Service Provisioning at the Network Edge

A *VNF-Sharing* Approach

By Amir Mohamad

A thesis submitted to the

School of Computing

in conformity with the requirements for

the Degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
June, 2022

Abstract

The next generation of mobile networks (5G) is expected to address the performance requirements of various use cases in different industries, including ultra-reliable low latency communication (URLLC). The demand for URLLC requirements is fueled by the growing popularity of real-time media, delay-sensitive, and time-critical applications. Most services consist of virtual network functions (VNFs) stitched together in a specific order that form service function chains (SFCs). Premium SFCs are time-critical, and their failure (pre- or post-deployment) could result in Quality of Experience (QoE) degradation. Best-effort SFCs are prone to delay, and can tolerate waiting for resource availability.

Edge computing is positioned to fulfill the aforementioned stringent latency requirements. Due to edge limited resources, the tendency to reject service requests -including time-critical ones- can be high, translating into unsatisfied customers, lost revenue for service providers, and even safety hazards. Yet, existing work on the provisioning of SFCs at the edge make unrealistic assumptions. Such assumptions include: the edge has an abundance of resources; and SFCs have the same priority.

In this thesis, to improve the utilization of the service provider's limited edge resources and reduce the cost of service provisioning, we introduce the sharing of VNFs

among different SFCs. Taking advantage of operations dynamics, *VNF sharing* utilizes the unused capacity of deployed VNFs before instantiating new ones when satisfying new SFC requests. Specifically: 1) Using *VNF sharing*, the system utilization is enhanced by satisfying more SFC requests and using fewer resources per request. 2) SFC placement schemes that prioritize premium over best-effort services are introduced. 3) Prediction- and preemption-based placement schemes for time-critical SFCs are proposed to mitigate the consequences of SFCs rejections.

Extensive simulations demonstrate the effectiveness of *VNF sharing*-based schemes in achieving significant improvements to system utilization and reducing the rate of rejection of premium SFCs. The results indicate that *VNF sharing* helps service providers lower the service provisioning cost while respecting the stringent delay budget of time-critical applications and services.

Co-Authorship

Journal Articles:

1. **Amir Mohamad** and Hossam S. Hassanein, "Preemptive Prediction-based SFC Placement with VNF Sharing at the Edge," 2022, (In preparation).

Conference Publications:

1. **Amir Mohamad** and Hossam S. Hassanein, "On Demonstrating the Gain of SFC Placement with VNF Sharing at the Edge," IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9014106.

2. **Amir Mohamad** and Hossam S. Hassanein, "PSVShare: A Priority-based SFC placement with VNF Sharing," IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2020, pp. 25-30, doi: 10.1109/NFV-SDN50289.2020.9289837.

3. **Amir Mohamad** and Hossam S. Hassanein, "At the Edge? Wait no More: Immediate Placement of Time-Critical SFCs with VNF Sharing," IEEE 8th International Conference on Network Softwarization (NetSoft), 2022.

4. **Amir Mohamad** and Hossam S. Hassanein, "Prediction-based SFC Placement

with VNF Sharing at the Edge," IEEE 47th Conference on Local Computer Networks (LCN), 2022.

Acknowledgments

Praise to Allah, who has guided us to this; and we would never have been guided if Allah had not guided us.

I will be my life indebted to my supervisor Prof. Hossam Hassanein, for giving me the opportunity to join his lab and supporting me throughout the stressful PhD journey.

I would like to thank my wife Samiha for the endless and unconditional support she provided and for taking the burden of caring for our kids while I was traveling and busy with submissions and deadlines. The gratitude and apology extends to my kids Adham and Hana. They were the source of encouragement to continue in this journey. The apology is for the many times I was not there for them and for not being the father I should have been.

I would like to acknowledge my parents, brothers and sisters. This thesis, a manifestation of the many sleepless nights, would not have been possible without their support.

I would like to thank my colleagues and friends Adel Ibrahim, Abdullah Abdelrahman, Ahmad Nagib, and Mohannad Alharthi, for the fruitful discussions and for the endless support and guidance I received throughout my time at the TRL.

Statement Of Originality

I hereby certify that this Ph.D. thesis is original and that all ideas and inventions attributed to others have been properly referenced.

Contents

List of Tables

List of Figures

List of Acronyms

3GPP 3rd Generation Partnership Project.

ACL Access Control List.

AI Artificial Intelligence.

AR Augmented Reality.

ARPU Average Return Per User.

BBU Base-band Unit.

BSS Business Support System.

C-RAN Cloud RAN.

CAPEX Capital Expenditure.

CDN Content Delivery Network.

CDNaaS CDN as a Service.

CG-NAT Carrier-grade NAT.

CMS Cloud Management System.

CNF Cloud-Native Network Function.

CO Central Office.

CPE Consumer Premise Equipment.

CSPs Communications Service Providers.

CU Centralized Unit.

CUPS Control and User Plane Separation.

DPI Deep packet inspection.

DU Distributed Unit.

eMBB enhanced mobile broadband.

EMS Element Management System.

EPC Evolved Packet Core.

ETSI European Telecommunications Standards Institute.

FW Firewall.

HA Highly Available.

IDS Intrusion Detection System.

IETF Internet Engineering Task Force.

IIC Industrial Internet Consortium.

IIoT Industrial IoT.

ILP Integer Linear Program.

IoT Internet of Things.

IPS Intrusion Prevention System.

ISG Industry Specifications Group.

K8s Kubernetes.

LF-Edge Linux Foundation Edge.

LFN Linux Foundation Networking.

