FlexSFC: Flexible Resource Allocation and VNF Parallelism for Improved SFC Placement

Sagar Agarwal, Venkatarami Reddy Chintapalli, Bheemarjuna Reddy Tamma
Indian Institute of Technology Hyderabad - INDIA
Email: {cs20mtech11005, cs17resch01007, tbr}@iith.ac.in

***Abstract*—To reduce the processing delay from the sequentially running virtual network functions (VNFs) in a service function chain (SFC), network function parallelism (NFP) is introduced that allows VNFs of the SFC to run in parallel. Existing NFP solutions only focused on improving parallelism benefits without paying much attention to resource utilization while deploying VNFs of SFCs. We take advantage of resource-delay dependency to propose a flexible and efficient parallelized SFC placement mechanism called *FlexSFC* which determines the optimal SFC placement while reducing resource usage and meeting end-to-end delay guarantees of the SFCs deployed. Initial results show that *FlexSFC* guarantees the end-to-end delay requirement with better resource utilization and SFC acceptance rate than the state-of-the-art approaches.**

I. Introduction

Network Functions Virtualization (NFV) is introduced to address the limitations of traditional proprietary middleboxes with more flexible Virtual Network Functions (VNFs). In NFV, a network service is delivered by a series of predefined VNFs called Service Function Chaining (SFC). Serially running VNFs in an SFC imposes propagation delay as well as processing delay; in some cases, higher processing delay becomes a bottleneck for SFC acceptance by causing an increase in end-to-end delay. To mitigate the impact caused by the processing delay, Network Function Parallelism (NFP) is introduced which tries to deploy VNFs of a given SFC request in parallel at one of the physical servers provisioned for SFC placement [1].

As demonstrated in recent works [1], [2], if the operations of two VNFs do not conflict then those VNFs can be executed in parallel. As an example, Deep Packet Inspection (DPI) can be executed in parallel with Flow Monitor (FM) as they only inspect packet data streams without any modifications. On the other hand, DPI and encryption cannot execute in parallel as both might modify the packets. For instance, a typical sequential SFC, $VNF_1 \rightarrow VNF_2 \rightarrow VNF_3 \rightarrow VNF_4 \rightarrow VNF_5$, is shown in Fig. 1. The total processing delay of the sequential SFC is 123 ms, which can be reduced by 27% (33 ms) through parallelization of VNF_2, VNF_3, and VNF_4 without effecting the performance of the SFC.

Recently, SFC placement using NFP has received a lot of attention [3]–[6]. But, these works overlooked the aspect of resource utilization while placing SFC requests with consideration of parallelism opportunities. Furthermore, these works have two major limitations: (1) Parallelized SFC strains the system resources. For instance, the copying and merging modules are designed to copy packets to multiple VNFs running in parallel and merge the parallelly processed packets to enable SFC parallelization in a network environment [1], [2]. It adds an overhead on network resources for delivering the duplicate packets and computational overhead for copying and merging the packet payloads; (2) In an SFC, the processing delays of parallel VNFs are usually dissimilar, which leads to packet deposition problem. Consequently, merging module has to allocate additional memory resources to buffer the early-arrived packets. A significant imbalance in processing delays between parallelized VNFs may cause severe packet deposition along with higher memory resource consumption. To address the above challenges, we propose a flexible and efficient SFC placement mechanism called *FlexSFC*. To overcome the first limitation of existing works, *FlexSFC* assumes all the parallelized VNF instances in an SFC request are deployed in the same physical machine to avoid non-negligible cost of duplicating and merging packets. To address the second limitation, Adaptive Parallel Processing Mechanism (APPM) [4] serializes some parallel VNFs to balance the delay based on the peak VNF processing delay of the parallelized VNFs. But, it fails to get full benefits of NFP. *FlexSFC* approaches this problem differently by addressing the question of "How to adjust resources allocated to the parallelized VNFs in order to meet the peak delay of parallelized VNF in an SFC?". Recent studies [7], [8] have shown that within a given range of resource allocation, the processing delay of a VNF is a linearly non-increasing function of the resource allocated to it. We have verified this behaviour by conducting experiments on two different VNFs. *FlexSFC* balances the processing delay of parallelized VNFs by flexibly allocating resources to VNFs based on the peak VNF delay of parallelized VNFs, while not losing the benefits of VNF parallelism. It helps to avoid the extra allocation of resources to VNFs. As a result, the resources saved can be used for other SFC requests, resulting in a higher SFC acceptance rate.

