilization and service throughput. Thus, minimizing the delay difference between parallel branches with reasonable traffic scheduling is also crucial in PSFCs.

The delay difference D_{e2e}^f of a PSFC is defined as the total difference between its non-critical parallel branches ($p \neq p^*$) and its critical parallel branch ($p = p^*$):

$$\begin{aligned} &D_{e2e}^f = \sum_{p\neq p^*}(D_{e2e}^c - D_{e2e}^p) \\ &= \sum_{p\neq p^*}(D_{e2e}^c - (\sum_{j\in p}(\phi_{ij} + \tfrac{v_i}{b_l})x_{ij}^k + \sum_{j,j+1\in p} D_{l(k,k')}y_{j,j+1}^{k,k'})), \\ &i \in S, j, j+1 \in F, k, k' \in N. \end{aligned} \tag{3}$$

In this paper, the optimization objective is to jointly minimize the end-to-end delay of PSFCs as well as minimize the delay difference between parallel branches of PSFCs using a reasonable scheduling action a_t, which will be defined in the next section:

$$\mathcal{O} = \min_{\{a_t\}} \sum_{i\in S}(\alpha D_{e2e} + \beta D_{e2e}^f), \tag{4}$$

where α and β are the coefficients of the two optimization objectives.

In order to ensure that the resources meet the existing network conditions when processing service requests, the constraints are represented by Equations (5) through (7).

$$\sum(x_{ij}^k + \xi_{ij}^k) = 1, \forall i \in S, j \in F, k, k' \in N, \tag{5}$$

$$\tau_{ij-1} + \sum \frac{v_i}{b_l} x_{ij-1}^k + \sum_{uv \in E} D_{l(k,k')} y_{j-1,j}^{k,k'} \leq T_{ij}, \quad \forall i \in S, \forall j-1, j \in F, \forall k, k' \in N. \tag{6}$$

$$T_{ij} + \sum \phi_{ij} x_{ij}^k \leq \tau_{ij}, \forall i \in S, \forall j \in F, \forall k \in N_k. \tag{7}$$

Among them, (5) ensures that the VNF can be either deployed in a new server k ($x_{ij}^k = 1$) or reusing the shared VNF which has been already instantiated in server k' ($\xi_{ij}^k = 1$). τ_{ij} represents the time when the server deploying the jth VNF starts to forward the service flow, T_{ij} represents the time when the jth VNF starts to process traffic, and N_k is a set of candidate servers that can instantiate the jth VNF. Constraint (6) ensures that the downstream VNF processing start time must be after the upstream VNF's data transmission and propagation has completed, and constraint (7) ensures that the service flow can be transmitted to the downstream VNF after the previous VNF is processed.

4. Reinforcement Learning-Based PSFC Scheduling

4.1. Q-Learning Method

Traditional exact and heuristic algorithms can find the optimal and near-optimal solution of the above problem in small-scale network scenarios, but in large-scale network scenarios, especially in the face of real-time online network services, traditional methods have difficulty achieving fast decision. Reinforcement learning (RL) is an autonomous decision-making algorithm in which an agent continuously interacts with its environment and adjusts the strategy through feedback to learn the best actions that maximizes the long-term cumulative reward.

This paper employs a lightweight RL method called the Q-learning algorithm. It is critical in the Q-learning algorithm to learn and maintain a state–action matrix Q, known as the Q-table. A state–action pair is represented by each Q-table element. The best action $\overline{a}$ of the current state s_t that satisfies the following relation is chosen based on the Q-table:

$$Q(s_t, \overline{a}) = \max_{a \in A} Q(s_t, a). \tag{8}$$

The ϵ-greedy strategy is used to select actions to prevent the RL algorithm from falling into a local optimum. The algorithm selects the best action according to (8) with a probability of $(1 - \epsilon)$ for each time step or randomly explores a new action that is not present in the Q-table with a probability of small ϵ. ϵ changes dynamically with the number of learning cycles in the Q-learning algorithm, e.g., $\epsilon = \frac{10}{\omega}$, where ω denotes the number of learning cycles. Actions are initially chosen in a more random manner, suggesting that the RL algorithm initially explores the environment through novel actions. The actions are mostly determined by the Q-table as it fills and converges.