MANO Management and Orchestration.

MEC Multi-access Edge Computing.

MIP Mixed Integer Program.

mMTC massive machine-type communications.

MR Mixed Reality.

MWD Malware detection and elimination.

NBI Northbound Interface.

NETCONF Network Configuration.

NFV Network Function Virtualization.

NFVI Network Function Virtualization Infrastructure.

NFVO Network Function Virtualization Orchestrator.

NSH Network Service Header.

O-RAN Open RAN.

ODL Open Daylight.

ONAP Open Network Automation Platform.

ONF Open Networking Foundation.

OPEX Operation Expenditure.

OSM Open Source Management and Orchestration.

OSS Operation Support System.

OTT Over-The-Top.

OVSDB Open Virtual Switch Database.

P-GW Packet gateway.

PBR Policy-based Routing.

PEP Performance Enhancement Proxy.

PNF Physical Network Function.

QoE Quality of Experience.

QoS Quality of Service.

RAN Radio Access Network.

RoIC Return on Invested Capital.

RRU Remote Radio Unit.

RU Radio Unit.

SBI Southbound Interface.

SD-RAN Software-defined RAN.

SD-WAN Software-defined WAN.

SDN Software-Defined Networking.

SDN-C SDN Controller.

SFC Service Function Chain.

SLA Service Level Agreement.

TOSCA Topology Orchestration Specification for Cloud Applications.

URLLC ultra-reliable low-latency communications.

VIM Virtualized Infrastructure Manager.

VM Virtual Machine.

VMP VM Placement.

VNE Virtual Network Embedding.

VNF Virtual Network Function.

VNFM Virtual Network Function Manager.

VR Virtual Reality.

Chapter 1

Introduction

1.1 Introduction

Over the past decade, cloud computing has dominated the information technology (IT) landscape. Cloud computing's salient feature is the reduction in the required time to market a new service as well as the reduction/savings in the total cost of ownership (TCO) [104]. Thanks to the globally distributed data-centers and high internet speeds, users tend to disregard the physical proximity to data-centres. The false sense of disregarding physical proximity cannot stand anymore in the face of the challenges posed by most delay-sensitive use cases, such as Internet of Things (IoT), Industrial IoT (IIoT), multi-player gaming, and virtual, augmented and mixed reality (VR/AR/MR) applications [38, 104]. Cloud computing cannot provide for the challenging requirements of the mentioned applications and use cases, such as

1. Unprecedented increase in generated data by end devices and the prohibitively excessive back-haul bandwidth required to push such data to the core cloud.

2. Ultra-low latency required by delay-sensitive applications that cannot tolerate the round-trip time between the data source and cloud data center.

3. Privacy requirements of some applications that necessitate processing generated data locally and not moving data beyond defined geographical boundaries which is not always satisfiable by cloud data centers.

4. The need for an uninterrupted service even with intermittent internet/cloud connectivity.

5. The cloud cannot provide the situational awareness that requires a timely acquisition of knowledge about events and conceptual reasoning of those events into a complete view related to a specific mission [102].

Edge computing is a distributed computing infrastructure close to data sources that facilitates decentralized processing and is considered the enabling technology positioned to satisfy the requirements of emerging applications where cloud computing falls short. The interest and demand for edge computing are growing, fuelled by the growing interest of both Communications Service Providers (CSPs) and Over-The-Top (OTT) service providers. The edge computing market is projected to grow from $36B in 2021 to $87B by 2026, at a compound annual growth rate of 19% during the forecast period [83].

By adopting Network Function Virtualization (NFV) and Software-Defined Networking (SDN), service provisioning is more agile, scalable, and economical because CSPs can reduce capital and operations expenditures [44]. To provision a service, an orchestrator must make a placement decision to associate service functions (VNFs) with physical hosting nodes. Most enterprise and network services consist of component functions/VNFs forming service function chains (SFCs) and traffic traverses these functions in a specific order.

1.2 Motivation

While there are similarities between cloud and edge computing, edge computing has several unique features such as the distributed resources across many sites that need to be managed; limited resources compared to the abundant cloud resources; and the challenging service provisioning with distributed and limited resources [95]. With the edge's limited resources and the increasing demand for edge computing, efficient resource utilization will play a major role. The majority of emerging 5G use cases are time-critical, such as real-time media (VR/AR/MR), industrial control, remote control, and mobility automation [43]. Time-critical, henceforth premium (Pr), services and applications have stringent time constraints and would fail if such constraints were not met [54, 61, 78]. Consequently, service placement must consider delay-urgency/priority to minimize or eliminate the wait and/or rejection of premium services. Enhancing the utilization of edge resources will help CSPs satisfy more services and seize revenue opportunities.

This thesis demonstrates how *VNF sharing*-based service placement schemes tackle the challenges arising from limited edge resources and the increasing demand for delay-sensitive applications. Extensive simulations demonstrate the effectiveness of our schemes in increasing the utilization and significantly reducing/eliminating the premium services rejections.

1.3 Research Statement

Service placement, including *VNF sharing*-based placement, has been extensively addressed in the literature; however, some unrealistic assumptions were used such as setting a predefined number of flows that can share a deployed VNF [86], mixing the

concept of infrastructure assets and *VNF Sharing* [76] and a priority that changes based on the situation or the required resources [81, 105].

We believe that:

Priority-aware *VNF sharing*-based service placement will enhance edge resource utilization, reduce premium services rejections, and help CSPs serve their customers better and increase revenue.