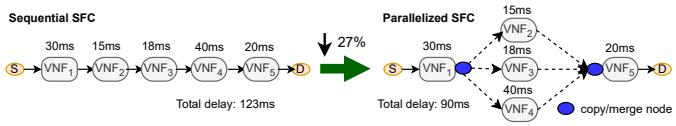
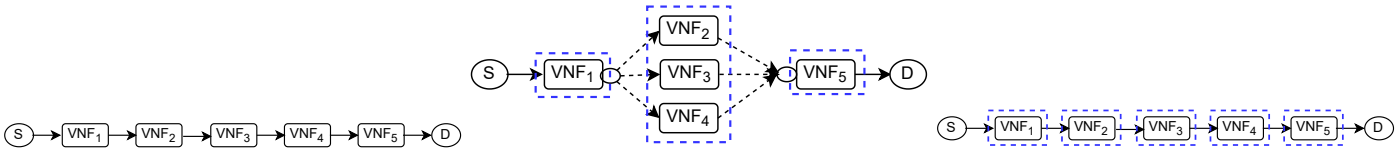


Figure 1: Traditional sequential SFC Vs. SFC parallelism.

II. System Model and Problem Description

The service provider network is modelled as an undirected graph $G = (V, E)$, where V is the set of nodes (i.e., servers) and E is the set of links which interconnect the nodes. Each



(a) Sequential SFC request. (b) PSFC with PE with multiple branches. (c) PSFC with all PEs with single VNF

Figure 2: Example of different P-SFCs. S and D represent the source and destination, respectively.

node can host multiple instances of different VNFs, based on its resource capacity. Each link has certain bandwidth capacity and propagation delay. M denotes a set of SFCs and each SFC request $m \in M$ is represented as a vector (s_m, d_m, δ_m, b_m, sc_m), where s_m and d_m are the source and destination nodes, b_m represents the bandwidth demand, δ_m is the tolerable end-to-end delay of the SFC request, and sc_m is an ordered interconnection of VNFs through which packets are traversed. In this work, we consider the case of each VNF requiring some computing (CPU) resources which are measured in units.

Parallelism-based SFC: VNFs perform various operations on ingress data packets. For non-conflicting VNFs, NFP enables parallel VNF execution to quickly process the packets for the same flow [1]. We follow the works [1], [9] to determine whether any two VNFs of an SFC are independent based on their operations and to construct the parallelism-based SFC (PSFC). A PSFC is comprised of a series of Parallel Entities (PEs), each of which consists of either a set of parallelizable VNFs or a single VNF. Figs. 2b and 2c illustrate two possible PSFCs that are composed from the sequential SFC shown in Fig. 2a, where a blue dashed-line rectangle represents a PE. The PSFC in Fig. 2c includes five PEs, each of which contains one VNF, i.e., five VNFs will be sequentially executed in Fig. 2c.

End-to-end delay of PSFC: The processing delay of i^{th} PE (ψ_i) is denoted by $PD_{\psi i}$, which depends on the VNF that has the highest processing delay among all parallelized VNFs in ψ_i. According to Eq. (1), only the VNF with the highest processing delay determines the processing delay of the PE, while processing delay due to other VNFs can be safely ignored. τ_v represents the processing delay of VNF v.

$$PD_{\psi i} = \max_{\{v \in \psi_i\}} \tau_v \tag{1}$$

The end-to-end delay of a packet traversing an PSFC is defined as the sum of VNF processing delays and propagation delays in the deployed path. It is calculated as follows:

$$\sum_{\psi_i \in B, v \in V} PD_{\psi_i}^{v} + \sum_{l \in p_{sc}} T(l) \tag{2}$$

where $PD_{\psi_i}^{v}$ represents the processing delay of PE ψ_i on node v and B represents the set of PEs including source and destination in a PSFC. p_{sc} denotes the path traversed by each placed VNF and $T(l)$ indicates the link delay of link l.

In an NFV-enabled network, the service provider requires to optimally deploy the PSFC requests to maximize the profit while ensuring the delay requirement of each PSFC request. The problem can be described as: **given:** a physical network topology G and a set of PSFC requests M, for each PSFC request, **determine:** 1) how to route the PSFC in G and where to place VNF instances; 2) how to allocate processing resources of the underlying nodes to the corresponding VNFs; 3) **maximize** the number of deployed PSFC requests, and 4) **ensure** end-to-end delay requirements of the PSFC requests.