When an action is chosen and executed, the system advances to the next state s_{t+1} and returns $r(s_t, a_t)$. The Q-table is then updated by:

$$Q(s_t, a_t) = Q(s_t, a_t) + \theta[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1} - Q(s_t, a_t))], \tag{9}$$

where (s_t, a_t) and $r(s_t, a_t)$ denote the state–action pair and instant reward at time t, s_{t+1} is the upcoming state at time $t + 1$, and θ and γ indicate the learning rate and discount factor of the RL agent, respectively.

4.2. Markov Decision Process

The RL agent continuously interacts with the environment and selects the optimal strategy according to the immediate reward $s(s_t, a_t)$ of the environment, called the Markov decision process. The basic elements of the Markov decision process are defined as:

State space s_t: the network state at time t, represented by the matrix $s_t = \{x_{11}, x_{12}, x_{13} ... x_{ij}\}$ where the diagonal elements $x_{ij}, (i = j)$ represent the resource

capacity of the ith server, and $x_{ij}, (i \neq j)$ represents the link delay between servers i and server j.

Action space a_t the action taken by the agent at time t, $a_t = \{x_{ij}^k, y_{j,j+1}^{k,k'}\}$, where $x_{ij}^k = \{0,1\}$, $x_{ij}^k = 1$ indicates that the jth VNF of s_i is processed by the server k, and 0 otherwise. $y_{j,j+1}^{k,k'} = \{0,1\}$, $y_{j,j+1}^{k,k'} = 1$ indicates that the traffic flow from the jth VNF instance on server k will be sent to the $j+1$th instance on server k', and 0 otherwise.

Reward function R_t: the RL method aims to find an optimal policy to maximize the cumulative reward $\mathcal{R}(t) = \sum_{t=1}^{T} r(t)$ while following the policy. This paper jointly optimizes the service end-to-end delay and the difference between parallel branches, and the instant reward consists of two parts, which is the weighted sum of the service end-to-end delay and the difference between parallel branches:

$$r_t = M - \alpha D_{e2e} + \beta D_{e2e}^f, \tag{10}$$

where M is a sufficiently large integer. It can be seen that the smaller the total end-to-end delay and delay difference between different parallel branches, the larger the reward function, where α and β are the coefficients of two optimization objectives.

The specific learning process is as follows: First, the agent initializes the Q-table and the initial state s_0. For each VNF in an SFC, the agent randomly selects actions based on the current state according to the e-greedy strategy or selects the current optimal strategy according to the current Q-table (lines 7–16). The learning processing will stop automatically if it exceeds the large exploration period *max_episodes* or the Q-table converges (line 8). In the early stage of the agent's exploration, the Q-table is still empty, and it is more inclined to explore random actions. When it learns to the later stage, it selects the server and traffic scheduling path according to the action value function. After selecting a server, it calculates the performance of traffic scheduling to this server, that is, the total service delay and delay difference (lines 17–20) and updates the Q-table. If the selected server has insufficient resource capacity or is already occupied by other services, a negative reward is returned (line 22). Finally, the agent selects the best action and updates *time_table* (lines 27–28). After executing the action, the system will enter the next state and update the system state to the current state (line 29). The detailed learning process is shown in Algorithm 1.

4.3. Algorithm Complexity

The training process of the RL agent will stop when the Q-table converges or reaches the *max_episode*, assuming E = *max_episode*. In the training phase of Algorithm 1, lines 7–26 run up to 10 E times, assuming the number of network function of each service is M. Then, lines 4–30 running 6 M + 10 ME operations, assuming there is a total of S services, the total running operations of lines 3–31 are 6 SM + 10 SME; therefore, the total complexity of Algorithm 1 is 6 SM + 10 SME +2. In real-world network services, the number of network functions of each service generally does not exceed 7 [4], so M can be ignored. In the simulation experiment, we set E = 500 due to the considerable convergence effect. In short, the complexity of the algorithm is $\mathcal{O}(SE)$ which indicates that the proposed algorithm has relatively low algorithm complexity. A specific running time comparison of the algorithm will be given in Section 5.3.5.

Algorithm 1: Delay-aware traffic scheduling algorithm based on reinforcement learning.