1.4 Thesis Contributions

The main contributions of this thesis are as follows:

1. *Improving Edge Resources Utilization*: We present an SFC placement scheme with *VNF sharing* to improve edge resource utilization and satisfy more SFC requests while fulfilling the performance requirements (mainly maximum end-to-end delay). Taking advantage of operation dynamics, common VNFs among SFCs, and the shareability of some VNFs, we first utilize the free capacity of deployed shareable VNFs when satisfying new SFC requests. The sharing decision is based on the state of deployed VNFs at the time of placement decision-making with no prior assumptions.

2. *Priority-Based Service Placement*: PSVShare is a priority-based SFC placement scheme coupled with a *VNF Sharing*, prioritizing premium services over best-effort services. Moreover, a migration scheme to handle situations where a host VNF cannot accommodate traffic increase, as a result of sharing its capacity with guest VNF(s). VNFs are treated as ephemeral components not infrastructure assets, the priority is known and fixed before deployment, and all SFC's VNFs have the same priority. Our results demonstrate that PSVShare

satisfies more services, achieves lower rejection rates compared to placement schemes without *VNF Sharing*, and is independent of the arrival order and ratio of premium and best-effort services.

3. *Prediction-Based Service Placement*: Even though the *VNF Sharing* has increased system utilization and satisfied more premium services, the rejections of premium services are still concerning. To remedy this, we present PSVS, a prediction-based SFC placement scheme with *VNF Sharing* to reduce the rejection rate of premium services significantly. Considering the predicted arrival of premium services requests, PSVS will decide to satisfy best-effort requests or defer the deployment to save resources for future premium requests. Even though PSVS significantly reduces premium services rejection rate, premium rejections are still concerning.

4. *Immediate Placement of Time-Critical Premium Services*: We investigate the rejections that impact time-critical services due to resource limitations which might cause terrible consequences. As such, we design IPTSV, a preemptive placement scheme that reduces the premium services rejection rate to near zero, define the baseline performance of preemption-based service placement, and recommend which preemption criterion to use given the service domain context and provider's policies and priorities. Our results show an almost zero rejection rate of time-critical premium services and the best preemption criterion that is least disturbing to best-effort services.

1.5 Thesis Outline

In this chapter, we present the motivation of the primary research problem, and highlight the major contributions toward *VNF Sharing*-based service placement at the network edge. The rest of this thesis is organized as follows.

Chapter 2 introduces the background and related work. We review the importance, enabling technologies and fundamental concepts of edge computing as well as edge platforms. At the end of this chapter, we explain popular service placement, migration and replication techniques.

Chapter 3 introduces our *VNF Sharing*-based service placement scheme. We begin with how *VNF Sharing* will better the edge resource utilization. We then describe the system settings, based on which the *VNF Sharing*-based placement scheme is formulated. We describe the assumptions, system and simulation settings, and discuss results.

Chapter 4 highlights our priority-based service placement scheme (PSVShare). We start with the reasons for prioritizing certain service categories over others, followed by a review of related work and how our scheme is addressing the overlooked important aspects of priority-based service placement with *VNF Sharing*. To validate PSVShare claims, we compare the performance of PSVShare against priority-based placement without *VNF Sharing*. A heuristic technique is presented to overcome the computational complexity of ILP-based placement scheme.

Chapter 5 builds on the priority-based placement scheme and presents a prediction-based service placement scheme (PSVS) to reduce premium service rejection rate due to resource limitations. PSVS uses the same ILP-based placement as PSVShare but utilizes predicted required resources of future premium service that will arrive in a

defined lookahead window. Indeed, PSVS significantly reduces the premium service rejection rate; however, premium service rejections are not eliminated.

Chapter 6 introduces the zero rejections preemption-based service placement scheme (IPTSV). To eliminate premium service rejections, when premium services requests cannot be satisfied due to resource limitations, IPTSV utilizes a preemption criterion to deport running best-effort services. Different preemption criteria are tested to study the impact and disturbance to running/deployed best-effort services. To reduce the side effects that preemption-based placement scheme has on best-effort services, we introduce a preemptive prediction-based placement scheme (PPTS). Building on the significant reduction in premium services rejections that PSVS achieved, we utilize a preemption criterion for the remaining rejection cases to eliminate rejections and cause less disturbance to deployed best-effort services.

Chapter 7 presents a summary of the topics addressed in this thesis and discusses future research directions.

Chapter 2

Background

"CSPs missed the Cloud Revolution! Edge is 4X & will hit $4 Trillion Economy"

Arpit Joshipura, GM LFN

2.1 Edge Computing Overview

As defined by the Linux Foundation's *State of The Edge* report [17], edge computing is *"The delivery of computing capabilities to the logical extremes of a network in order to improve the performance, operating cost, and reliability of applications and services. By shortening the distance between devices and cloud resources that serve them, and also reducing network hops, edge computing mitigates the latency and bandwidth constraints of today's Internet."* Edge computing pushes the centralized cloud infrastructure closer to the network edge. Edge computing is not meant to be a single layer between end devices and the core cloud. Rather, it is a continuation that extends from where the core cloud infrastructure ends to the edge of the network, and

in some use cases extend to the consumer's premises.

Currently there are three edge computing paradigm proposals: Fog Computing [31], Cloudlets (micro data centres) [103], and Multi-access Edge Computing (MEC) [45, 108]. The main idea behind the different edge computing proposals is to bring the cloud closer to where the end devices are; that is why edge computing paradigms are considered next-generation cloud computing [112]. Alliances between academia and industry have formed to promote and encourage the adoption of different edge computing paradigms, by providing use cases and extending some of the cloud platforms to serve the new edge computing requirements. Open Fog Consortium for Fog Computing, Open Edge Computing for Cloudlets, and European Telecommunications Standards Institute (ETSI) manages MEC via the respective Industry Specifications Group (ISG), are the alliances formed to date.

