PSVShare: A Priority-based SFC placement with VNF Sharing

Amir Mohamad
School of Computing
Queen's University
Kingston, Ontario, Canada
Email: {a.mohamad}@queensu.ca

Hossam S. Hassanein
School of Computing
Queen's University
Kingston, Ontario, Canada
Email: {hassanh}@queensu.ca

***Abstract*—Edge computing resources deployed at the access network are distributed and limited compared to the abundant core cloud resources. With the increasing demand on edge resources by emerging delay-sensitive use-cases, efficient resource utilization is to play an indispensable role. Taking advantage of operation dynamics and common functions in network/enterprise services, we propose PSVShare a priority-based, least-cost and resource-efficient service placement algorithm. The algorithm considers practical settings and conditions such as service categories (premium and best-effort), service arrival and completion times, and service relocation/migration due to changing traffic loads. PSVShare satisfies more services, achieves lower rejection rates, and is agnostic to arrival order and ratio of premium to best-effort services.**

***Index Terms*—NFV, VNF placement, Service Function Chaining, edge computing, MEC**

I. Introduction

Edge Computing is witnessing a phenomenal growth [1] and is a major player in the next generation of mobile networks (5G) and some enterprise use-cases. 5G requires innovation both on the network core side as well as the radio side. The network core is going to be fully virtualized utilizing the Network Function Virtualization (NFV) [2]. Edge computing will provide the required platform which will host the separated and virtualized user plane functions [3]. On the radio side, radio access network (RAN) components are disaggregated into distributed units (DUs) and centralized units (CUs). DUs are installed at cell sites and CUs are pooled and deployed in base-band units (BBUs) Hotels or Central Offices (COs). Edge computing secures the required infrastructure to host the virtualized CUs which has stringent real-time requirements and need to be placed as close as possible to the cell-sites.

In mobile/multi-access edge computing (MEC) as the mobile networks version of edge computing, hosted services will have access to user mobility data as well as wireless channel related measurements [4]. With such visibility, service providers will be able to take actions to enhance both the quality of service (QoS) and users' perceived quality of experience (QoE) at a millisecond scale.

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

Most enterprise and network services consist of component functions that are stitched together in a specific order to form service function chains (SFCs). NFV brings elasticity to the creation and deployment of services by virtualizing their functional components and decoupling them from their specially-built hardware [5]. Communications Service Providers' (CSPs) telco cloud, edge and core, provides the required infrastructure that hosts SFC's virtual network functions (VNFs). VNFs are assigned all required compute and bandwidth resources and are expected to operate at their full capacity. However, due to operation dynamics, some VNFs might be underutilized, receiving and processing traffic less than their full capacity. Services might have common VNFs, as seen in Figure 1, services S_1 and S_2 both have a common functional component V_2.

Edge computing has limited resources compared to the core cloud. Considering the anticipated high demand for edge resources and the importance of the edge being a precious asset for CSPs, the efficient utilization of edge resources will play a pivotal role in the fulfilment of delay-sensitive requirements of services and applications. With SFC requests continuously arriving, it would be more resource-efficient to utilize deployed underutilized VNFs first, and only deploy a new VNF instance if no deployed underutilized VNF of the same type exists. This should be done while being mindful of the SFC performance requirements. We decided to use end-to-end latency as the performance requirement that has to be satisfied.

To date, only a few research works with limited scope exist on VNF sharing. Previous studies either consider predefined number of traffic flows a VNF can handle, ignoring the operation dynamics, or propose prioritization mechanisms that dynamically assign priorities based on current system state. Hence, in this paper, we propose PSVShare a priority-based SFC placement algorithm with VNF sharing. PSVShare takes SFC placement decisions at the point-of-presence (PoP) level, and supposed to be part of the NFV orchestrator (NFVO).

The main contributions of this paper are: the introduction of VNF sharing-based SFC placement, prioritizing premium (Pr) services over best-effort (BE) services; and a migration scheme to handle situations where a host VNF cannot accommodate traffic increase, as a result of sharing its capacity with

978-1-7281-8159-2/20/$31.00 ©2020 IEEE

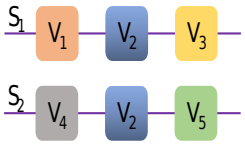

Fig. 1: SFC S_1 and S_2 with common VNF V_2

guest VNFs.