Data: $s_t, \alpha, \beta, \theta, \gamma, \varepsilon$
Result: $x_{ij}^{k}, y_{j,j+1}^{k,k'}$

Initialize: Q-table ;
Initialize state: s_0 ;
for $s_i \in S$ **do**
  for $j = 1$ *to* $|s_i|$ **do**
    Current state: s_t;
    $max\_episode$ = 500;
    for $\omega = 1$ *to* $max\_episodes$ **do**
      if $\omega >$ $max\_episode$ *or Q-table is converged* **then**
        break;
      else
        $\lambda \leftarrow$ randomly generate from $[0, 1]$;
        if $\lambda \geq \varepsilon$ **then**
          $a \leftarrow$ randomly select an action;
        else
          $a \leftarrow$ select action a according to (8);
        end
        if *selected server can host the VNF* **then**
          $D_{e2e} \leftarrow$ calculate end-to-end delay based on (2);
          $D_{e2e}^{f} \leftarrow$ calculate delay difference based on (3);
          $r_t \leftarrow$ calculates the instant reward according to (10);
        else
          $r_t \leftarrow$ -100;
        end
        update Q-table according to (9);
      end
    end
    select the best action $\bar{a} = \{x_{ij}^{k}, y_{j,j+1}^{k,k'}\}$ according to Q-table to scheduling f_{ij};
    add running time of f_{ij} into $time\_table$ of node $\bar{a}$;
    $s_t \leftarrow s_{t+1}$; // update current state
  end
end

5. Simulation Results and Discussions

5.1. Parameter Settings

We adopt four real-world network topologies with scales from 25 to 100 nodes from the Internet Topology Zoo [28] to verify the performance in different network scales. Each node is regarded as a host server, the resource capacity of each server is set to 100–250 units, the bandwidth between two servers is set to 5–10 Gbps, and 1–3 types of VNFs were instantiated in each server. SFC requests with lengths of 3–6 VNFs are randomly generated, PSFCs have all the possible architecture as defined in Figure 3a, and the processing delay of each VNF is randomly generated from 10 to 100 ms. The link delay is set to 25–30 ms according to the physical distance between two nodes. The average data rate of SFC is set to 0.1–1 Gbps, assuming that the resource requirement of VNF is proportional to the data rate. The learning rate θ and discount factor γ are set to 0.9 and 0.6 based on the best convergence performance, and the coefficients of α and β are set to 0.6 and 0.4, respectively. The specific setup of our simulation is listed in Table 4.

Table 4. Simulation Setting.

Parameter	Setting
topology	25–100 nodes
node resource capacity	250–300 units
link bandwidth capacity	5–10 Gbps
link delay	25–30 ms
SFC length	3–6 VNFs
VNF resource request	5–10 units/Gbps
VNF processing time	10–100 ms
average data rate	0.1–1 Gbps
learning rate θ	0.9
discount factor γ	0.6
D_{e2e} coefficient α	0.6
D^{f}_{e2e} coefficient β	0.4

5.2. Baseline Algorithms

We compared our proposed DASM method with the current advanced PSFC scheduling mechanism and two greedy traffic scheduling methods, listed as follows:

Joint deploying and scheduling mechanism of PSFCs based on reinforcement learning (JoRL) [4]: JoRL is a two-stage optimization method. In the first stage, it uses a parallelism optimization algorithm (POA) based on a bin-packing problem to optimize the parallelism degree of PSFC and then uses the *Q*-learning algorithm to determine the optimal deployment and scheduling strategy.

Greedy best availability algorithm (GBA): GBA schedules traffic to the best available server that has smallest queuing delay in the current queue.

Greedy shortest path algorithm (GSP): The GSP algorithm greedily chooses the shortest path to route the traffic, but the delay balance between differences is ignored.

This paper compares the scheduling benefits of different algorithms, including total SFC delay, delay difference between parallel branches, resource utilization, service acceptance rate, and algorithm running time, defined as follows:

- SFC end-to-end delay: including VNF processing delay and link propagation delay. In a distributed network, the data transmission delay can be ignored in comparison to the processing delay and link delay.
- Delay difference: defined as the sum of the delay differences between different parallel branches.
- Resource utilization: the resource utilization of a node is defined as the ratio of the used resources in the node to the total resources;
- Bandwidth utilization: the bandwidth utilization is defined as the ratio of the used bandwidth to the total bandwidth.
- Service acceptance rate: defined as the percentage of successful SFCs of all 100 SFCs.
- Running time: defined as the running time for the algorithm to deploy and schedule 100 SFCs from ingress to egress node.

