At the Edge? Wait no More: Immediate Placement of Time-Critical SFCs with VNF Sharing

Amir Mohamad*[§] and Hossam S. Hassanein*
*Queen's University, Canada, [§]Cairo University, Egypt
Email: a.mohamad@queensu.ca, hossam@cs.queensu.ca

2022 IEEE 8th International Conference on Network Softwarization (NetSoft) | 978-1-6654-0694-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/NetSoft54395.2022.9844068

***Abstract*—The increasing demand on real-time and time-critical applications such as augmented reality, virtual reality, collision avoidance and industrial IoT, is fuelled by the low-latency promised by next-generation mobile networks (5G). Time-critical applications and services are real-time software whose failure could result in catastrophic consequences such as fatalities, damage to property, even financial losses. Edge computing is the main enabler of 5G ultra-low latency use cases. Edge resources are limited compared to the abundant cloud computing resources. As such, provisioning time-critical applications at the edge is more challenging and demanding. Even though virtual network function (VNF) sharing improves the utilization of the service providers' resources, service requests -including time-critical ones- can still be rejected due to insufficient resources. This paper proposes IPTSV, an immediate placement scheme for time-critical services with VNF sharing. The proposed scheme prioritizes time-critical premium (Pr) services over best-effort (BE) services. In cases when no resources are available for Pr services, a preemption mechanism preempts resources for the Pr service, by deporting one or more deployed BE services. The experimental results show that IPTSV can reduce the Pr services rejection rate to $\sim 0\%$, while minimizing the disturbance that BE services might witness such as prolonged waiting and turn-around times.**

***Index Terms*—Edge, SFC, NFV, VNF**

I. Introduction

Recently, there is a growing interest in edge computing both on the industry and academia fronts. Communication service providers (CSPs) are capitalizing on edge computing to host their own services, core and radio access network (RAN) disaggregated virtualized network functions (VNFs), in addition to third-party services such as over-the-top (OTT) services. The edge computing market is projected to grow from $36.5 billion in 2021 to $87.3 billion by 2026, at a growth rate of 19% during the forecast period [1].

With the agility that network function virtualization (NFV) brings, CSPs can now provision functions and services whenever they need over NFV infrastructure (NFVI) [2]. To do so, unlike the middle boxes era, a placement decision is needed to determine which physical node/server will host each VNF. Since the introduction of NFV in 2012, there has been a huge body of research addressing VNF placement and resource allocation. Most enterprise and network services consist of component functions/VNFs that are stitched together in a specific order to form service function chains (SFCs). As shown in Figure 1 (b), SFCs may have common VNFs, for example sfc_1 and sfc_2 have V_3 in common and V_4 is common between sfc_2 and sfc_3.

The majority of emerging 5G use cases are time-critical in nature, such as real-time media (augmented reality (AR) and virtual reality (VR)), industrial control, remote control, and mobility automation [3]. Time-critical, henceforth premium (Pr), services and applications are a class of software that have stringent time constraints and a service would fail if such constraints were not met [4]–[6]. Catastrophic consequences might follow as a result of service failure, for example, a collision warning service failure might result in more collisions and more fatalities. Edge computing is a distributed version of cloud and its resources are limited compared to cloud's. The demand generated by time-critical applications necessitates efficient utilization of edge resources. With the stringent time constraints of time-critical applications and services, there must be a mechanism by which time-critical service requests are immediately satisfied.

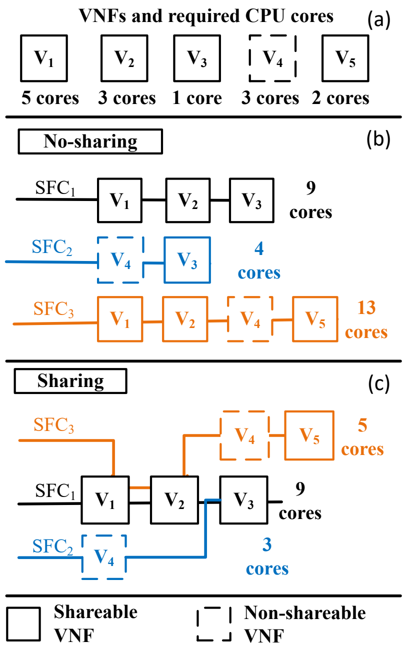


Fig. 1. Compute resources savings when sharing VNFs among SFCs

To address these two needs, efficient utilization of edge resources and immediate satisfaction of Pr service/SFC requests, this paper proposes IPTSV, an immediate placement scheme of

This research is supported by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant number: RGPIN-2019-05667.

time-critical SFCs with VNF sharing. This is accomplished by taking advantage of the common VNFs across SFCs and the operations dynamics that might leave some deployed VNFs underutilized. The IPTSV satisfies sequential SFC requests by sharing underutilized shareable VNFs. With limited resource at the edge, when a Pr SFC request cannot be satisfied, a preemption criterion is used to stop and deport some or all of the deployed best-effort (BE) lower priority SFCs to release resources and successfully deploy the Pr SFC. The preemption criterion is designed to address the trade-off between releasing the resources to deploy a Pr SFC and minimizing the number of disturbed BE SFCs.

The main contributions of this paper are:

- Introduction of preemptive placement scheme that reduces Pr SFC requests rejection rate to near zero.
- Defining the baseline performance of preemption-based service placement.
- Recommending which preemption criterion to use given the service domain context and provider's policies and priorities.

The remainder of this paper is structured as follows. Section II covers related work. Proposed time-critical SFC placement, system model, and problem formulation are detailed in Section III. In Section IV we detail the simulation framework. Performance evaluation and results analysis are covered in Section V. Conclusions and future work is presented in Section VI.

II. Related Work

A. VNF/SFC Placement Decision Levels

The placement of SFC is challenging due to orderliness and other constraints that a placement scheme has to consider compared to the placement of a single VNF. Unlike interdependent VNFs in an SFC, microservices components in a cloud-native application need not be deployed all at once and the components could be pro-actively deployed, which is extremely beneficial in resource-limited environments like edge computing [7]. Unlike the work in [8] that treats SFCs as a compute task that has a sequence of executions, to start processing traffic flow, all VNFs of any SFC must be deployed and running.

Depending on the architecture and scale of the NFV environment, the placement decision is made at different levels. Some techniques propose the placement of VNFs/SFCs at the server level, i.e., which server hosts each VNF, such as the proposed placement techniques in [9]–[12]. Authors in [13] proposed Octans in which they address the SFC placement at a CPU core level in many-core systems. Authors in [14] proposed Finedge which allocates resources at the CPU core time/share level and it dynamically monitors and reacts to the possible performance degradation.