The remainder of this paper is structured as follows. Section II covers related work. Proposed sharing-based SFC placement, system model, and problem formulation are detailed in section III. Performance evaluation and results analysis are covered in section IV. Conclusions and future works are presented in section V.

II. Related Work

The work in [6] represents the generic VNF/SFC placement. Others introduced placement at the edge-central cloud and at mobile network edge [7]. With the introduction of NFV in 2012 [2], service provisioning became more agile and placement of VNFs as the building block of SFCs started to gain traction. Sharing of a VNF by more than one SFC flow is triggered by the fact that some VNFs could be shared by SFC flows, such as anti-virus and parental-control. With the exception of the work in [8] and [9], existing work does not consider sharing deployed underutilized VNFs while deploying new SFC requests. The work in [8], proposes sharing VNF among SFC flows based on a predefined number of flows that a VNF can handle ignoring the operation dynamics. The work in [9] is relatively newer and proposes a different way of priority-based provisioning with VNF sharing. The priority is dynamically assigned to flows or VNFs, depending on the situation, rather than predetermined before deployment. Also, VNF container, virtual machine or container, is treated as an infrastructure asset, rather than an ephemeral component. In our work [10], using simple system settings, we demonstrated the performance gain of SFC placement with VNF sharing.

III. System Model and Problem Formulation

In a typical service domain, services are not equal in terms of priority and preferences. It has been pointed out, including the work in [11], that services come in two types/classes: premium (Pr) service that is provisioned with highest expected load, not oversubscribed, and served with dedicated high-priority queues; and best-effort (BE) service is sent and queued with lower priority. Moreover, due to operation dynamics, the traffic flow that deployed services/SFCs receive and process, continuously varies up and down. A practical and efficient SFC placement algorithm should consider both the aforementioned practical service domain aspects. To do so, PSVShare is a priority-based SFC placement with VNF sharing, which will have to handle complex situations. Take for example the operation dynamics, which results in a varying traffic load that SFCs and their VNFs serve. The capacity of a host VNF is distributed among its guest VNFs. Any increase of the traffic flow of the host VNF or any of its guest VNFs, may require one or more VNFs, hence SFCs, to be deported to accommodate the traffic increase. Doing so in light of two service categories, should strive to minimize the migrations a Pr SFC will have to experience. Out of limited resources at that time, an SFC that is deported may end up in a queue, waiting to be redeployed once resources become available. Even though PSVShare is designed for edge service provisioning at the network edge, it is usable for core services as well.

A. System Model

An SFC request consists of an ordered list of VNFs and comes with a service category, Pr or BE. VNFs are selected from a list V of on-boarded VNFs. In this list, each VNF has resource requirements and a maximum traffic flow it can handle once assigned the resources required. Some VNFs can be shared among SFC flows while others not. $S(v_i)$ is a flag to determine whether VNF v_i is shareable or not. Once selected in sfc_j, the inflow and outflow of VNFs should be determined. The substrate network is modeled as a graph $G(N, E)$, where N is the set of nodes, with compute resources, and E is the set of links. Each link has bandwidth capacity and propagation delay. The substrate network topology is fixed and described by a connectivity matrix. The description of substrate network and SFC parameters is provided in Table I.

TABLE I: Parameters description

Parameter	Description
N	Set of substrate network nodes
E	Set of substrate network links
$cpu_c(n)$	CPU capacity in cores of node $n \in N$
$ram_c(n)$	RAM capacity in GBs of node $n \in N$
$cpu_{av}(n)$	Available CPU cores at node $n \in N$
$ram_{av}(n)$	Available RAM GBs at node $n \in N$
$L_{nn'}$	A link exists from node n to node $n^{'}$, $n, n^{'} \in N$
$bw_c(L_{nn'})$	BW capacity in Mbps of link $L_{nn'}$
$bw_{av}(L_{nn'})$	Available BW at link $L_{nn'}$
$Del(L_{nn'})$	Propagation delay of link $L_{nn'}$
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 $v_i \in V$
$ram(v_i)$	RAM GBs required for VNF $v_i \in V$
$F_{max}(v_i)$	Maximum inflow VNF v_i can handle
$S(v_i)$	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
$cat(v_i)$	Category of VNF v_i's SFC , BE = 1 & Pr = 2
$\|sfc_j\|$	Number of VNFs in sfc_j
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