5.3. Simulation Results

The simulation results are shown as follows. In order to avoid the accidental circumstance of service which may cause the indeterminacy of experimental results, we deployed 100 SFCs in each episode, repeated the experiments up to 500 episodes (as shown in Algorithm 1), and took the average as the output.

5.3.1. SFC End-to-End Delay

Figure 6 shows the service end-to-end delay of four algorithms under different network scales, i.e., from 25 nodes to 100 nodes. As expected, GSP has the smallest end-to-end delay, because the GSP algorithm greedily selects the path with the smallest delay for traffic scheduling each time, but ignores the delay balance between different parallel branches, which is not the optimal strategy in our proposed scenario. The proposed DASM method has a lower SFC delay than the GSP algorithm. Although the PSFC can be converted into serial SFCs, the traffic can still be scheduled according to the shortest path, but the proposed method also considers the delay balance between multiple branches, so it is not solely the pursuit of the lowest delay. The JoRL algorithm has a slightly higher latency than the DASM because when the JoRL selects the traffic scheduling path, it can only schedule the traffic based on the previous deployment. This solution may cause the traffic to be far away from the destination node and cause delay imbalance, whereas the GBA algorithm chooses the most available server for traffic scheduling each time, effectively reducing the queuing delay but also bringing traffic detour.

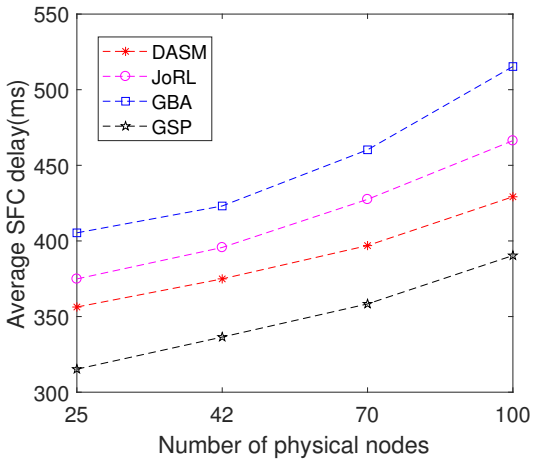

Figure 6. Average SFC delay with the increasing topology scales.

We further verified the traffic scheduling delay of four algorithms under different SFC lengths. The simulation was implemented on a 100-node network topology, and the SFC length was set from three to six VNFs. The simulation results are shown in Figure 7, which has similar trends as Figure 6. The service end-to-end delay of the GSP algorithm is the smallest. However, DASM is still second only to the GSP algorithm, and the performances of the JoRL and GBA algorithms have changed significantly with the increase in SFCs length.

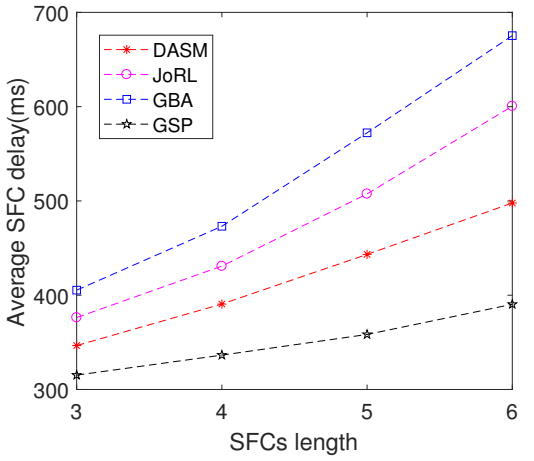

Figure 7. Average SFCs delay with the increasing SFCs lengths.

5.3.2. Delay Difference between Parallel Branches

In a serial SFC, the sole optimization goal is to minimize the latency, but in parallelized SFC, minimizing the delay difference between different parallel branches is equally important because the service delay is determined by the highest parallel branch, and if the delay difference between parallel branches is too large, the processing speed between parallel branches will be inconsistent, resulting in performance degradation of parallelized SFCs and resource wasting. As shown in Figure 8, our proposed DASM algorithm has the lowest delay difference between the different parallel branches, because reducing the delay difference between two serial SFCs is easier than reducing them in a parallelized SFC with fixed ingress and egress nodes. JoRL is second to our proposed scheme, with the network topology scale increasing the optimize performance of our proposed scheme is more obviously than JoRL algorithm. The GBA and GSP algorithms have the highest delay difference because GBA and GSP algorithms aim to minimize the queuing delay and link delay, and neither of these two algorithms involve optimization of delay balance.