B. VNF Sharing

"*VNF sharing*" is one of the techniques used to reduce the cost and efficiently utilize resources. For example, as shown in Figure 1, the required resources to satisfy sfc_1, sfc_2 and sfc_3 are 26 $cpu\,cores$ compared to only 17 $cores$ when VNF sharing is used. Unlike some surveyed VNF sharing papers which consider all VNFs are shareable, in this example having V_4 as non-shareable, the VNF sharing is still able to use 35% less resources.

We remark that in the literature more than one term is used to refer to *VNF sharing* concept. For example, the authors of [15] and [16] used the term *"multi-tenancy"* and *"VNF merging"*, in [17] authors used *"VNF reuse"* (VM reuse), task and request scheduling are used in [7] and [18], and the most common term was *VNF sharing*, used in [19]–[23].

There is an increasing interest in *VNF sharing* among CSPs and OTT service providers. For example, deploying evolved packet core (EPC) VNFs on public cloud used to be a deserted and excluded idea, however, there is increasing deployment of EPC VNFs on the public cloud. Moreover, sharing non-security-critical VNFs such as mobility management across end-to-end 5G slices is getting attention [20]. In [19], authors proposed sharing the same CDN cache VNF (vCache) among ISPs with common infrastructure. Consequently, *VNF sharing* can play an imperative role in reducing the cost-of-service provisioning by efficiently utilizing resource-limited edge environments. Excluding security reasons, not sharing VNFs may result in inefficient resource utilization because of the idle/redundant capacity that is never used and resource fragmentation [22].

The work in [11], proposes sharing VNF among SFC flows and specifies predefined number of flows that a VNF can serve, which ignores the operations dynamics and may leave VNFs over-/under-utilized. The authors of [17] mix the concepts of infrastructure VM and *VNF sharing*, yet they assume that a VM can only host one VNF. However, the VNF container, VM or container, should not be treated as an infrastructure asset, it should be an ephemeral component that is instantiated, deployed, replicated, and terminated.

C. Priority-based Placement

In the literature of priority-based SFC and VNF placement, the concept and handling of priority is diverse. The work in [23] utilizes priority that is dynamically assigned to SFCs, VNFs or flows, depending on the situation, rather than a predetermined priority before deployment. In [24], the priority is determined after receiving the SFC request and is based on the resources required by SFC, the more resources required the higher the priority. While these priority assignment techniques may sound practical, we believe that the SFC request priority should be the same for all SFC's VNFs, the priority should not change and should be known before satisfying the SFC request. For example, for a time-critical SFC request the same higher priority should be assigned to all its VNFs before arriving at an orchestrator, the priority should not change and should be agnostic to required resources.

III. System Model and Problem Formulation

In NFV-based service domain, some services, of both CSPs and third-party service providers come in different quality of service (QoS) categories. Indeed, there could be more than two service categories, however, for simplicity, in this paper two service categories have been chosen. Pr service that is provisioned with highest expected load, not oversubscribed, and served with dedicated high-priority queues; and BE service is sent and queued with lower priority.

Due to operations dynamics, traffic processed by SFC's VNFs vary and VNFs can be underutilized at times. *VNF sharing*-based SFC placement scheme takes advantage of operations dynamics and shareable VNFs to enhance resource utilization, reduce SFC deployment cost and rejection rate. To satisfy a new SFC request, *VNF sharing*-based placement scheme scans deployed VNFs for underutilized similar VNF(s) that can manage the expected load of the SFC request at hand. A new VNF is only instantiated in cases where: the VNF is non-shareable, there are no similar previously deployed VNFs, or a similar VNF(s) are deployed but fully utilized.

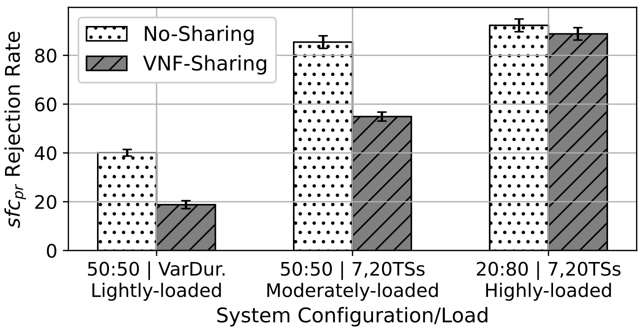


Fig. 2. Rejection rate of Pr SFC requests (sfc_{pr}) of VNF-sharing-based scheme vs no-sharing-based scheme for different system loads. (Experiment duration is 200 TSs, arrival rate λ = 2 requests/TS. VarDur.: duration of SFCs is variable, the average is 10 TSs. 7-20: duration is fixed, sfc_{pr}=7 TSs & sfc_{be}=20 TSs). Results are based on the same setup as in Sec. IV.

Despite using *VNF sharing*, there is a possibility that a received Pr SFC request cannot be satisfied due to insufficient resources. Based on results of our *VNF sharing*-based placement scheme in [21] (used same setup as in Section IV), the rejection rate of Pr SFCs is unavoidable and concerning. As shown in Figure 2, the best-case scenario is 'lightly-loaded' system. Utilizing *VNF sharing* resulted in a 50% reduction in Pr SFCs rejection rate; however, the rejection rate of the *VNF sharing*-based placement scheme is still about 19%. It is even worse for higher system loads, where there are either longer duration BE SFCs or more BE SFCs. These rejections represent unsatisfied customers and lost revenue for CSPs.

In response to this, we propose immediate placement of time-critical SFCs with VNF sharing (IPTSV) to help CSPs manage their resource utilization in an efficient manner and seize revenue opportunities. In situations where resources are not available to satisfy Pr SFCs, a criterion should be in-place to preempt resources for Pr SFC by deporting lower-priority BE SFC(s). The preemption criterion should strive to balance between, immediately satisfying Pr SFCs and minimizing number of disturbed BE SFCs, by preemption. Priority-aware preemptive-scheduling is not a new idea, it has been used extensively, especially in the context of process scheduling for CPU. To the best of our knowledge, priority-aware preemptive-scheduling was never been used in the context of SFC placement with *VNF sharing* at the resource-limited edge environment.

In our system, Pr SFCs can be in one of these states: received, rejected, running, or completed. Due to their lower priority, BE SFC can be in states: received, running, pending redeployment, or completed. The states and actions that trigger state changes of SFCs are detailed in Figure 3.

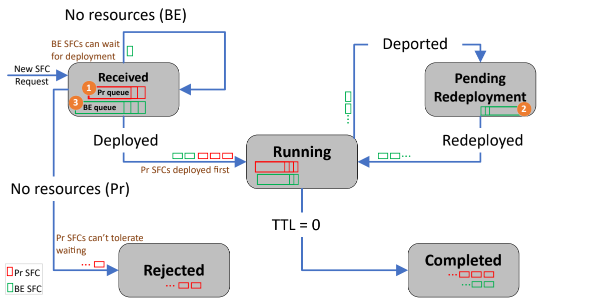


Fig. 3. States an SFC request can take in IPTSV and order of priority and deployment of queues ($1 \rightarrow 2 \rightarrow 3$).