2.1.1 Edge Computing Paradigms

A cloudlet is a trusted single layer of resource-rich server or a cluster of servers connected to the internet and available for use by nearby mobile devices. In such architecture, services are instantiated at a nearby cloudlet, then users can start accessing services via a one-hop wireless connection [103]. Fog computing is proposed as a continuum from cloud to things. This continuum fills the gap between cloud and Internet of Things (IoT) devices, in an N-Tier architecture. To date, there is no agreed-on definition of what a fog node is [82]. Open fog consortium, now Industrial Internet Consortium (IIC), and their technical partners constructed a detailed system architecture for fog nodes and networks [13]. ETSI MEC framework defines the general entities involved at the system, host, and network levels. The reference

architecture shows the functional elements that comprise the edge system and reference points between them [45]. MEC uses commodity servers deployed at the edge of the Radio Access Network (RAN) to provide the promised 5G ultra-low latency, user experience and cognition continuity, and reliability for mobile, IoT, and Industrial IoT (IIoT) applications/services [110]. Fog, Cloudlets, and MEC can complement one another. For instance, fog can make use of cloudlets as fog nodes [42,82] and Fog RAN is one of the proposals for Fog deployments in the RAN.

Edge computing means different things to different stack holders. To tackle this ambiguity, the Linux Foundation's "State of the Edge" project has developed a widely accepted taxonomy of edge types [17], see Figure 2.1. Regardless of the type, edge computing has two main components, edge infrastructure and edge software stack and networking.

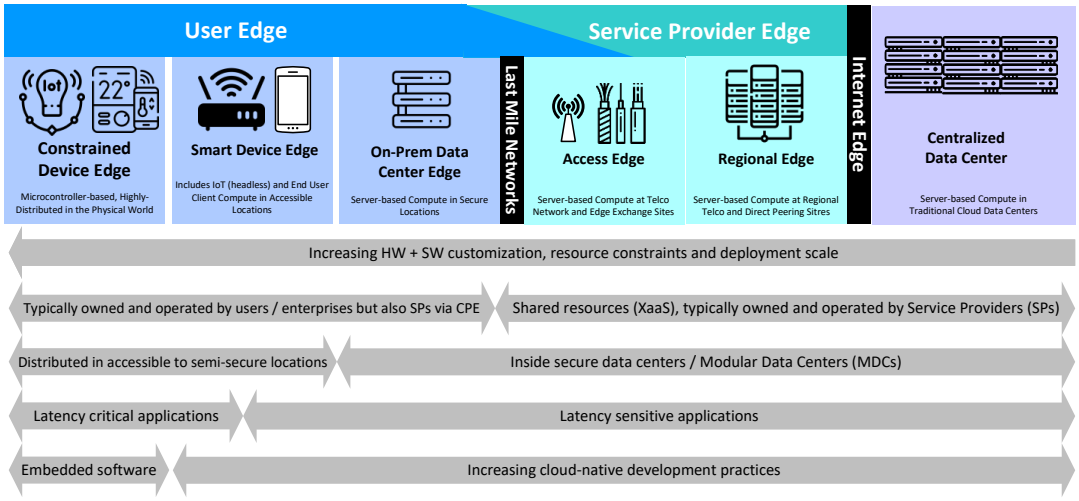


Figure 2.1: Types of edge in edge continuum, reproduced with permission from [17].

Edge Infrastructure

The diversity of use cases addressable by edge computing has resulted in a wide range of edge hardware that will continue to diversify [17]. Edge is a distributed cloud and most cloud hardware vendors will add edge to their portfolio; however, the form factor will be diverse to match the diversity of edge use cases. Edge is to support a continuum of requirements that extends from where the cloud ends all to users' end devices. As such, edge hardware is to range from regular sized server racks in the Central Office (CO) to a smart meter/camera. For example, micro edge data centers, such as those attached to light poles and street-side cabinets, are especially suited for smart city and IIoT use cases.

Edge Software

Specialized edge software is required for application delivery, managing edge hardware, and moving workloads around edge locations [17]. Edge software is a complete stack that is expected to benefit from the best practices learned from hyperscalers, IoT, web-scale service provisioning, and Content Delivery Network (CDN). As will be shown in Figure 2.10 in Section 2.3, the Linux Foundation Edge (LF-Edge) stack is an example of an edge stack that spans vertically through infrastructure and application layers and horizontally spans different types of edge [74]. On the one hand, some edge workloads can run in the cloud and at the edge. On the other hand, edge-native applications are a class of software that cannot operate without the edge, such as wearable cognitive assistance which utilizes both Augmented Reality (AR) and Artificial Intelligence (AI) [17].

2.1.2 Importance of Edge Computing

Edge Computing is no longer a hype; it is a reality and most network operators and enterprises are investing in either building their own edge infrastructure, provide edge offerings or planning to integrate and utilize edge computing as an integral part of their operations. The edge computing market is projected to grow from $36B in 2021 to $87B in 2026 [83]. From 2019 to 2028, a cumulative Capital Expenditure (CAPEX) of up to $800B is expected to be allocated to edge infrastructure [17].