B. Problem Formulation

The PSVShare algorithm utilizes an integer quadratically-constrained program (IQCP) model. The objective is to minimize the total deployment cost by optimizing resource utilization while satisfying QoS requirements/constraints. The PSVShare prioritizes Pr SFCs over BE SFCs and handles migration situations that arises due to VNF sharing and traffic increase. PSVShare, listed in Algorithm (1), considers SFC requests arriving at the beginning of each time-slot (TS). Once received, PSVShare uses the IQCP to find the least-cost deployment solution. If a solution exists, the state of satisfied SFC changes from *received* to *running* and gets added to the Pr or BE running queue $Run_{pr|be}$. Run_{pr} queue is for Pr SFCs and Run_{be} queue is for BE SFCs. In case of no solution, SFC state will change to *waiting* and gets added to the $New_{pr|be}$ queue. Once deployed a *running* SFC is subject to one of three possible state changes. If the SFC's time-to-live (TTL) is zero at the start of a TS, its state changes from *running* to *completed* and gets moved from the $Run_{pr|be}$ to the $Comp_{pr|be}$ queue. If traffic flow increase cannot be accommodated, either because host VNF own flow or due to an increase of a guest VNF flow, one or more migrations are unavoidable. If an SFC migration is successful, SFC ends in the *running* state, but passing by a *terminated* state. Otherwise, the state changes to *pending-migration* and the SFC moves from the $Run_{pr|be}$ to the $Mig_{pr|be}$ queue.

Besides the TTL checks/decrements, PSVShare starts with satisfying the Pr SFCs pending migration in the Mig_{pr}, then BE SFCs in the Mig_{be}. For those SFCs waiting in the $New_{pr|be}$ queue, PSVShare attempts to satisfy Pr SFCs then BE SFCs. Finally, it checks if there is any migration is triggered, because of traffic flow increase. To do so, and for all SFCs in the $Run_{pr|be}$ queues, PSVShare checks if the traffic increase is not applicable. The SFC under-investigation, with inapplicable traffic increase, is sent to the $migReqrd$ module, detailed in Algorithm (2), to return a list of SFCs to be deported to accommodate the traffic increase. If a host VNF of the SFC under-investigation cannot handle the traffic increase, first $migReurd$ tries to add VNF's BE guest SFCs with highest inFlow and TTL to SFC migration list $migList$. If all the guest BE SFCs of host VNF are not enough to accommodate the traffic increase, $migReqrd$ will resort to complete the remaining required flow by looking to VNF's guest Pr SFCs. If a guest VNF of the SFC cannot accommodate the traffic increase, the SFC under-investigation itself is added to the $migList$. PSVShare checks if the $migList$ contains the SFC under-investigation, if so, only that SFC is terminated. Otherwise, all SFCs in the $migList$ are terminated. After accommodating the traffic increase, PSVShare attempts to re-satisfy all terminated SFCs.

*1) **IQCP**:* To satisfy an SFC, new or deported, the $satisfy$ function in Algorithm (1): lines[9,30, and 37] is implemented as an IQCP, in which decision variables are binary and some constraints are quadratic. With SFC request sfc_j consisting of VNFs $v_i, i \in [1 - |sfc_j|]$, our decision variables are: X^j_{in}