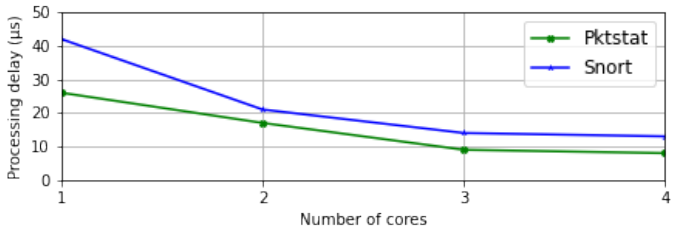


Figure 3: Resource-delay dependency for Pktstat and Snort.

III. Motivation

A. Resource-Delay Dependency

The works in [7], [8] show that the processing delay of a VNF is impacted by the amount of resources allocated to it. Processing delay of any VNF can be defined as a linear function of system resources allocated to it [8]. It indicates that when the amount of resources allocated to a VNF increases, the processing delay decreases. However, this behaviour is only observed for a certain range of resource allocation. It can be defined by Eq. (3):

$$D_{vnf}^{d} = h(\Phi_{alloc}) = \alpha \times \Phi_{alloc} + \beta \tag{3}$$

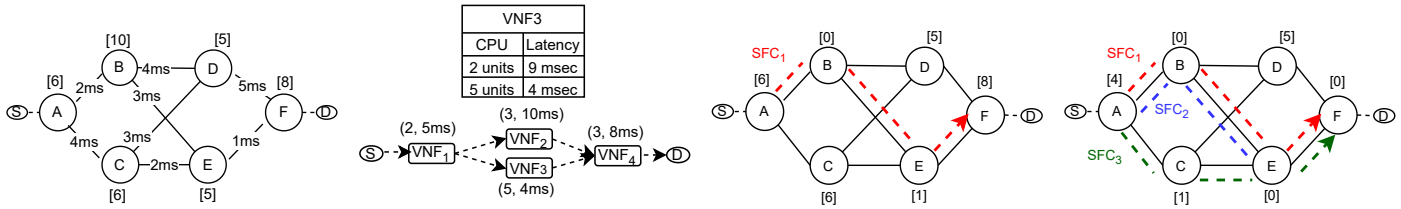
where α and β are given by Eq. (4) and Eq. (5), respectively.

$$\alpha = \frac{\zeta_{max}^{vnf} - \zeta_{min}^{vnf}}{\eta_{min}^{vnf} - \eta_{max}^{vnf}} \tag{4}$$

$$\beta = \frac{(\zeta_{min}^{vnf} \times \eta_{min}^{vnf}) - (\zeta_{max}^{vnf} \times \eta_{max}^{vnf})}{\eta_{min}^{vnf} - \eta_{max}^{vnf}} \tag{5}$$

Here, the range of the resources which can be allocated to a VNF is $[\eta_{min}^{vnf}, \eta_{max}^{vnf}]$ and the corresponding range of processing delays is in $[\zeta_{max}^{vnf}, \zeta_{min}^{vnf}]$. Where, ζ_{min}^{vnf} is the minimum delay which can be achieved when the maximum number of resources (η_{max}^{vnf}) are allocated to VNF and similarly ζ_{max}^{vnf} is the maximum delay which can be achieved when the minimum resources (η_{min}^{vnf}) are allocated to the VNF. It should be noted that after some point, allocation of the additional resources does not help in decreasing the VNF processing delay further [8].

Experimental Observations: To assert this resource-delay dependency in real-world scenarios, we choose two different VNFs *Snort* and *Pktstat* for experimentation on a machine with 12 core Intel i7-4770 3.60 GHz CPU with 32 GB of RAM,



(a) Network topology. (b) PSFC request. (c) Using PARC approach. (d) FlexSFC.

Figure 4: An example of parallelized SFC placement considering flexible resource allocation.

Ubuntu 18.04 LTS. As shown in Fig. 3, we observe that the processing delay of a VNF increases super-linearly with the decrease in its resource allocation. It is worth noting that after allocation of certain amount of resources, we did not observe any significant reduction in delay. However, even with the same resource allocation (in terms of CPU Cores), processing delay varied depending on the VNF's processing logic.