A. System Model

The list of on-boarded VNFs V is from where SFC's VNFs are selected. Each VNF $v_i \in V$ has resource requirements and once these requirements are provided, the VNF is expected to process a maximum flow of traffic $F_{max}(v_i)$. $S(v_i)$ is a flag to determine whether VNF v_i is shareable or non-shareable. Once selected in sfc_j, the actual inflow and outflow (subject to change due to operations dynamics) of VNFs should be declared. The substrate network is represented as a graph $G(N, E)$, where N is the set of nodes and E is the set of links. Each node $n \in N$ has compute resources, *cpu cores* and $memory$, and each link $e \in E$ has bandwidth capacity bw_c and propagation delay Del. The topology of substrate network is determined by a configurable connectivity matrix. Table I lists detailed description of substrate network and SFC parameters.

B. Problem Formulation

The IPTSV scheme, in Algorithm 1, hinges critically on two main components, the placement algorithm, and the preemption criterion. The different scores used in the preemption criteria are detailed in Section III-B2. The *VNF sharing*-based placement algorithm is modelled as an integer quadratically-constrained program (IQCP) with binary decision variables.

The IPTSV scheme manages sequential SFC requests arriving at each time slot (TS). First, IPTSV looks for deployed SFCs in list $Run_{pr|be}$ with time-to-live (TTL) value, in TSs, equal to zero to terminate and release the resources, otherwise, it decreases TTL value by one. Second, IPTSV considers requests in received Pr queue (Rec_{pr}). After trying to satisfy Pr SFC requests, IPTSV attempts to satisfy the BE SFCs in the Pen_{be} queue, which holds the BE SFCs that were previously deported to accommodate a Pr SFC and were not successfully

TABLE I. System Parameters Description

Parameter	Description
$cpu_c(n)$	CPU capacity in cores of node $n \in N, \sim U[8,64]$
$ram_c(n)$	RAM capacity in GBs of node $n \in N, \sim U[16,128]$
$cpu_{av}(n)$	Available CPU cores at node $n \in N$
$ram_{av}(n)$	Available RAM GBs at node $n \in N$
$e_{nn'}$	A link exists from node n to node n', $n, n' \in N$
$bw_c(e_{nn'})$	BW capacity in Gbps of link $e_{nn'}, \sim U[2,5]$
$bw_{av}(e_{nn'})$	Available BW at link $e_{nn'}$
$Del(e_{nn'})$	Propagation delay in μsec of link $e_{nn'}, \sim U[0.5,4]$
V	Set of available/on-boarded VNFs
v_i	Is a VNF, where $v_i \in V$
$cpu(v_i)$	CPU cores required for VNF $v_i \in V, \sim U[2,8]$
$ram(v_i)$	RAM GBs required for VNF $v_i \in V, \sim U[4,16]$
$F_{max}(v_i)$	Maximum inflow VNF v_i can handle, function of $cpu(v_i)$ and $ram(v_i)$
$S(v_i)$	Flag to indicate VNF v_i is shareable
$Drop(v_i)$	Flag to indicate that VNF v_i drops/compresses inflow
sfc_j	SFC request j
$\|sfc_j\|$	Number of VNFs in sfc_j, , $\sim U[4,7]$
v_i^j	The i^{th} VNF of sfc_j
$F_{in}(v_i^j)$	Actual inflow that VNF v_i will be serving
$F_{out}(v_i^j)$	Outflow VNF v will produce
$Del(sfc_j)$	Maximum end-to-end delay of $sfc_j = (\|sfc_j\| - 1) * AvgLinkDelay$, where $AvgLinkDelay = \frac{1}{\|E\|}\sum_{n=1}^{\|N\|}\sum_{n'=1}^{\|N\|} Del(e_{nn'})$
$V_{diff}(sfc_{j,k})$	number of VNFs in sfc_j not in sfc_k
$Gst(sfc_j)$	total number SFCs hosted by sfc_j

Algorithm 1: IPTSV

```
// netModel: network Model, No.TSs: Simulation time in time-slots
Input  : netModel, No.TSs
Init.  : $Rec_{pr|be}$, $Run_{pr|be}$, $Pen_{be}$, $Rej_{pr}$, $Com_{pr|be}$
Output: Different queues
for $i \leftarrow 1$ to No.TSs do
    $Rec_{pr} \leftarrow received\,Pr\,requests$
    $Rec_{BE} \leftarrow received\,BE\,requests$
    // Update TTL of running SFCs
    foreach $sfc_{pr|be}$ in $Run_{pr|be}$ do
        if ttl($sfc_{pr|be}$) $= 0$ then
            $Com_{pr|be} \leftarrow sfc_{pr|be}$
        else decTTL($sfc_{pr|be}$)
    foreach $sfc_{pr}$ in $Rec_{pr}$ do
        // Using IQCP & Gurobi solver
        sol $\leftarrow$satisfy($sfc_{pr}$,netModel)
        if sol $\neq \emptyset$ then
            deploy($sfc_{pr}$,netModel)
            $Run_{pr} \leftarrow sfc_{pr}$
        else // Check preemptCPU algorithm for details
            // $sfc_{pr}$ is only used with 'similar' preemption criterion
            sol,$Pen_{be}$ $\leftarrow$preemptCPU(criterion,$Run_{be}$)
            if sol $\neq \emptyset$ then
                deploy($sfc_{pr}$,netModel)
                $Run_{pr} \leftarrow sfc_{pr}$
            else $Rej_{pr} \leftarrow sfc_{pr}$
    foreach $sfc_{be}$ in $Pen_{be}$ do
        sol $\leftarrow$satisfy($sfc_{be}$,netModel)
        if sol $\neq \emptyset$ then
            deploy($sfc_{be}$,netModel)
            $Run_{be} \leftarrow sfc_{be}$
        // else $sfc$ stays in $Pen_{be}$
    foreach $sfc_{be}$ in $Rec_{be}$ do
        sol $\leftarrow$satisfy($sfc_{be}$,netModel)
        if sol $\neq \emptyset$ then
            deploy($sfc_{be}$,netModel)
            $Run_{be} \leftarrow sfc_{be}$
        // else $sfc$ stays in $Rec_{be}$
```

redeployed. Finally, IPTSV addresses those SFC requests in the received queue Rec_{be}. There is no Pen_{pr} queue because Pr SFCs cannot tolerate being queued in a received waiting queue (Rec_{pr}) beyond the TS they were received in, or being deported and waiting for redeployment. There is no Rej_{be} list, as we assume that BE queues are infinite, and the BE SFCs are to stay in queues waiting for deployment or redeployment.