Algorithm 1: PSVShare

Input : netM, No.TSs
Init. : queues:$New_{pr|be}$, $Mig_{pr|be}$, $Run_{pr|be}$, $Comp_{pr|be}$
Output: Different queues

```
1  for i ← 1 to No.TSs do
2      if i > 1 then
3          foreach SFC rs_{pr|be} in Run_{pr|be} do
4              if ttl(rs_{pr|be}) = 0 then
5                  remove(rs_{pr|be}, Run_{pr|be})
6                  add(rs_{pr|be}, Comp_{pr|be})
7              else decTTL(rs_{pr|be})
8          foreach SFC ms_{pr|be} in Mig_{pr|be} do
9              sol ←satisfy(ms_{pr|be}, netM)
10             if sol ≠ ∅ then
11                 remove(ms_{pr|be}, Mig_{pr|be})
12                 deploy(ms_{pr|be}, netM)
13                 add(ms_{pr|be}, Run_{pr|be})
14         foreach SFC ns_{pr|be} in New_{pr|be} do
15             sol ←satisfy(ns_{pr|be}, netM)
16             if sol ≠ ∅ then
17                 remove(ns_{pr|be}, New_{pr|be})
18                 deploy(ns_{pr|be}, netM)
19                 add(ns_{pr|be}, Run_{pr|be})
20         foreach SFC rs_{pr|be} in Run_{pr|be} do
21             if traffChange(rs_{pr|be}, nIF) ≠ true
                then
22                 migList ←migReqrd(rs_{pr|be}, nIF)
23                 if contains(migList, rs_{pr|be}) =
                    true then terminate(rs_{pr|be})
24
25                 else foreach SFC ms_{pr|be} in
                    migList do
26                     terminate(ms_{pr|be})
27
28                 ApplyTraffChange(rs_{pr|be}, nIF)
29                 foreach SFC ms_{pr|be} in migList do
30                     sol ←satisfy(ms_{pr|be}, netM)
31                     if sol ≠ ∅ then
32                         deploy(ms_{pr|be}, netM)
33                         add(ms_{pr|be}, Run_{pr|be})
34                     else add(ms_{pr|be}, Mig_{pr|be})
35
36     else foreach new SFC nas_{pr|be} do
37         sol ←satisfy(nas_{pr|be}, netM)
38         if sol ≠ ∅ then
39             deploy(nas_{pr|be}, netM)
40             add(nas_{pr|be}, Run_{pr|be})
41         else add(nas_{pr|be}, New_{pr|be})
42
43
```

Algorithm 2: migReqrd function

```
// nIF: newInFlow & nOF: newOutFlow
Input  : rs_{pr|be}, nIF
Output: migList: if migration required, null: otherwise
Function migReqrd(rs_{pr|be}, nIF):
    foreach VNF v_i in rs_{pr|be} do
        if isHost(v_i) = true then
            if (nIF-F_in(v_i)) > F_av(v_i) then
                add(v_i,diagList)
        // v_i is a guest VNF
        else if (nIF-F_in(v_i)) > F_av(hostVNF(v_i)) then
            add(v_i,diagList)
        if (nOF-F_out(v_i)) > bw_av(outLink(v_i)) then
            add(v_i,diagList)
    if diagList isn't empty then
        foreach VNF v_i in diagList do
            if isHost(v_i) = true then
                // Add BE SFCs sorted with ↑ inFlow and ↓ TTL
                add(SFCs_be,migList)
                if Added SFCs' inFlow isn't enough then
                    // Add Pr SFCs sorted with ↑ inFlow and ↓ TTL
                    add(SFCs_pr,migList)
            else add(rs_{pr|be},migList)
        return migList
    // No migration(s) required
    else return null
```

to place new instance of VNF v_i of sfc_j at node n; and R^j_{in} 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. Decision variables and other parameters descriptions are in Table II.

*a) **Objective Function**:* The objective is to select the placement that minimizes the overall cost, hence optimize resource utilization, and minimizes number of migrations of Pr SFCs. The objective function in equation (1) is formulated in a way to prefer sharing over instantiating new VNF instances.

$$min \sum_{i=1}^{|sfc_j|} \sum_{n \in N} [cpu(v_i^j) U^c_{cpu}(n) + ram(v_i^j) U^c_{ram}(n)] X^j_{in} + F_{out}(v_i^j) U_c bw [X^j_{in} + R^j_{in}] \quad (1)$$