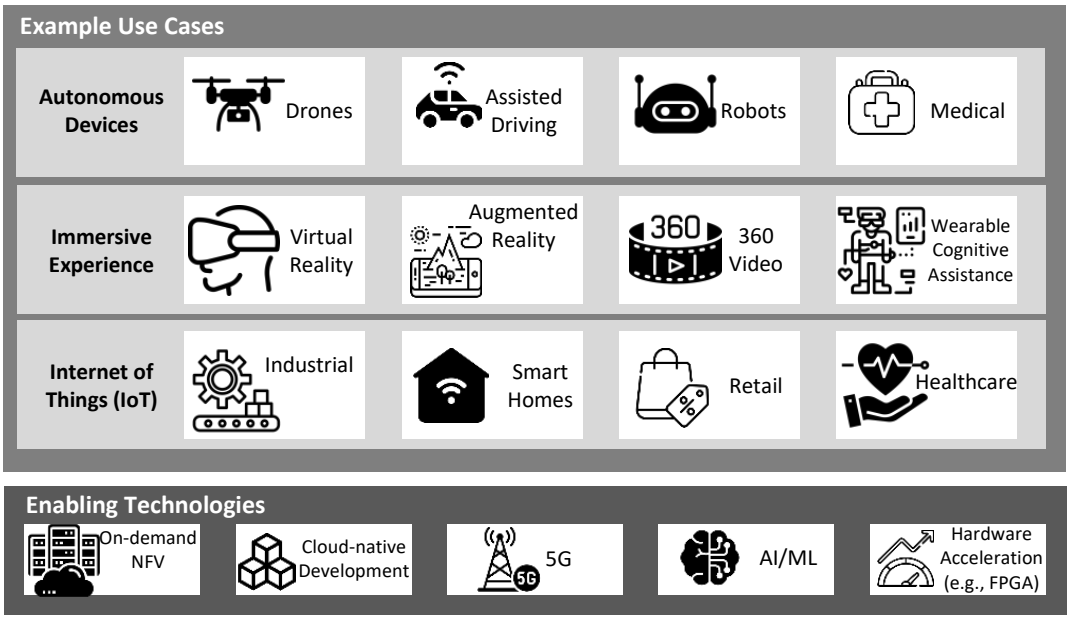


Figure 2.2: A sample of edge computing use cases, reproduced with permission from [74].

The phenomenal growth of edge computing is attributed to the roll-out of 5G and other communication business use cases, see Figure 2.2. The next-generation cellular network 5G requires innovation both on the network core and the radio sides.

The network core is fully virtualized utilizing the Network Function Virtualization (NFV) [44]. Also, network core functional modules are redesigned to make scaling and provisioning more agile and flexible. Control and User Plane Separation (CUPS) is a step towards achieving such agility. Control plane functions are split and kept central in the core, while the user plane functions are pushed closer to end users. This goes hand-in-hand with the service-based architecture (SBA) adopted by the 3rd Generation Partnership Project (3GPP) for the 5G core and the adoption of cloud-native principles [53]. Scaling out, up and down, will provide more flexibility for control and user plane functions, independently. Pushing the virtualized user plane functions closer to end users requires a distributed compute infrastructure, management, and orchestration mechanisms, provided by edge computing platforms. On the radio side, the same edge computing is required to host the Open RAN (O-RAN) disaggregated and virtualized components Distributed Unit (DU), Centralized Unit (CU), and near real-time RAN intelligent controllers (nRT-RICs) [93]. The disaggregated RAN functions must be provisioned with stringent time constraints [63].

Even though Telecom use cases represent the major driver behind the development of edge computing, Virtual Reality (VR)/AR/Mixed Reality (MR) and autonomous and connected vehicles are the leading use cases for prospective edge service providers. However, the Communications Service Providers (CSPs) focus will be on optimizing the network cost [115]. Edge computing is not only for CSPs but also for enterprises, Over-The-Top (OTT) service providers, and cloud providers. Edge computing will be an integral part of the future internet, which is why network operators and cloud providers are keen to have a share in that future. Edge is mainly characterized by location and last-mile connectivity. As shown in Figure 2.3, the same service

is deployed in both a CO (green) and an AWS CloudFront location (red). For the same user equipment (UE), because it has one-hop access to the green service it is considered an edge service. Despite its physical proximity to the UE, the red service is a core cloud service because the cellular operator cannot directly access the fixed broadband assets. Even though the AWS CloudFront is considered an edge for the devices connected via fixed broadband, it is considered a core cloud for cellular UE.