*1) **Placement Algorithm**:* With SFC request sfc_j consisting of VNFs $v_i, i \in [1 - |sfc_j|]$, the decision variables are: X_{in}^j, meaning a new instance of VNF v_i belonging to sfc_j is to be placed at node n; and R_{in}^j means that VNF v_i of sfc_j is to share and become the guest of a deployed underutilized VNF of the same type at node n. Queues, decision variables, and parameter descriptions are in Table II.

*a) **Objective Function**:* The objective is to select the placement that minimizes the overall cost, hence optimize resource utilization. The objective function in equation (1) is formulated to prioritize sharing over deploying new VNF instances.

$$min \sum_{i=1}^{|sfc_j|} \sum_{n \in N} [cpu(v_i^j)U_{cpu}^c(n) + ram(v_i^j)U_{ram}^c(n)]X_{in}^j + F_{out}(v_i^j)U_c bw[X_{in}^j + R_{in}^j] \quad (1)$$

*b) **Constraints**:* A feasible solution must have each VNF of SFC assigned only once to a substrate node, where each VNF is realised by a new instance, or by sharing the free capacity of a deployed VNF, equation (2). If sharing is the decision, a shareable VNF of the same type must have been

TABLE II. Queues, Decision Variables and Constants

Variable/Queue	Description
Rec_{pr}	A queue holding new Pr SFCs until deployed in the same TS
Rec_{be}	A queue holding new BE SFCs until deployed
Rej_{pr}	A list of rejected Pr SFCs
Pen_{be}	A queue holding deported BE SFCs to be redeployed
$Run_{pr\|be}$	A list of deployed Pr or BE SFCs
$Com_{pr\|be}$	A list of finished Pr or BE SFCs
X^j_{in}	Binary decision for placing VNF v_i of sfc_j at node n
R^j_{in}	Binary decision for sharing the flow of VNF v_i of sfc_j with already deployed VNF of same type at node n
D^i_n	VNF of same type as v_i already deployed at node n
$F_{av}(D^i_n)$	Available unused flow of v_i at node n
$U^c_{cpu}(n)$	Unit cost of cpu at node n
$U^c_{ram}(n)$	Unit cost of ram at node n
$U_c(bw)$	Unit cost of bw at all links

deployed, equation (3), and its free capacity is a sufficient amount for SFC's VNF inflow, equation (4). If a new instance is to be deployed, X^j_{in} to be valid, substrate node n must have the required resources i.e., cpu and ram, equations (5 & 6). Equations (7) and (8) are to ensure continuity of both SFC's VNFs and the substrate nodes hosting them, and make sure that available bandwidth in the physical link $e_{nn'}$ is enough for the outflow VNF v^j_i. Finally, equation (9) is to guarantee that the quality-of-services (QoS) requirements (maximum end-to-end latency) of sfc_j is satisfied. We assume that both transmission and processing delays are negligible and the delay of each placement solution is the summation of propagation delay of solution's links.

$$\sum_{n\in N} X^j_{in} + R^j_{in} = 1 \quad , \quad \forall i \in [1-|sfc_j|] \tag{2}$$

$$\sum_{n\in N} X^j_{in} + D^j_i\ S(v_i)\ R^j_{in} = 1, \quad \forall i \in [1-|sfc_j|] \tag{3}$$

$$F_{in}(v^j_i)\ R^j_{in} \le F_{av}(D^i_n),\ \forall n \in N\ \&\ \forall i \in [1-|sfc_j|] \tag{4}$$

$$\sum_{i=1}^{|sfc_j|} cpu(v^j_i)\ X^j_{in} \le cpu_{av}(n), \quad \forall n \in N \tag{5}$$

$$\sum_{i=1}^{|sfc_j|} ram(v^j_i)\ X^j_{in} \le ram_{av}(n), \quad \forall n \in N \tag{6}$$

$$e_{nn'}\,(X^j_{in} + R^j_{in})(X^j_{(i+1)n'} + R^j_{(i+1)n'}) = 1 \\ \forall n, n' \in N\ \&\ \forall i \in [1-(|sfc_j|-1)] \tag{7}$$

$$\sum_{n\in N}\sum_{n'\in N} F_{out}(v^j_i)(X^j_{in} + R^j_{in})(X^j_{(i+1)n'} + R^j_{(i+1)n'}) \\ \le bw_{av}(e_{nn'}), \quad \forall i \in [1-(|sfc_j|-1)] \tag{8}$$

$$\sum_{i=1}^{|sfc_j|-1}\sum_{n\in N}\sum_{n'\in N} Del(e_{nn'})\,(X^j_{in} + R^j_{in}) \\ (X^j_{(i+1)n'} + R^j_{(i+1)n'}) \le Del(sfc_j) \tag{9}$$

2) ***Preemption Criteria:*** In a conventional preemptive resources allocation algorithm, deporting a running process/service always releases a fixed number of resources. With *VNF sharing*, deporting a running SFC does not necessarily release the exact same number of resources that SFC is utilizing. This is why we propose to study different number of scoring and preemption criteria to produce a recommendation for the best criterion for certain contexts.

As in Algorithm 2, IPTSV deports BE SFCs one at a time from a list sorted according to the calculated scores. The simplest preemption criterion that IPTSV could use is '*All*,' in which all running BE SFCs are deported to release resources for the Pr SFC at hand. This criterion represents a baseline performance and will be used as a benchmark to evaluate other criteria. Some of the deported BE SFCs will be successfully redeployed, which means they were gratuitously deported, while others cannot be deployed and will be put in a redeployment-pending queue Pen_{be}. On one hand, the '*All*' criterion is very simple and does not require executing the placement algorithm for the Pr SFC more than once. On the other hand, it unnecessarily disturbs all running BE SFCs and consequently, to redeploy BE SFCs, we have to execute the placement algorithm as many times as the number of deported BE SFCs.