B. Why Accurate VNF Resource Allocation Matters?

Existing works did not consider the resource-delay dependency and assigned a predefined, fixed amount of resources to all parallelized VNF instances. Such strict resource allocation can negatively impact the quality of the placement. A significant difference in processing delays between parallelized VNFs results in severe packet deposition along with higher memory resource consumption. It can be overcome through balancing the processing delays of parallelized VNFs in a PE by scaling down the resources of VNFs (using the resource-delay dependency) based on the peak VNF delay of the PE, without losing the benefits of parallelization. This helps to avoid over provisioning of resources to VNFs inside a PE. As a result, the resources saved can be used for accepting SFC requests, resulting in a higher acceptance rate. For example, as shown in Fig. 1, the resources for VNF_2 and VNF_3 can be scaled down before deploying PSFC to minimize the delay difference among the three VNFs placed in parallel.

It should be noticed that all VNFs in a PE should be deployed on the same physical server (node) to avoid the cost of duplicating and merging packets. Eq. (6) is used to find the delay difference among VNFs in a PE based on the peak VNF processing delay and scales down the resources accordingly. Where, Δt_i represents delay difference between i^{th} VNF of ψ_j PE ($PD_{\psi_j^i}$) and the peak VNF delay ($PD_{\psi_j^c}$) of the same PE.

$$\Delta t_i = \sum_{i \in \psi_j} PD_{\psi_j^c} - PD_{\psi_j^i} \tag{6}$$

IV. Heuristic Algorithm for Resource-aware Parallelized SFC Deployment

This work mainly focuses on designing a mechanism for deploying PSFC requests while avoiding unnecessary resource allocation to VNFs and ensuring end-to-end delay requirement. Since the PSFC placement problem is proved to be NP-hard [3], [4], [9], we propose a flexible and efficient heuristic algorithm named *FlexSFC* to determine the optimal parallelized SFC placement while reducing resource usage and meeting end-to-end delay guarantees of the PSFCs deployed.

Algorithm 1 provides an overview of *FlexSFC*. It takes a network topology G, a set of PSFC requests, VNFs' resource-delay dependency table as inputs, and returns PSFC placement and acceptance ratio as the output. VNF resource-delay dependency table contains the information of maximum and minimum resources allocated to a VNF and its corresponding delays. SFCs are parallelized after finding the dependency among VNFs, as provided in [3], [9]. We first initialize the acceptance ratio Υ and the number of accepted requests μ to zero, and the total number of requests is stored in variable λ. *Acceptance ratio* is defined as the ratio of total number of PSFC requests accepted over the total number of requests submitted.

FlexSFC algorithm consists of two parts. In the algorithm's first part (lines 2-13), we scale the PSFCs ($\in M$). Line 2 sorts the PSFCs in non-decreasing order based on total resources required. Then, for each PSFC, the maximum delay among each PE is calculated. Based on the maximum delay ($PD_{\psi i}$) calculated, we use Eqs. (3), (4), and (5) to scale down the VNFs resources in each of the PE of the PSFC (lines 5-11). The new set of PSFCs after scaling is stored in set M'. In the second part of the algorithm, we try to deploy PSFCs in the given network. For each PSFC in M', k-shortest paths are computed using $Yen's$ algorithm (line 15). These paths are computed based on the link delays between source and destination, satisfying the end-to-end delay and bandwidth constraints. To place VNFs of a PE, we find a node with maximum residual capacity, deploy all VNFs in it, and then place the remaining VNFs of PSFC in that path. If all VNFs are deployed in the current path, increase the accept count and move to the subsequent PSFC request; otherwise, remaining k-paths are explored. Finally, it returns the acceptance ratio.

A. An Illustrative Example

Fig. 4 shows a motivational example. In the network of Fig. 4a, five different nodes are available with different resource capacities to deploy VNFs. The value in square brackets at each node represents the capacity available and edge represents the link delay. Fig. 4b shows the PSFC request of four VNFs where VNF_2 and VNF_3 can run in parallel. The resource requirement and corresponding delay are represented at each VNF. The tolerable end-to-end delay of this request is assumed as 30 ms. We assume three SFC requests (i.e., SFC_1, SFC_2, and SFC_3) with the same set of VNFs for deploying in the given network. As shown in Fig. 4c, a state-of-the-art scheme named *PARC* [3] prefers to choose the path with the minimum end-to-end delay. It places VNFs based on the maximum

Algorithm 1 FlexSFC Algorithm

Input: $G(V, E)$, M, VNF resource-delay dependency table
Output: PSFC Placement and Acceptance Ratio (Υ).