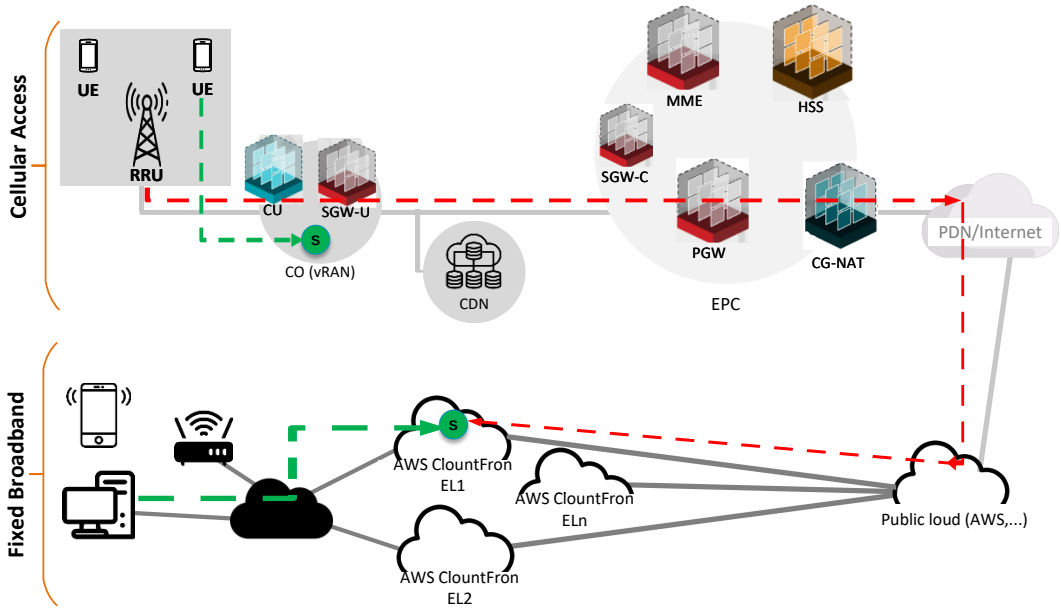


Figure 2.3: Importance of both location and access/last-mile connectivity.

On the one hand, cloud providers (hyperscalers) are deploying distributed local data centers/clusters in metro areas, intending to neutralize network operators access network ownership advantage [99]. On the other hand, utilizing their unique location with natural distributed existence and the ability to support mobility, network operators are utilizing NFV, Software-Defined Networking (SDN) [51], and cloud-native technologies to cloudify and transform their access network. With the cloudified

access network, network operators will be able to host their own Virtual Network Function (VNF) as well as third-party/enterprise workloads. Which explains why most CSPs are counting on edge as a key to monetize 5G and is regarded as their beach-front property [60]. Both network operators and cloud providers are missing parts of the full edge picture and no single side can provide edge services without collaborating with the other side. Over the past year, many MEC collaborations between CSPs and hyperscalers have been announced [24, 25, 113].

2.2 Enabling Technologies

Edge computing itself is an enabler for most 5G use cases. Edge computing just like cloud computing is powered by virtualization technologies such as hypervisor-based virtualization (Virtual Machine (VM)) and OS-based virtualization (containers). VNFs and their rising successor Cloud-Native Network Function (CNF) are the main deployment block over edge infrastructure. Most enterprise and network services consist of component functions/VNFs forming a Service Function Chain (SFC) and traffic should traverse these functions in a specific order. Finally, and despite not being mandatory, SDN complements NFV and provides a single pane of glass management, resulting in smoother and cheaper operation.

2.2.1 NFV and SFC

The monotonic increase in Operation Expenditure (OPEX) and CAPEX are driving CSPs to adopt innovative solutions and technologies to either stifle the increase or reduce the OPEX and/or CAPEX. Moreover, CSPs are facing fierce competition from OTT service providers, e.g. Netflix and YouTube, and cloud service providers,

which resulted in a continuous decline in both Average Return Per User (ARPU) and Return on Invested Capital (RoIC), shown in Figure 2.4. Even worse, to cope with the increasing traffic demands and to retain their current customer base, CSPs must continuously roll out services and yet be able to achieve revenue to remain competitive. However, rolling out services is not cheap and is usually coupled with increased CAPEX and OPEX.

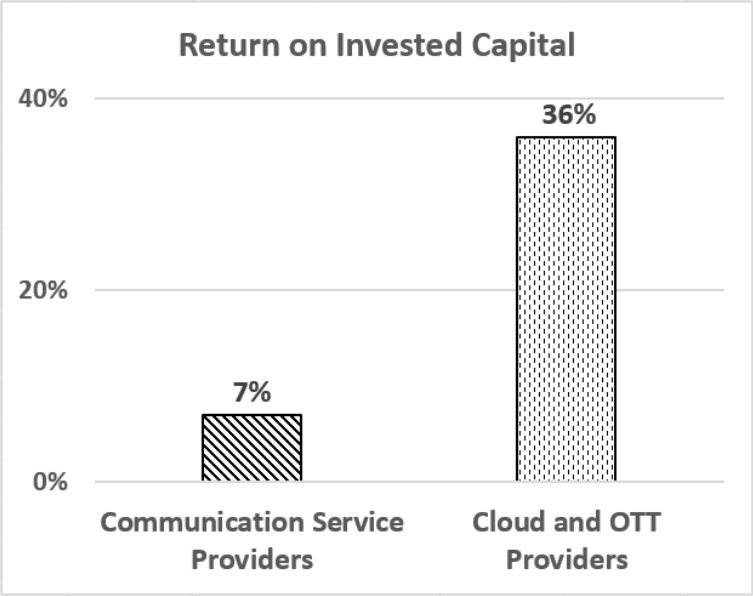


Figure 2.4: ROIC: Telecom Providers vs Cloud and OTT Providers ©Dell Technologies.

In 2012, to solve the dilemma, six of the world's leading CSPs, with the help of ETSI, introduced the NFV call to action [44]. Disaggregation is a key in the network transformation journey that started in 2009 with the introduction of SDN. By decoupling/disaggregating the network function software from the specially-built hardware, NFV enables the network function software to progress separately from the hardware and vice versa. NFV disrupts and introduces many differences in how services are provisioned. The instantiation of network function will be easily automated by utilizing

already available and mature cloud and virtual network technologies.