Algorithm 2: preemptCPU

```
Input  : netModel, criterion, $sfc_{pr}$, $Run_{be}$
Output: sol, $Pen_{be}$
1  if criterion $\neq$ $Random$ then
       // $sfc_{pr}$ is needed for 'similar' criterion
       // sort() sorts 'as.' $\uparrow$ or 'dec.' $\downarrow$ based-on given criterion
2      $Run_{be} \leftarrow$ sort(score($Run_{be}$,criterion,
         $sfc_{pr}$),[$as|dec$])
3  sol $\leftarrow \emptyset$
4  while sol $= \emptyset$ and len($Run_{be}$) $\neq 0$ do
5      if criterion $=$ $Random$ then
           // $Run_{be}$ isn't sorted
6          $sfc_{be} \leftarrow$ getRandSfc($Run_{be}$)
7      else // $Run_{be}$ is sorted
8          $sfc_{be} \leftarrow Run_{be}[0]$
9      deport($sfc_{be}$)
10     $Pen_{be} \leftarrow sfc_{be}$
11     sol $\leftarrow$ satisfy($sfc_{pr}$,netModel)
12 return sol, $Pen_{be}$
```

To solve the negative effects of the '*All*' criterion, we propose alternate preemption criteria. The main goal is to minimize both number of disturbed BE SFCs (reduce gratuitously deported SFCs) and number of placement algorithm executions. First, a score is calculated per BE SFC then the list of scores is sorted in a descending or ascending order depending on the criterion. Finally, BE SFCs are deported one at a time until the Pr SFC is successfully deployed. The best-case scenario is to deport only one BE SFC and the worst-case scenario is to deport all BE SFCs. In the latter scenario, there is a chance that the Pr SFC cannot be deployed, as the resources utilized by BE SFCs were not enough for the Pr SFC, or there were no deployed BE SFCs.

The successful deployment of SFC requests is depending on the availability of resources, especially the $cpu\ cores$. That is why, the score calculation should either directly or indirectly involve $cpu\ cores$ utilized by deployed BE SFCs and can use $cpu\ cores$ required by the new Pr SFC. The scores we decided to use are: SFC-Length (number of VNFs in SFC), SFC-CPU (number of $cpu\ cores$ utilized by SFC), SFC&Node-CPU (number of $cpu\ cores$ utilized by SFC plus the number of free $cores$ at nodes hosting SFC's VNFs), Similar (similarity measure between deployed BE SFCs and the new Pr SFC), and Random (no scores calculated, the list of deployed BE SFCs used as-is).

The *'longer-first'* and *'shorter-first'* criteria sort the list of length scores in descending and ascending order, respectively. The *'SFC-CPU-first'* and *'SFC&Node-CPU-first'* criteria use descending sorted CPU scores lists. The *'most-similar-first'* criterion is the only one that depends on the new Pr SFC in calculating the similarity score. The premise here is to search for the most similar BE SFC to the new Pr SFC, which should minimize the number of deported BE SFCs. As in equation (10), the similarity score is the inverse of difference score which includes: number of different VNFs, Pr-length minus BE-length, Pr-max-outflow minus BE-max-outflow, difference in required CPU cores, and total number of SFCs that BE SFC is hosting. Each difference term in equation (10) is normalized against its peers calculated for all BE SFCs before calculating the similarity score.

$$SIM(sfc_{pr}, sfc_{be}) = 1/\{V_{diff}(sfc_{pr,be}) + (|sfc_{pr}| - |sfc_{be}|) + [max(f_{out}(v_i^{pr})) - max(f_{out}(v_i^{be}))] + [\sum_{i=1}^{|sfc_{pr}|} cpu(v_i^{pr}) - \sum_{i=1}^{|sfc_{be}|} cpu(v_i^{be}) + Gst(sfc_{be})\} \quad (10)$$

The $deport(sfc_{be})$ procedure, in Algorithm 2 line 9, is accomplished with VNF sharing considered. In VNF-sharing, shareable VNFs are either host or guest. For example, in Figure 4, VNF V_2 of sfc_1 is a host VNF sharing its unused capacity with two similar guest VNFs belonging to sfc_3 and sfc_4. In cases where a guest VNF's SFC is to be deported, the shared capacity is simply retuned to the host VNF. If the host VNF's SFC is to be deported, then a guest VNF must be promoted to assume the host role. The simplest scenario is when there is only one guest VNF, it will take the host role automatically. If more than one guest VNFs exist, the one that has the highest TTL will be promoted, to avoid the overhead of frequent promotions if we have selected a shorter-living VNF.

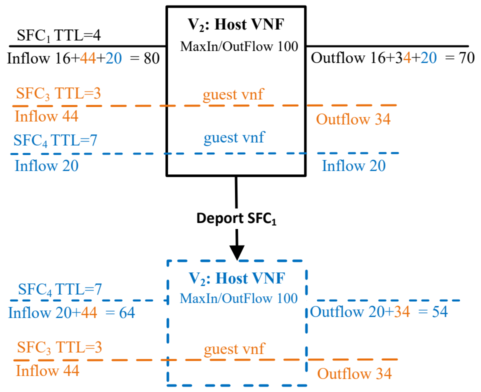


Fig. 4. Promoting guest VNF to act as a host, part of sfc_{be} deportion

IV. Simulation Framework

Because the development of edge computing and MEC is relatively new, there are no edge/MEC demand and workload traces that are publicly available and sufficiently suitable for our system setup [7], [25]. Therefore, we decided to synthetically generate SFC requests per each TS. The arrival of SFC requests per TS follows a Poisson distribution with average rate $\lambda = 2$.

We developed a Java-based simulation environment, in which we generate substrate network model, synthetically generate demand by creating SFC requests at time slots, execute the placement decisions, and track SFCs different state/queue transitions. SFC length is drawn from a uniform distribution $|sfc_j| \sim U[4, 7]$ [26]. The service time, i.e., SFC duration in TSs, is fixed, where $sfc_{pr} = 7$ and $sfc_{be} \in \{5, 20\}$. If SFC duration is to be variable, it is randomly sampled from a uniform distribution $U[5, 18]$. The ratio of Pr to BE SFC requests are either *'50:50'* or *'20:80,'* and the order of arrival is randomly shuffled.

In our simulations, for all experiments, we used the same NSFNET network model and the same topology that has 13 nodes and 32 directional links. With the simulation time set to 200 TSs, we generated around 415 SFCs, taking care of the number of SFC requests per TS; type of VNFs of each SFC; actual inflow and outflow of each VNF; and the cap end-to-end delay of each SFC as the QoS requirement. This process was repeated ten times and generated data are saved in files and used to experiment with different preemption criteria. The list of on-boarded VNFs contains 16 VNFs of different flavors and requirements, where 60% of VNFs are shareable. The IQCP model is solved using the Gurobi solver [27].