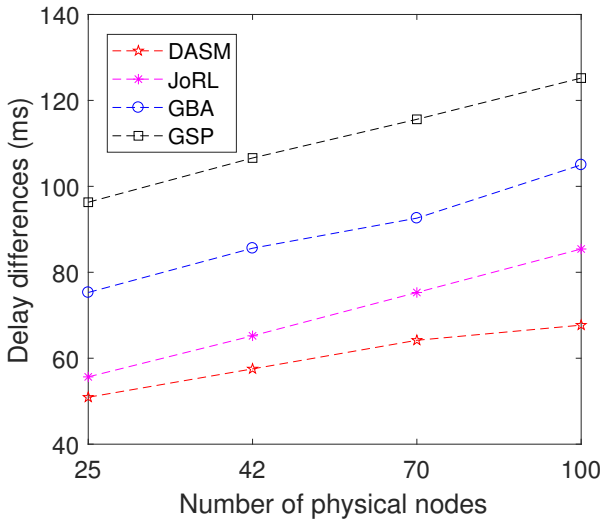

Figure 8. Delay difference of parallel branches.

5.3.3. Resource Utilization

The resource utilization rate is shown in Figure 9. The resource utilization rates are gradually decreased with the increasing of topology scales, due to increasing server resource capacity. The GBA algorithm tends to select the idlest server for traffic scheduling, resulting in the lowest resource utilization rate. This is followed by JoRL, and the resource utilization rate of DASM is slightly higher than that of JoRL. This is because the conversion process of DASM brings additional cost and resource consumption, especially because elimination of the dependency of the upstream serial VNF in PSFCs requires the additional instantiation consumption of VNFs, whereas there is no such process in JoRL algorithm. Furthermore, the GSP algorithm tends to selects the short path for traffic scheduling, resulting the most centralized VNF deployment; hence, the GSP has the highest resource utilization rate.

While bandwidth consumption has the same trend with increasing topology as the resource utilization rate (as shown in Figure 10), GSP has the lowest bandwidth consumption due to the centralized VNF deployment and traffic scheduling, and GBA tends to schedule the traffic to the idlest server at the expense of traffic path; thus, GBA has the highest bandwidth consumption. DASM has the second-highest bandwidth consumption, and DASM has a higher bandwidth consumption when compared to JoRL due to the additional VNF instantiation and traffic routing path.

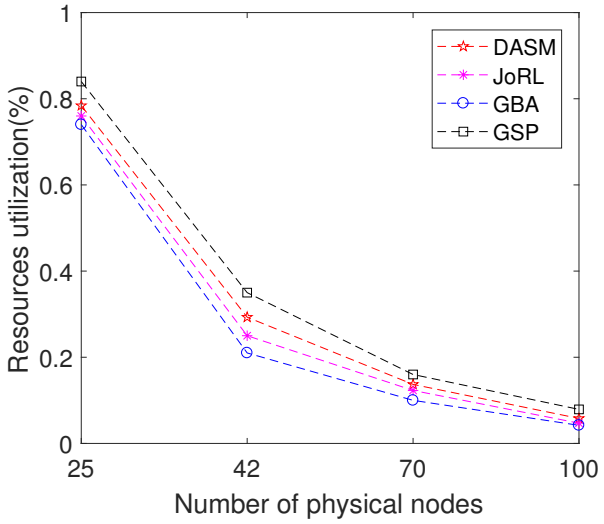

Figure 9. Resource utilization rate.

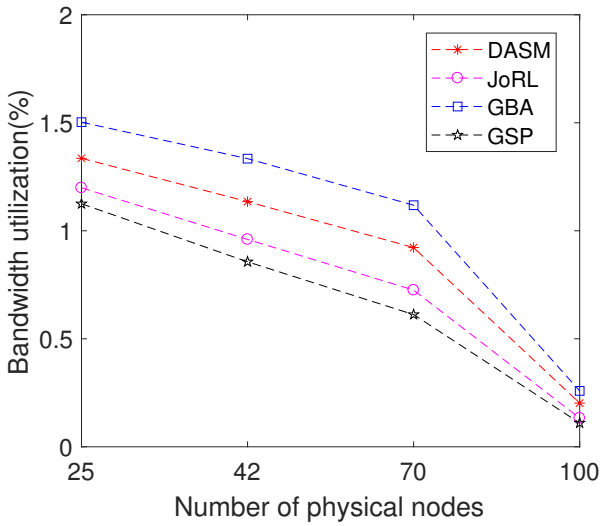

Figure 10. Bandwidth consumption rate.