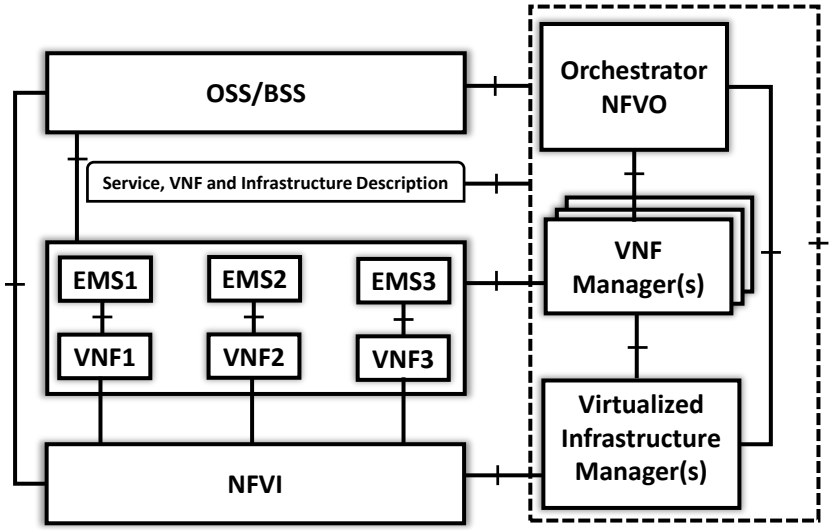


Figure 2.5: ETSI NFV Architecture Framework, reproduced with permission from [47].

ETSI's NFV ISG is responsible for creating standardization documents to give CSPs and vendors a reference NFV architecture that can be used to develop platforms and seamlessly interoperable tools. Since its inception, the NFV ISG has produced four releases of NFV specifications, the fourth release targets enhancing the NFV framework and providing support for 5G and novel fixed access network deployments. The most important document is the NFV Architecture Framework shown in Figure 2.5 [47]. The ETSI NFV architecture consists of four components, Network Function Virtualization Infrastructure (NFVI) and the related Virtualized Infrastructure Manager (VIM), the Management and Orchestration (MANO) module, and the VNFs. The architecture defines and standardizes the interfaces between different components, those founded by the architecture, (e.g. VIM and MANO), and the already existing conventional components such as the Element Management

System (EMS) and the Operation Support System (OSS)/Business Support System (BSS). NFVI represents all hardware and software components that represent the environment on top of which VNFs will be deployed and managed [47]. NFVI could span single or multiple sites/locations called NFVI-points-of-presence (NFVI-PoPs). A virtualization layer is required to abstract the hardware resources and disaggregate VNFs from the underlying hardware.

VIM is responsible for controlling and managing the interaction between the VNFs, Network Function Virtualization Orchestrator (NFVO) or Virtual Network Function Manager (VNFM) and the NFVI different compute resources. On the one hand, VIM is responsible for managing the resources by monitoring the available inventory of resources belonging to different hypervisors, resource allocation to VNFs VMs, resource reclamation and energy efficiency. On the other hand, VIM reports to north bound components, (e.g. MANO), regarding the status of deployed and running VNFs, analysis of performance issues causes, and infrastructure fault information.

The VNFM manages VNFs life-cycle management, from instantiation to termination going through migration and replication. A VNFM can serve single or multiple VNFs. NFVO manages the life-cycle of network services (SFCs). The orchestrator interfaces with VNFM to follow the status of deployed VNFs and takes corrective actions as required. The orchestrator also interfaces and communicates with OSS to receive network services deployment requests and with BSS updates billing and other business-related information. The trigger for deploying network services comes from the OSS by sending the service descriptor. The service descriptor is typically written in one of data modeling/orchestration languages such as the YANG modelling language used with Internet Engineering Task Force (IETF) Network Configuration

(NETCONF) Protocol or the Topology Orchestration Specification for Cloud Applications (TOSCA) language. ETSI hosted Open Source Management and Orchestration (OSM), an open-source project that uses NETCONF YANG [96], and TOSCA is used with Linux Foundation Networking (LFN) Open Network Automation Platform (ONAP) project [75].

Service Function Chaining (SFC)

Network, enterprise and OTT services consist of a set of functions which are more useful if not confined and restricted by location. Taking advantage of the agility that NFV introduced, enables service providers to deploy service functions wherever and whenever needed. SFCs provide the ability to define an ordered list of physical and virtual service functions. These service functions/VNFs are stitched together to form the service chain. Service chaining is a set of processes and technologies that CSPs utilize to softwarize network services configuration [33]. Unlike the pre-NFV era, software-based service chaining provides the agility to dynamically design and deploy SFCs. Sometimes SFC is referred to as the SDN version of Policy-based Routing (PBR) [16].

IETF has put together an SFC architecture, shown in Figure 2.6, that applies to a single network administrative domain [59]. The architecture consists of three main components, the classifier, control plane, service functions and the Network Service Header (NSH). The classifier receives and identifies if a traffic flow should be forwarded to the service chain processing path. Control plane is responsible for setting up the chain path. Classifiers are deployed at both ends of bidirectional chains to route ingress and returning traffic. NSH is a data-plane encapsulation used

to determine the service functions/VNFs that make up a chain. NSH defines a new data-plane protocol that adds an extra header between the transport layer header and the network/internet layer header.