V. Performance Evaluation

A. Evaluation Metrics

Similar to other SFC placement schemes, IPTSV uses metrics such as resource utilization, rejection rate, and percentage of SFCs waiting for deployment or pending for redeployment. In addition, the preemption related metrics are the following: the percentage of gratuitously deported BE SFCs; the average number of deported BE SFCs to satisfy one Pr SFC; the average number of deportations per BE SFC; the maximum

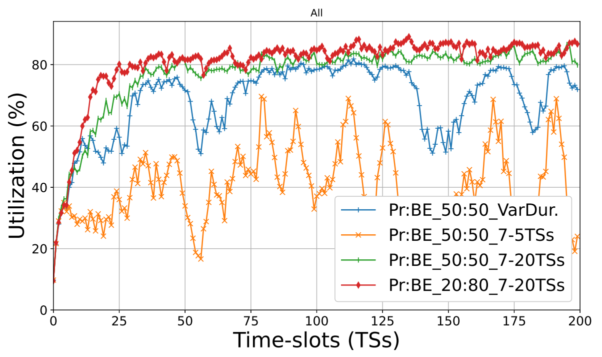


Fig. 5. Resource utilization for different system loads for '*All*' preemption criterion

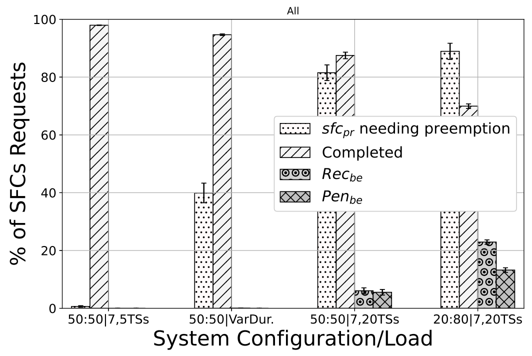


Fig. 6. End-of-simulation queue sizes for different system loads for '*All*' preemption criterion

and minimum number of deportations of one BE SFC; and the percentage of received Pr SFC requests that needed preemption to be satisfied. To assess the impact on deported BE SFCs, we use the average waiting time (AWT) and turn-around time (TAT). AWT is the time an SFC spends in the system not in the running state before completed and is calculated for SFCs in Rec_{be}, Pen_{be}, and Com_{be} queues and lists. The TAT is the total time spent from reception to completion and is calculated for and averaged over completed SFCs only. Due to the sensitivity to uncontrollable processes running in the background, we chose to use TS, instead of system time in milliseconds, as the unit to report AWT and TAT.

The purpose of these experiments is not to crown a winning preemption criterion, rather first, it is to demonstrate the plausibility of preemption in the context of time-critical SFC placement with VNF sharing (using the '*All*' criterion). Second, to study how successful other criteria are in addressing the side effects of the '*All*' criterion. Lastly, check if the almost overhead free 'Random' criterion's performance is comparable to the best criterion, and in which circumstances.

B. Numerical Results and Analysis

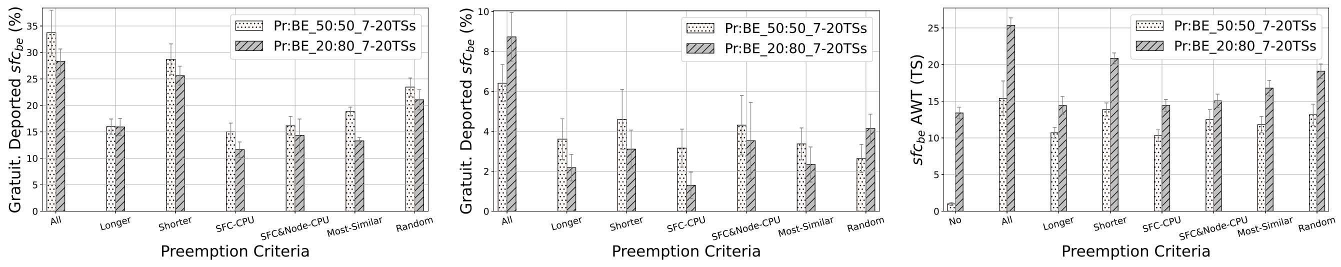
To ensure that we are deriving results and conclusions from a steady/stable system, we used different system loads, by varying the Pr:BE ratio and SFC duration in TSs. We experimented with the Pr:BE ratio of *'50:50'* and *'20:80,'* and for the SFC duration, we experimented with variable duration (average of 10 for both Pr and BE SFCs), Pr 7 and BE 5, 7 and 20 TSs. In this experiment, the preemption criterion used is the '*All*'. For this experiment we reported the utilization throughout the experiment duration (200 TSs).

As shown in Figure 5, the 'Pr:BE% 50:50|7-5 TSs' case sustains the least utilization, that is because the system never gets to the point where resources are used up. Deployed SFCs, especially BE SFCs complete their job quickly and release resources sooner. As we increase the duration of BE SFCs, more resources will be used up, and hence, the queues Rec_{be} and Pen_{be}, will start to build up and the rejection rate will start to increase.

The idea here is: the longer SFCs will have to stay in the system, the higher the rejection rate will be. As illustrated in Figure 6, the 'Pr:BE% 50:50|7-5 TSs' is a trivial case with almost no Pr SFCs needing preemption to be satisfied and the 'Pr:BE% 50:50| VarDuration' is where we begin to see a rise of Pr SFCs needing preemption. It is even more serious for the 'Pr:BE% 50:50|7-20 TSs' and 'Pr:BE% 20:80|7-20 TSs' cases. As such, and due to the many configurable knobs in our system, for the remaining experiments, we will be using the 'Pr:BE% 50:50|7-20 TSs' as a moderately-loaded system, and 'Pr:BE% 20:80|7-20 TSs' as a highly-loaded system.

To evaluate the performance of preemption criteria, we measured the number of deported BE SFCs to deploy one Pr SFC and the average number of deportations a BE SFC has to endure. As detailed in Table III, taking the '*All*' criterion as a reference, the *'SFC-CPU-first'* criterion is ahead in most measures. This is attributed to the sometimes misguided *'SFC&Node-CPU-first'* criterion when the number of utilized $cores$ by a BE SFC is low, but one of its VNFs' hosting node has a very high number of free $cores$. In such case, this SFC will climb to the top of the sorted list and will be deported first. The *'Longer-first'* criterion is not as good, since it depends on the length, in which longer SFCs do not necessarily utilize more $cpu\,cores$. Moreover, the longer the SFC, the higher the probability that more VNFs are either host or guest VNFs. In either case, deporting those VNFs, will not release any resources. The *'most-similar-first'* criterion is closely competing, even better in one column (average deportations/sfc_{be}), however, unlike other criteria, it recalculates scores of BE SFCs for every Pr SFC. Yet, the results does not parallel the burden of processing overhead.