5.3.4. Service Acceptance Rate

We verified the service acceptance rate of algorithms in different network topologies. In general, as the scale of the network topology increases, the service acceptance rates of all algorithms gradually increase because the number of nodes available for selection increases and more services can be satisfied. The service acceptance rate is shown in Figure 11. Both DASM and JoRL algorithms have a relatively high service acceptance rate, which can even approach 90%. With an increase in network topology, GBA and GSP algorithm cannot achieve the considerable acceptance rates of JoRL and DASM. Both JoRL and DASM realize the traffic scheduling based on the reinforcement learning method; however, DASM still has a relatively higher acceptance rate than JoRL, which indicates that the proposed DASM scheme indeed improves the service throughput by reducing the dependency of parallel branches.

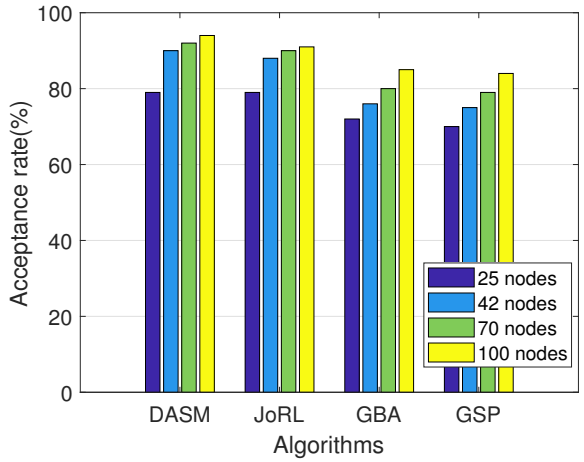

Figure 11. Service acceptance rate.

5.3.5. Running Time

The complexity of the algorithm reflects the adaptability and application feasibility of the algorithm, which is verified by the running time of the algorithm. The result is shown in Figure 12. The GSP algorithm has the highest running time due to the complex shortest path searching. Especially with an increase in network scale, it is not appropriate to find the shortest path globally, whereas the greedy algorithm has the relatively low algorithm complexity due to the near-optimal result searching. Our proposed DASM and JoRL realize traffic scheduling based on reinforcement learning method, but DASM still has a relatively lower running time compared to JoRL, which indicates that the proposed DASM scheme indeed reduces the complexity of the original problem.

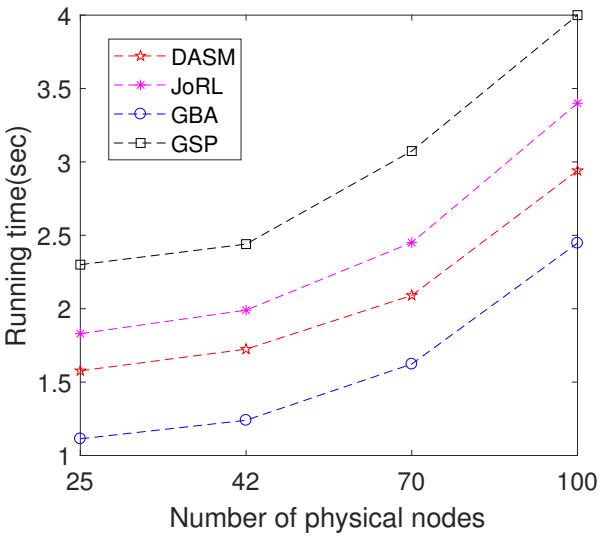

Figure 12. Algorithm running time.

6. Discussions

This paper first systematically summarizes previous studies and finds that the current research on traffic scheduling of parallelized SFC is still in an open stage. The current research on serialized SFCs is not suitable for PSFCs scenario, due to the special structural and VNF dependencies of parallelized SFC. Moreover, although some researchers and our previous studies have made some attempts, there is still a lack of a general methods for supporting PSFC traffic scheduling.