*b) **Constraints**:* First, a feasible solution has to have each VNF of SFC request mapped only once to a physical node. Moreover, the mapping should be either to place a new or to share a deployed VNF instance, equation (2). Second, when a sharing decision is to be taken, there has to be a deployed shareable VNF of the same type as the one in hand, equation (3), and there is unused capacity enough for new VNF inflow, equation (4). Third, for the placement decision X^j_{in} to be valid, there must be enough cpu and ram resources at node n, equations (5 & 6). Fourth, for any two consecutive VNFs v^j_i and v^j_{i+1} of sfc_j to be placed on two nodes n and $n^{'}$: first, there has to be a link $L_{nn'}$ connecting the two nodes, equation (7); second, the outflow of first VNF $F_{out}(v^j_i)$ should not exceed available bandwidth at that link $bw_{av}(L_{nn'})$, equation (8). To minimize number of migrations of Pr SFCs, Pr SFCs should only be hosted by Pr SFCs, while BE SFCs can be a guest of Pr and BE SFCs, equation (9). Finally, the performance requirements (end-to-end latency) of sfc_j must be satisfied, equation (10).

TABLE II: Decision variables and Constants

Variable	Description
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

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

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

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

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

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

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

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

$$cat(v_i^j)\ R_{in}^j \ \leq\ cat(D_n^i),\ \forall n \in N\ \&\ \forall i \in [1 - |sfc_j|] \quad (9)$$

$$\sum_{i=1}^{|sfc_j|-1} \sum_{n \in N} \sum_{n' \in N} Del(L_{nn'})\ (X_{in}^j\ +\ R_{in}^j)\ (X_{(i+1)n'}^j\ +\ R_{(i+1)n'}^j) \leq Del(sfc_j) \quad (10)$$

IV. Performance Evaluation

To evaluate and demonstrate the performance of PSVShare, we developed a Java-based simulation environment. The simulation environment generates substrate network model, creates SFC requests, executes the placement decisions, and tracks SFCs different state/queue transitions. SFC requests arrival follows a Poisson distribution. The service time, i.e. SFC duration in TSs, follows an exponential distribution. The average length of generated SFC requests is 5 VNFs. The IQCP model is solved using the Gurobi solver [12]. All simulation experiments executed on Dell Inspiron 15 7000 laptop with Intel(R) core i5-8250U CPU@1.6 GHz, 24 GB RAM with Windows 10 Home. We used the NSFNET network topology with 13 nodes and 32 directional links.

All experiments utilized the same network model, with the same topology, nodes resources, and links bandwidth. Unless we are experimenting with the arrival rate λ, experiments are done with $\lambda = 2$ SFCs per TS. With simulation time set to 100 TSs, we generated and saved SFCs for 100 TSs, i.e. number of SFC requests per TS and length of each SFC. Unless otherwise stated, SFCs generated are 50% Pr and 50% BE with shuffled order of arrival. The only difference among experiments, is the type of VNFs each SFC is comprised of as well as the QoS requirements. SFC requests are generated with VNFs in the list of on-boarded VNFs. The list contains VNFs of different flavours and requirements. Unless stated otherwise, in all experiments, PSVShare utilizes 60% shareable VNFs. Finally, if not evaluating the queue sizes or SFC expiration, the queue sizes are not bounded and SFCs never expire.

A. Numerical Results and Analysis

To assess the proposed PSVShare under extreme cases, we compare PSVShare with SFCs created from "100%" shareable VNFs vs SFCs created from "0%" shareable VNFs. Figure 2 shows the different queues sizes as percentage of received SFCs at the end of simulation. As shown, having "100%" shareable VNFs resulted in satisfying, completed and running, 26% more SFCs compared to the "0%" shareable case. This reveals the positive impact PSVShare has over efficient resource utilization and the provisioning cost of services. This is attributed to VNF sharing that PSVShare utilizes, which increases the effective capacity of network, and hence satisfy more SFCs.

To assess the performance of PSVShare under different loads, we experimented with different arrival rates $\lambda = \{1, 1.5, 2, 2.5, 3, 4\}$, using the same network model (same