$\Upsilon \leftarrow 0$; $\mu \leftarrow 0$; $\lambda \leftarrow \Sigma_M s$
Sort SFCs in non-decreasing order based on total resources required
$M' \leftarrow \{\}$
for each $s \in M$ **do**
  for each $\psi_i \in B_s$ **do**
    $PD_{\psi i} \leftarrow \max\{\tau_v | v \in \psi_i\}$
    for $v \in \psi_i$ **do**
      Scale VNFs using Eqs. (3), (4) and (5) based on the value of $PD_{\psi i}$
    end for
  end for
  $M' \leftarrow$ store the scaled SFC
end for
for each $s' \in M'$ **do**
  $K_{paths} \leftarrow$ compute k-shortest paths which satisfy tolerable end-to-end delay and bandwidth constraints.
  for each path $k \in K_{paths}$ **do**
    Find the node with maximum residual resources and place the parallelized VNFs in that node
    Deploy rest of the VNFs in the path
    if path k is able to place s' **then**
      $\mu \leftarrow \mu + 1$
      break
    end if
  end for
end for
$\Upsilon \leftarrow \mu/\lambda$
return Υ

residual resources at nodes along the path and places parallelized VNFs on the same node. Thus, SFC_1 is deployed as follows: $A \rightarrow B(VNF_1, VNF_2, VNF_3, VNF_4) \rightarrow E \rightarrow F$. Resultantly, SFC_2 and SFC_3 are rejected due to lack of resource availability. Even if we consider other paths (i.e., $A \rightarrow B \rightarrow D \rightarrow F$ and $A \rightarrow C \rightarrow E \rightarrow F$) which meet the delay requirement, *PARC* failed to find nodes with sufficient resource availability.

As shown in Fig. 4d, our proposed approach *FlexSFC* considers the VNF resource-delay dependency for parallelized VNFs and scales down the resources to VNFs in order to minimize the processing delay difference among the parallelized VNFs. After scaling down the resources, the total amount of required resources for the parallelized VNFs (i.e., VNF_2 and VNF_3) is reduced to 5 from 8 units. Note that the resource-delay dependency table for VNF_3 is presented in Fig. 4b. As a result, *FlexSFC* accepts all three SFC requests. SFC_1, SFC_2, SFC_3 are deployed on the following paths: $A \rightarrow B(VNF_1, VNF_2, VNF_3, VNF_4) \rightarrow E \rightarrow F$, $A \rightarrow B \rightarrow E(VNF_1) \rightarrow F(VNF_2, VNF_3, VNF_4)$, and $A(VNF_1) \rightarrow C(VNF_2, VNF_3) \rightarrow E(VNF_4) \rightarrow F$, respectively.

V. Performance Evaluation

We evaluate the performance of the proposed *FlexSFC* algorithm using the well-known USNET network topology. To evaluate the performance, we developed a C++ based simulator. The network topology and SFC request parameters are taken from [4], [5], which are shown in Table I. Each experiment is repeated 50 times and average results are presented. Similar results are obtained for the large scale CORONET network topology but not included due to space limitations.

Performance comparison: We compare *FlexSFC* with the following three state-of-the-art algorithms: 1) *PARC [3]:* It places VNFs based on the maximum residual resources at nodes and assumes to deploy all of the parallelized VNFs in the same server; 2) *NFP [1]:* It only allows to place entire SFC in one server; 3) *FRAM [8]*: *FRAM* did not consider parallelism in SFC placement, instead it focused on resource-delay dependency and allocating resources to VNFs to meet end-to-end tolerable delay.

Performance metrics: To evaluate *FlexSFC*, we consider the following performance metrics: 1) *SFC acceptance ratio:* It is the ratio of SFC requests accepted over the total requests. An SFC request is said to be accepted when its end-to-end delay requirement is met; 2) *Average resource utilization:* It is the ratio of remaining resources after placing the SFCs over the total resources; 3) *Average end-to-end delay:* It is the average delay for each SFC taken after the placement of SFCs.