Gratuitously deported BE SFCs are those SFCs deported to satisfy a Pr SFC and were successfully redeployed in the same TS. Figure 7a reports gratuitously deported SFCs as a percentage of all deported BE SFCs. The *'SFC-CPU-first'* criterion yields the best performance both in the moderately-loaded and highly-loaded systems. When deporting the SFC that utilizes the highest number of CPU cores, the probability is higher that this is the best-case scenario, i.e., deporting only one BE SFC or fewest number of BE SFCs. We are using the word *'probability'* since VNFs are shared and there is a chance that an SFC utilizing the highest number of $cpu\,cores$, but upon deportation, few to zero $cpu\,cores$ are released. The



(a) IPTSV (b) No-sharing (c) sfc_{be} AWT

Fig. 7. (a) Gratuitously deported BE SFCs using IPTSV, (b) Gratuitously deported BE SFCs (No-Sharing), and (c) AWT of sfc_{be}.

TABLE III. Preemption Performance (Red the is worst, Green is the best)

	Pr:BE 50:50,7-20 TSs		Pr:BE 20:80,7-20 TSs	
	Avg deported BE/Pr	*Avg deportations/BE*	*Avg deported BE/Pr*	*Avg deportations/BE*
All	**19.75**±0.33	**16.79**±0.56	**25.81**±0.17	**6.76**±0.28
Longer	**2.53**±0.11	**4.47**±0.24	**2.87**±0.14	**2.01**±0.1
Shorter	**4.55**±0.2	**6.2**±0.26	**5.66**±0.23	**2.97**±0.09
SFC-CPU	**1.94**±0.07	**2.59**±0.11	**2.37**±0.09	**1.35**±0.03
SFC&Node CPU	**2.11**±0.07	**2.57**±0.09	**2.42**±0.1	**1.40**±0.03
Similar	**2.17**±0.07	**2.57**±0.06	**2.59**±0.14	**1.41**±0.05
Random	**2.72**±0.07	**2.70**±0.1	**3.45**±0.15	**1.45**±0.03

reason being, (as explained in Section III-B2 and in Figure 4), when deporting a host VNF that has one or more guest VNFs, the $cpu\, cores$ will not be released. To prove this, we measured the same metric with *non-sharing*, shown in Figure 7b. The overall percentage of gratuitously deported BE SFCs significantly dropped about $79\% - 81\%$ in the *'50:50'* system and $69\% - 89\%$ in the *'20:80'* system. This significant drop is due to the absence of sharing and the guaranteed release of resources once an SFC is deported. Furthermore, the *non-sharing* version of *'Shorter-first'* is better than that of IPTSV version which is almost as bad as the *'All'* criterion.

The AWT of BE SFCs, reported per TS, increases almost linearly as time progresses and load increases. Results of *'No-preemption'* criterion is used as a reference lower-bound of BE SFC AWT. In the concluding results of the experiment shown in Figure 7c, the best, least, AWT is that of *'No'* and the worst as expected is that of *'All'* criterion. In both *'50:50'* and *'20:80'* systems, the *'Longer-first'* and *'SFC-CPU-first'* exchange best AWT. In *'20:80'* system, the AWT of *'No'* criterion started to increase as a result of having more BE SFCs staying 20 TSs, yet *'SFC-CPU-first'* criterion maintains consistent least AWT (second after the *'No'* criterion). Since the TAT is almost equal to SFC duration plus the AWT, we did not include average TAT of BE SFCs figure.

We formulated equation (11) as the preemption cost function. It is a function of j and k, where $j \in [0 - |Run_{be}|]$ is number of deported BE SFCs to satisfy one Pr SFC and $k \in [4 * |Run_{be}| - 7 * |Run_{be}|]$ is total number of VNFs in deported SFCs. The cost function has three components: c_1 the cost of lost revenue when deporting one BE SFC; c_2 cost of a single execution of placement algorithm; and c_3 cost of caching the state of a single VNF until redeployment. For simplicity, we used equal costs, $c_1 = c_2 = c_3 = 1$.

$$Cost(j,k) = \begin{cases} c_1.j + c_2(1+j) + c_3.k & ,\text{'}All\text{'}\, criterion \\ c_1.j + 2c_2.j + c_3.k & ,otherwise \end{cases} \tag{11}$$

As shown in Figure 8, the criteria that has the least preemption cost for *'50:50'* system are *'All'*, *'SFC-CPU-first'*, and *'Most-similar-first'*. Surprisingly, the *'All'* criterion has the least cost since it needs to run the placement algorithm only once to deploy the Pr SFC and as many times as the deported BE SFCs for redeployment. Other criteria, on the other hand, will need to run the placement algorithm twice the number of deported BE SFCs. One run to try deploying the Pr SFC, and the other for redeploying the deported BE SFC.

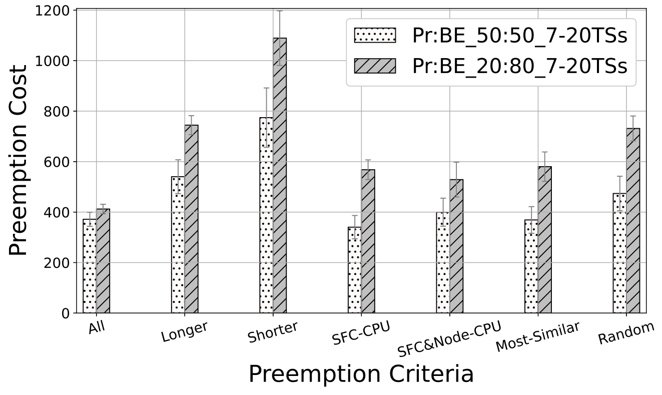


Fig. 8. Preemption Cost of preemption criteria

As expected, because of the unpredictable nature of the *'Random'* criterion, it fairly deports SFCs in Run_{be}, as shown in Figure 9. Using the *'Random'* criterion as a fairness reference, for *'50:50'* we can see that *'SFC&Node-CPU-first'* is as good as for *'Random'*. The overall fairness of *'20:80'* is way better than that of *'50:50,'* because the number of BE SFCs is 60% more in *'20:80'*.