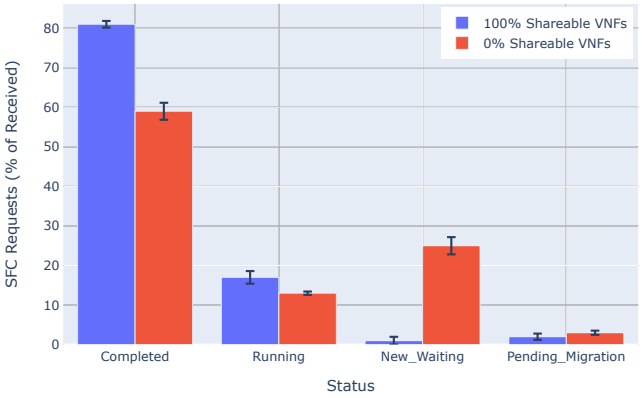

Fig. 2: PSVShare with 100% shareable VNFs vs 0% shareable VNFs. $\lambda = 2$ and shuffled $50 - 50\%$ Pr-to-BE ratio

capacity). Results, in Figure 3, show that PSVShare started to shine at higher loads. As we can see, at $\lambda = 1$, the satisfied (Completed and running) and pending SFCs are the same both for PSVShare and No-Sharing scheme. Once the load increased to $\lambda = 1.5$ and higher, the percentage of PSVShare satisfied SFCs is 14% to 25% more than No-Sharing. Again this is contributed to VNF sharing before deploying new VNF instances when satisfying SFC requests.

The rejection rate is a very important aspect that impacts real-time and time-critical services that need to be deployed once requested. In all previous experiments, the queue sizes were not bounded. For this study, we limited the New_{Pr} & New_{be} queues to a finite size equal to a percentage of total number of received SFC requests. We experimented with queue size equal to 0, 2, 4, and 6% of received SFC requests. In Figure 4, results show that PSVShare maintained a stable superior performance against the No-Sharing scheme, 95% confidence interval considered. The rejection rate of PSVShare ranged from 15% to 22% less compared to that of No-Sharing. The reason for such behaviour is that, at the time the No-Sharing network model started to saturate, i.e. no more resources were available for satisfying new SFCs, the PSVShare is still able to satisfy SFC requests. This is due to the efficient resource utilization of PSVShare.

To check whether PSVShare still outperforms the No-Sharing scheme when most of the received SFCs are Pr or BE, we experimented with different Pr-to-BE ratios. Figure 5 shows similar behaviour in the extreme cases with '0-100' and '100-0' Pr-to-Be ratio, with about 20-23% more satisfied Pr SFCs for PSVShare. The same applies for the '20-80' and '80-20' cases. For the '40-60', '50-50', and '60-40' cases, Pr SFCs have the higher priority, so PSVShare and No-Sharing satisfied almost the same percentage of received Pr SFCs. The advantage of PSVShare over the No-Sharing scheme is evident in the percentage of satisfied BE SFCs. This is mainly because, No-Sharing depleted big share of network model resources when satisfied most Pr and almost half of the BE SFCs, and left not enough resources for the rest of BE SFCs. Although, PSVShare satisfied the same percentage of Pr SFCs, yet it

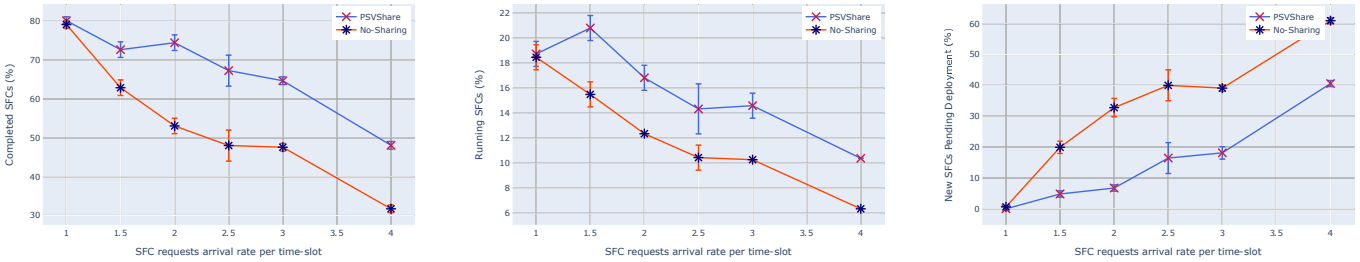

Fig. 3: Completed, running and new-waiting SFCs under different loads. All with 60% shareable VNFs, and shuffled $50-50\%$ Pr-to-BE ratio