A. Simulation Results

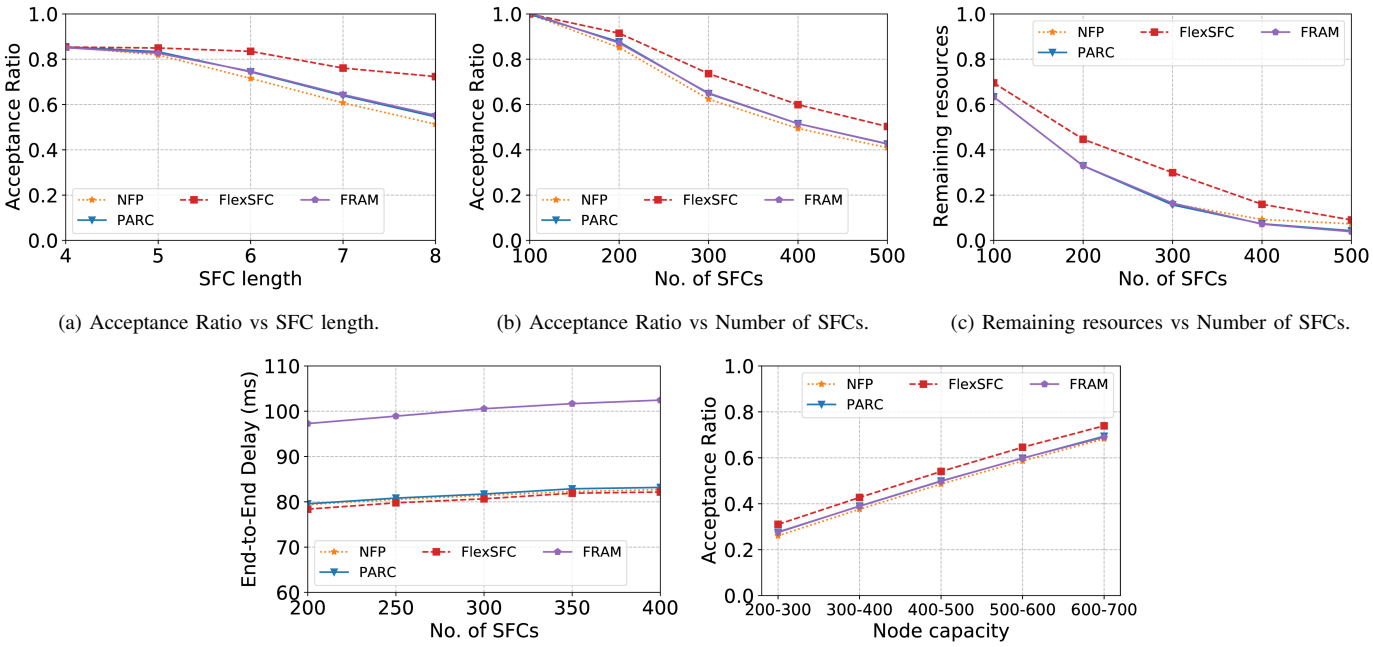
Acceptance Ratio vs Chain length: Fig. 5a presents the average rate of accepted SFC requests achieved, where the number of SFCs is fixed as 250. It can be observed that, as the SFC length increases, the acceptance ratio decreases for all the algorithms, but *FlexSFC* always achieves better acceptance ratio than *NFP*, *PARC* and *FRAM* approaches. It is because of *FlexSFC* scales down the resources of the parallelized VNFs and the resources saved are used to accommodate other SFCs. *FRAM* also follows the scaling approach, but it does not take parallelism into account. Thus, *FlexSFC* achieves a higher average acceptance ratio, it accepts up to 41%, 33%, and 31% more SFC requests over *NFP*, *PARC*, and *FRAM*, respectively.

Acceptance Ratio vs Number of SFCs: Fig. 5b depicts the average rate of accepted SFCs with respect to the total number of SFCs, which varies from 100 to 500 and has a fixed SFC length of 6. The results show that the acceptance ratio decreases as the number of SFCs increases, however, *FlexSFC* accepts up to 23%, 18%, and 18% more SFC requests over *NFP*, *PARC* and *FRAM*, respectively.

Resource utilization over varying number of SFCs: Fig. 5c shows the percentage of remaining resources for the same set of

Table I: Simulation Parameters

Parameter	Range	Parameter	Range
Length of SFC requests	4-8	Link delay	5-10 ms
Resources at each Node	200-700	Processing delay	1-20 ms
End-to-End latency	110-130 ms	Number of SFCs	100-1000
Requested bandwidth	100-300 Mbps	Link bandwidth	5-10 Gbps



(a) Acceptance Ratio vs SFC length.