To this end, we propose a delay-aware traffic scheduling mechanism (DASM) for PSFCs. In order to support low-delay and delay-balanced traffic scheduling, PSFCs are first

converted into multiple serial SFCs. Then, we convert multiple serial SFCs into independent serial SFCs by reducing the dependence of VNF between parallel branches. The delay-aware traffic scheduling of multiple parallel branches is implemented based on RL, i.e., the *Q*-learning method, which jointly considers the service end-to-end delay and delay difference between parallel branches. The simulation results show that the proposed DASM in this paper can effectively reduce the complexity of the problem and significantly improve the throughput of the service when compared to greedy-based and current advanced PSFC scheduling methods.

Although the RL-based method proposed in this paper has shown good performance in the current simulation experiments, there are still aspects that can be further improved. For example, (1) the dimension of the *Q*-table will increase with the increase in the scale of the network, which may require further optimization in very large networks. (2) Other aspects in a real-world deployment, such as energy efficiency, need to be further studied in future work. To tackle these limitations, deep reinforcement learning (DRL)-based methods can be promising candidates. DRL methods can fit the *Q*-table based on a neural network to solve the problem of dimensional explosion, thus improving the algorithm's efficiency.

7. Conclusions

The parallel processing of SFC (i.e, PSFC) breaking through the delay bottleneck of traditional serial SFC, is expected to become a key technology for low-latency delivery in future networks. To support efficient PSFC provision, this paper proposes a delay-aware traffic scheduling mechanism (DASM)/ DASM first converted the PSFC into multiple serial SFCs with minimum VNF dependency to search for a unified solution for PSFCs. DASM further proposed a delay-aware traffic scheduling method based on the RL for converted parallel branches, which jointly considers the total end-to-end delay as well as the delay difference between parallel branches. Simulation results showed that the DASM of PSFCs proposed in this paper can reduce the complexity of the original problem and outperform the current advanced PSFCs scheduling method in terms of delay performance and system throughput. Further work is in progress to consider DRL-based traffic scheduling in larger-scale IoT scenarios.

Author Contributions: Conceptualization, methodology: Z.H. and D.L.; software, validation, formal analysis, visualization, writing—original draft preparation: Z.H. and C.W.; writing—review and editing, project administration: D.L.; supervision, funding acquisition: D.L. and H.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China No. 2019YFB1804400, and MUST Faculty Research Grants No. FRG-21-031-IINGI.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Sun, C.; Bi, J.; Zheng, Z.; Yu, H.; Hu, H. NFP: Enabling network function parallelism in NFV. In Proceedings of the 2017 Conference of the ACM Special Interest Group on Data Communication, Los Angeles, CA, USA, 21–25 August 2017; pp. 43–56.
2. Liu, M.; Feng, G.; Zhou, J.; Qin, S. Joint two-tier network function parallelization on multicore platform. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 990–1004. [CrossRef]
3. Zhang, Y.; Anwer, B.; Gopalakrishnan, V.; Han, B.; Reich, J.; Shaikh, A.; Zhang, Z.L. Parabox: Exploiting parallelism for virtual network functions in service chaining. In Proceedings of the 2017 Symposium on SDN Research, Santa Clara, CA, USA, 3–4 April 2017; pp. 143–149.
4. Cai, J.; Huang, Z.; Liao, L.; Luo, J.; Liu, W.X. APPM: Adaptive parallel processing mechanism for service function chains. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 1540–1555. [CrossRef]
5. Cai, J.; Huang, Z.; Luo, J.; Liu, Y.; Zhao, H.; Liao, L. Composing and deploying parallelized service function chains. *J. Netw. Comput. Appl.* **2020**, *163*, 102637. [CrossRef]