VI. Conclusions and Future Work

This paper proposed IPTSV for immediate placement of time-critical services with VNF sharing. We found that *'SFC-*

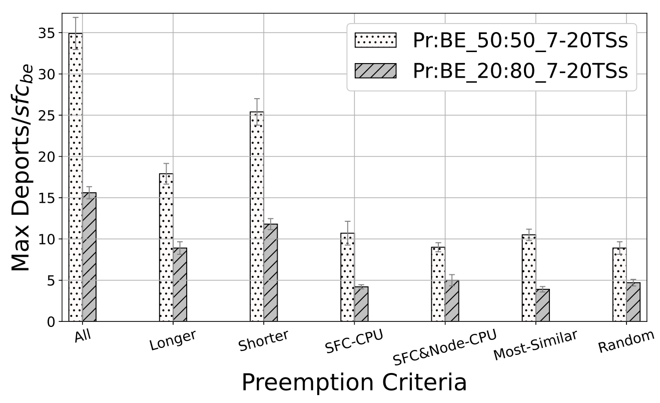


Fig. 9. Fairness of preemption criteria

CPU-first' is the least criterion to unnecessarily deport BE SFCs even when VNF sharing is not used, *'Most-similar-first'* and *'SFC&Node-CPU-first'* came as a close second and third, respectively. Again *'SFC-CPU-first'* is best for giving the least BE SFC AWT and is in close competition with *'Longer-first'* and *'Most-similar-first'* for the least TAT. Using equal costs, the *'All'* gives the least cost for both moderate and high loads, while the *'SFC-CPU-first'* criterion performs better in moderate load settings. The *'SFC-CPU-first,' 'Most-similar-first,'* and *'SFC&Node-CPU'* deport BE SFCs as fair as the *'Random'* criterion. It is clear that the *'SFC-CPU'* criterion is in the top-three if not the winner in all evaluation metrics. Finally, in environments where compute resources are scarce and would be preferably used to process actual services/workloads, the *'Random'* criterion works just fine.

We realized that the gratuitous deports of BE SFCs are unavoidable, even when using the best preemption criterion, because of VNF sharing and the unknown released resources when deporting BE SFCs. For future work, we will diagnose the failure of Pr SFC placement and use such diagnosis to design a better, less-disturbing to BE SFCs, preemption criterion. Moreover, to add the support of non-sequential SFCs to IPTSV, some preprocessing steps are needed. First, decompose the non-sequential chain into two or more sequential chains. Second, use IPTSV to do the placement of sequential chains. Finally, merge the individual placement solutions at the branching VNFs to get the solution for non-sequential SFC.

REFERENCES

[1] MarketsandMarkets, "Edge Computing Market by Component, Application, Organization Size, Vertical and Region - Global Forecast to 2025," https://www.marketsandmarkets.com/Market-Reports/edge-computing-market-133384090.html, accessed: 04-26-2022.

[2] ETSI, "Network function virtualization: An introduction, benefits, enablers, challenges & call for action," in *SDN and OpenFlow World Congress*, Oct. 2012, pp. 1–16.

[3] Ericsson, "Time-Critical Communication: Leading the next wave of 5G innovation," Ericsson, Technical-Overview, 2021. [Online]. Available: https://www.ericsson.com/4a9e9f/assets/local/internet-of-things/docs/19102021-time-critical-communication-brochure.pdf

[4] C. Grasso, K. E. KN, P. Nagaradjane, M. Ramesh, and G. Schembra, "Designing the tactile support engine to assist time-critical applications at the edge of a 5g network," *Computer Communications*, vol. 166, pp. 226–233, 2021.

[5] C. Liu, K. Liu, S. Guo, R. Xie, V. C. S. Lee, and S. H. Son, "Adaptive offloading for time-critical tasks in heterogeneous internet of vehicles," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 7999–8011, 2020.

[6] A. Jain and D. S. Jat, "An edge computing paradigm for time-sensitive applications," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, 2020, pp. 798–803.

[7] K. Ray, A. Banerjee, and N. C. Narendra, "Proactive microservice placement and migration for mobile edge computing," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 28–41.

[8] H. Yao, M. Xiong, H. Li, L. Gu, and D. Zeng, "Joint optimization of function mapping and preemptive scheduling for service chains in network function virtualization," *Future Generation Computer Systems*, vol. 108, pp. 1112–1118, 2020.

[9] M. Mechtri, C. Ghribi, and D. Zeghlache, "Vnf placement and chaining in distributed cloud," in *IEEE 9th International Conference on Cloud Computing (CLOUD)*, June 2016, pp. 376–383.

[10] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "Vnf placement optimization at the edge and cloud," *Future Int.*, vol. 11, no. 3, 2019.

[11] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *IEEE 3rd International Conference on Cloud Networking (CloudNet)*. IEEE, 2014, pp. 7–13.

[12] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," *IEEE Trans on Net (TON)*, vol. 26, no. 4, pp. 1562–1576, 2018.

[13] H. Yu, Z. Zheng, J. Shen, C. Miao, C. Sun, H. Hu, J. Bi, J. Wu, and J. Wang, "Octans: Optimal placement of service function chains in many-core systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2202–2215, 2021.

[14] M. Li, Q. Zhang, and F. Liu, "Finedge: A dynamic cost-efficient edge resource management platform for nfv network," in *2020 IEEE/ACM 28th Int. Symposium on QoS (IWQoS)*, 2020, pp. 1–10.

[15] D. Li, P. Hong, K. Xue, and J. Pei, "Virtual network function placement and resource optimization in nfv and edge computing enabled networks," *Computer Networks*, vol. 152, pp. 12–24, 2019.

[16] A. M. Medhat, G. Carella, J. Mwangama, and N. Ventura, "Multi-tenancy for virtualized network functions," in *Proceedings of the 1st IEEE Conf. on Net. Softwarization (NetSoft)*. IEEE, 2015, pp. 1–6.

[17] D. Li, J. Lan, and P. Wang, "Joint service function chain deploying and path selection for bandwidth saving and vnf reuse," *International Journal of Communication Systems*, vol. 31, no. 6, p. e3523, 2018.

[18] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 731–741.

[19] L. M. Contreras, A. Solano, F. Cano, and J. Folgueira, "Efficiency gains due to network function sharing in cdn-as-a-service slicing scenarios," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. IEEE, 2021, pp. 348–356.

[20] C. Mei, J. Liu, J. Li, L. Zhang, and M. Shao, "5g network slices embedding with sharable virtual network functions," *Journal of Communications and Networks*, vol. 22, no. 5, pp. 415–427, 2020.

[21] A. Mohamad and H. S. Hassanein, "Psvshare: A priority-based sfc placement with vnf sharing," in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2020, pp. 25–30.

[22] B. Yi, X. Wang, and M. Huang, "A generalized vnf sharing approach for service scheduling," *IEEE Communications Letters*, vol. 22, no. 1, pp. 73–76, 2017.

[23] F. Malandrino, C. F. Chiasserini, G. Einziger, and G. Scalosub, "Reducing service deployment cost through vnf sharing," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2363–2376, Dec 2019.

[24] N. Siasi and A. Jaesim, "Priority-aware sfc provisioning in fog computing," in *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, 2020, pp. 1–6.

[25] T. Bahreini and D. Grosu, "Efficient placement of multi-component applications in edge computing systems," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: Association for Computing Machinery, 2017.

[26] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service Function Chaining Use Cases in Mobile Networks," Internet Engineering Task Force, Internet-Draft, Jan. 2019, work in Progress.

[27] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2018. [Online]. Available: http://www.gurobi.com