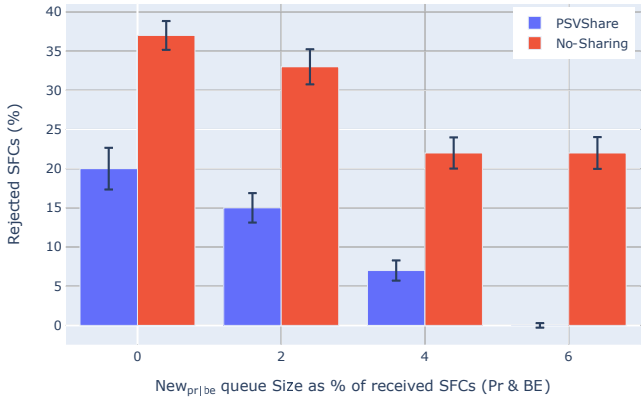

Fig. 4: Rejected SFC requests (%) for different queue sizes

is still able to satisfy more BE SFCs. This is due to the efficient resource utilization, both when satisfying Pr and then BE SFCs.

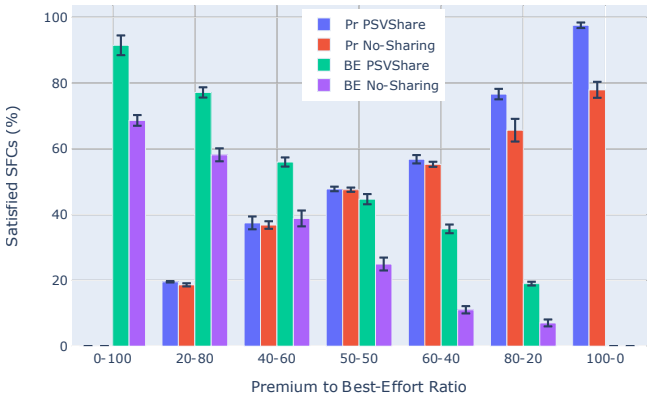

Fig. 5: Satisfied SFC requests (%) of different Pr-to-BE ratios, shuffled

V. Conclusion and Future Work

To take advantage of the changing operation dynamics and common VNFs among services, we designed PSVShare, a priority-based SFC placement with VNF sharing algorithm. PSVShare handles migration situations arising from sharing VNFs and traffic variation. Simulation results show a consistent outperformance of PSVShare over No-sharing scheme under different loads, with varying Pr-to-BE ratios, and with different queue sizes.

Generally, Pr SFCs are very critical and need to be immediately deployed once received with no tolerance to waiting for deployment. As a future work, we will work on a priority-based SFC placement utilizing VNF sharing with preemption. In situations where resources are not available to satisfy Pr SFCs, a criteria should be in-place to preempt lower priority/BE SFCs. The preemption criteria should strive to balance between, immediately satisfying Pr SFCs and minimizing number of preempted BE SFCs.

References

[1] VMware Telco Cloud Blog, "Innovation at the Telco Edge," https://blogs.vmware.com/telco/innovation-at-the-telco-edge/, accessed: 03-08-2020.

[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] F. van Lingen, M. Yannuzzi, A. Jain, R. Irons-Mclean, O. Lluch, D. Carrera, J. L. Perez, A. Gutierrez, D. Montero, J. Marti, R. Maso, and a. J. P. Rodriguez, "The unavoidable convergence of nfv, 5g, and fog: A model-driven approach to bridge cloud and edge," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 28–35, Aug 2017.

[4] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, thirdquarter 2017.

[5] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service function chaining use cases in mobile networks," IETF, Internet Draft, January 2019.

[6] 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/ACM Transactions on Networking (TON)*, vol. 26, no. 4, pp. 1562–1576, 2018.

[7] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "Vnf placement optimization at the edge and cloud †," *Future Internet*, vol. 11, no. 3, 2019. [Online]. Available: http://www.mdpi.com/1999-5903/11/3/69

[8] 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.

[9] 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.

[10] A. Mohamad and H. S. Hassanein, "On demonstrating the gain of sfc placement with vnf sharing at the edge," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[11] K. Nichols, V. Jacobson, and L. Zhang, "Rfc2638: A two-bit differentiated services architecture for the internet," USA, 1999.

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