6. Santos, G.L.; Endo, P.T.; Lynn, T.; Sadok, D.; Kelner, J. A reinforcement learning-based approach for availability-aware service function chain placement in large-scale networks. *Future Gener. Comput. Syst.* **2022**, *136*, 93–109. [CrossRef]
7. Alhussein, O.; Zhuang, W. Dynamic Topology Design of NFV-Enabled Services Using Deep Reinforcement Learning. *IEEE Trans. Cogn. Commun. Netw.* **2022**, *8*, 1228–1238. [CrossRef]
8. Tofighy, S.; Rahmanian, A.A.; Ghobaei-Arani, M. An ensemble CPU load prediction algorithm using a Bayesian information criterion and smooth filters in a cloud computing environment. *Softw. Pract. Exp.* **2018**, *48*, 2257–2277. [CrossRef]
9. Adoga, H.U.; Pezaros, D.P. Network Function Virtualization and Service Function Chaining Frameworks: A Comprehensive Review of Requirements, Objectives, Implementations, and Open Research Challenges. *Future Internet* **2022**, *14*, 59. [CrossRef]
10. Baek, H.; Jang, I.; Ko, H.; Pack, S. Order dependency-aware service function placement in service function chaining. In Proceedings of the 2017 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 18–20 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 193–195.
11. Ayoubi, S.; Chowdhury, S.R.; Boutaba, R. Breaking service function chains with Khaleesi. In Proceedings of the 2018 IFIP Networking Conference (IFIP Networking) and Workshops, Zurich, Switzerland, 14–16 May 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 64–72.
12. Chowdhary, A.; Huang, D. Sdn based network function parallelism in cloud. In Proceedings of the 2019 International Conference on Computing, Networking and Communications (ICNC), Honolulu, HI, USA, 18–21 February 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 486–490.
13. Wang, R.; Luo, J.; Dong, F.; Shen, D. ParaNF: enabling delay-balanced network function parallelism in NFV. In Proceedings of the 2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD), Porto, Portugal, 6–8 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 392–397.
14. Jiang, Y.; Cui, Y.; Wu, W.; Xu, Z.; Gu, J.; Ramakrishnan, K.; He, Y.; Qian, X. Speedybox: Low-latency NFV service chains with cross-NF runtime consolidation. In Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, 7–10 July 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 68–79.
15. Rui, L.; Chen, X.; Gao, Z.; Li, W.; Qiu, X.; Meng, L. Petri net-based reliability assessment and migration optimization strategy of SFC. *IEEE Trans. Netw. Serv. Manag.* **2020**, *18*, 167–181. [CrossRef]
16. Hu, Y.; Li, T. Enabling efficient network service function chain deployment on heterogeneous server platform. In Proceedings of the 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), Vienna, Austria, 24–28 February 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 27–39.
17. Engelmann, A.; Jukan, A. A reliability study of parallelized VNF chaining. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Chengdu, China, 19–21 December 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–6.
18. Engelmann, A.; Jukan, A.; Pries, R. On coding for reliable VNF chaining in DCNs. In Proceedings of the 2019 15th International Conference on the Design of Reliable Communication Networks (DRCN), Coimbra, Portugal, 19–21 March 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 83–90.
19. Zhang, Y.; Zhang, Z.L.; Han, B. HybridSFC: Accelerating service function chains with parallelism. In Proceedings of the 2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Dallas, TX, USA, 12–14 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–7.
20. Xie, S.; Ma, J.; Zhao, J. FlexChain: Bridging parallelism and placement for service function chains. *IEEE Trans. Netw. Serv. Manag.* **2020**, *18*, 195–208. [CrossRef]
21. Sun, G.; Chen, Z.; Yu, H.; Du, X.; Guizani, M. Online parallelized service function chain orchestration in data center networks. *IEEE Access* **2019**, *7*, 100147–100161. [CrossRef]
22. Bao, W.; Yuan, D.; Zhou, B.B.; Zomaya, A.Y. Prune and plant: Efficient placement and parallelism of virtual network functions. *IEEE Trans. Comput.* **2020**, *69*, 800–811. [CrossRef]
23. Luo, J.; Li, J.; Jiao, L.; Cai, J. On the effective parallelization and near-optimal deployment of service function chains. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 1238–1255. [CrossRef]
24. Wang, M.; Cheng, B.; Wang, S.; Chen, J. Availability-and traffic-aware placement of parallelized SFC in data center networks. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 182–194. [CrossRef]
25. Lin, I.C.; Yeh, Y.H.; Lin, K.C.J. Toward Optimal Partial Parallelization for Service Function Chaining. *IEEE/ACM Trans. Netw.* **2021**, *29*, 2033–2044. [CrossRef]
26. Lin, K.C.J.; Chou, P.L. VNF Embedding and Assignment for Network Function Parallelism. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 1006–1016. [CrossRef]
27. Zheng, D.; Shen, G.; Cao, X.; Mukherjee, B. Towards Optimal Parallelism-Aware Service Chaining and Embedding. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 2063–2077. [CrossRef]
28. Knight, S.; Nguyen, H.X.; Falkner, N.; Bowden, R.; Roughan, M. The internet topology zoo. *IEEE J. Sel. Areas Commun.* **2011**, *29*, 1765–1775. [CrossRef]