(b) Acceptance Ratio vs Number of SFCs.

(c) Remaining resources vs Number of SFCs.

(d) Average end-to-end delay vs Number of SFCs.

(e) Acceptance Ratio vs Node Capacity.

Figure 5: Comparison of *FlexSFC* with *NFP*, *PARC*, and *FRAM* using USNET topology.

SFCs considered in Fig. 5b. Fig. 5c shows that *NFP* has large number of remaining resources as it tries to accommodate the entire SFC in a single server, while *FRAM*, *PARC*, and FlexSFC consume almost all the resources as the number of SFC requests increases. As shown in Fig. 5c, *FlexSFC* saves around $1.3\times$ of the resources when the number of SFCs are 200. For the same requests, *FlexSFC* accepts more SFCs than other approaches, as shown in Fig. 5b.

Average end-to-end delay with varying number of SFCs: Fig. 5d shows the average end-to-end delay taken by each SFC with respect to number of SFCs which varies from 200 to 400 and SFC length is fixed at 6. The results show that *NFP* deploys SFCs with less average end-to-end delays as it places entire SFC in the same server and the average end-to-end delay incurred by *FlexSFC* is also close to *NFP*. *FRAM* shows higher end-to-end delay because it does not consider SFC parallelization.

Results with varying node capacity: Fig. 5e shows the impact of node capacity on SFC acceptance ratio with respect to total resource available at each node which varies from 200 to 700. The results show that as the capacity of node increases, the acceptance ratio for all algorithms increases, but *FlexSFC* shows better acceptance ratio as compared to *NFP*, *PARC*, and *FRAM*. It is because of *FlexSFC* saves a significant amount of resources by scaling down resources of parallelized VNFs. *FlexSFC* accepts up to 19%, 13% and 12% more SFC requests over *NFP*, *PARC*, and *FRAM*, respectively.

VI. Conclusions

In this work, we proposed *FlexSFC* to determine the optimal SFC placement with minimizing resource consumption while providing delay assurance. Results show that *FlexSFC* guarantees the end-to-end delay requirement with better resource utilization than the state-of-the-art approaches, reducing up to $1.3\times$ of resource consumption and increasing the acceptance rate by 40%. In future, we will extend this work by proposing a strategy to allocate resources in dynamic traffic environments.

References

[1] Chen Sun, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu. Nfp: Enabling network function parallelism in nfv. In *Proc. of ACM SIGCOMM*, 2017.

[2] Yang Zhang, Bilal Anwer, Vijay Gopalakrishnan, Bo Han, Joshua Reich, Aman Shaikh, and Zhi-Li Zhang. Parabox: Exploiting parallelism for virtual network functions in service chaining.

[3] Sihao Xie, Junte Ma, and Jin Zhao. Flexchain: Bridging parallelism and placement for service function chains. *IEEE Transactions on Network and Service Management*, 18(1):195–208, 2020.

[4] Jun Cai, Zhongwei Huang, Liping Liao, Jianzhen Luo, and Wai-Xi Liu. Appm: Adaptive parallel processing mechanism for service function chains. *IEEE Transactions on Network and Service Management*, 18(2):1540–1555, 2021.

[5] Danyang Zheng, Gangxiang Shen, Xiaojun Cao, and Biswanath Mukherjee. Towards optimal parallelism-aware service chaining and embedding. *IEEE Transactions on Network and Service Management*, 2022.

[6] Tung-Wei Kuo, Bang-Heng Liou, Kate Ching-Ju Lin, and Ming-Jer Tsai. Deploying chains of virtual network functions: On the relation between link and server usage. *IEEE/ACM Transactions on Networking (TON)*, 26(4):1562–1576, 2018.

[7] Hao Yu, Tarik Taleb, and Jiawei Zhang. Deterministic service function chaining over beyond 5g edge fabric. In *Proc. of IEEE GLOBECOM*, 2021.

[8] Abdelhamid Alleg, Toufik Ahmed, Mohamed Mosbah, Roberto Riggio, and Raouf Boutaba. Delay-aware vnf placement and chaining based on a flexible resource allocation approach. In *Proc. of IEEE CNSM*, 2017.

[9] Jun Cai, Zhongwei Huang, Jianzhen Luo, Yan Liu, Huimin Zhao, and Liping Liao. Composing and deploying parallelized service function chains. *Journal of Network and Computer Applications*, 163, 2020.