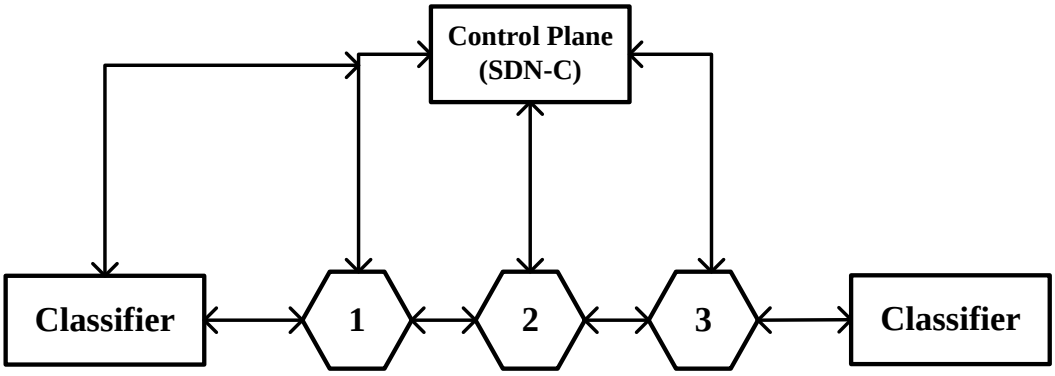


Figure 2.6: Service Function Chaining Architecture, reproduced from [59] per the IETF Trust Legal Provisions (TLP).

There are many example use cases of using SFCs both in mobile cellular networks and in data centers [58, 68]. The user traffic, in mobile networks, is tunnelled and terminates at the Packet gateway (P-GW). Usually the IP traffic is not directly forwarded to the application platform such as social network platform, but is steered through an SFC. Doing so, CSPs are distinguishing their services to their customer base, which tells how crucial SFCs are to CSPs business.

CSPs utilize different categories of SFCs, see Figure2.7:

Packet inspection: used to protect CSPs' network as well as subscribers' privacy. Service functions used in this category are: Access Control List (ACL), Intrusion Prevention System (IPS), Intrusion Detection System (IDS), Firewall (FW), encryption/decryption, and others.

Traffic optimization: used to guarantee the contracted Quality of Experience (QoE) and uses Performance Enhancement Proxy (PEP), such as TCP and video optimizers, video transcoding, traffic shaping, and Deep packet inspection (DPI).

Protocol proxies: functions that are used for certain purposes, such as Carrier-grade NAT (CG-NAT), DNS cache, HTTP proxy/cache, parental control, session border controllers (SBCs), and Malware detection and elimination (MWD).

Value-added services (VAS): such as Ad insertion, header enrichment, WAN acceleration, URL filtering, and parental control.

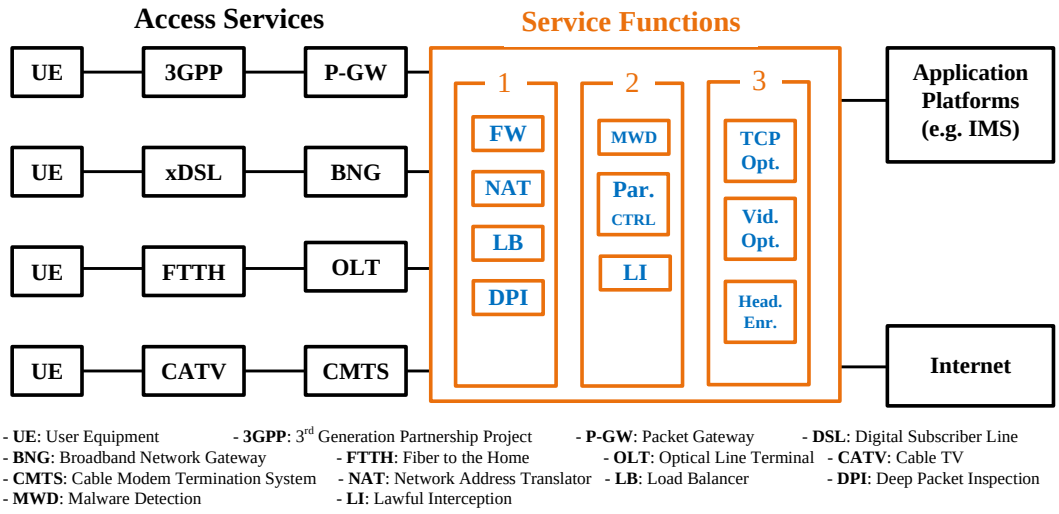


Figure 2.7: Various end-to-end carrier networks and service functions sorted into categories 1, 2 and 3, reproduced from [58] per the IETF Trust Legal Provisions (TLP).

In data centers, SFCs can be categorised in two broad categories, access SFCs and Application SFCs. Access SFCs are designated to service entering and leaving traffic

from data center while the traffic targeting applications is served by application SFCs, see Figure 2.8.

As can be seen in Figures 2.7 and 2.8, both mobile network and data center use cases are mostly sequential SFCs (without branching). As such, our focus in this thesis will be on the placement of sequential SFCs. The realization of SFCs is a two-step process, first, the service functions are instantiated and deployed, and second, a mechanism is used to configure the forwarding plane devices to steer the SFC traffic in the specified order as described by the SFC descriptor. Two mechanisms are used for SFC traffic steering, either using the IETF NSH [100] or an SDN-based method that utilizes port pairs, port pair groups and port chains [16]. On the one hand, NSH necessitates that forwarding devices (classifier or service function forwarder) in the service domain be aware of the new header to parse and process the NSH header properly. On the other hand, NSH can exchange meta-data along the service chain path as well as provide a proof-of-transit for packets in virtualized environments [32]. The SDN-based approach, aka bump-in-the-wire, does not require any added headers and hence any SDN-aware forwarding device can handle and correctly forward the SFC traffic.

NFV Drawbacks and Problems

Since its inception, NFV technology has faced many challenges and problems that resulted in a decline in NFV adoption [40, 89]. The main reason is that CSPs were utilizing NFV as a way of running specially-built network appliances in VMs, as a result of vendors packaging the conventional network functions in VMs. In this transformation, CSPs overlooked the operations side in hopes of utilizing their legacy

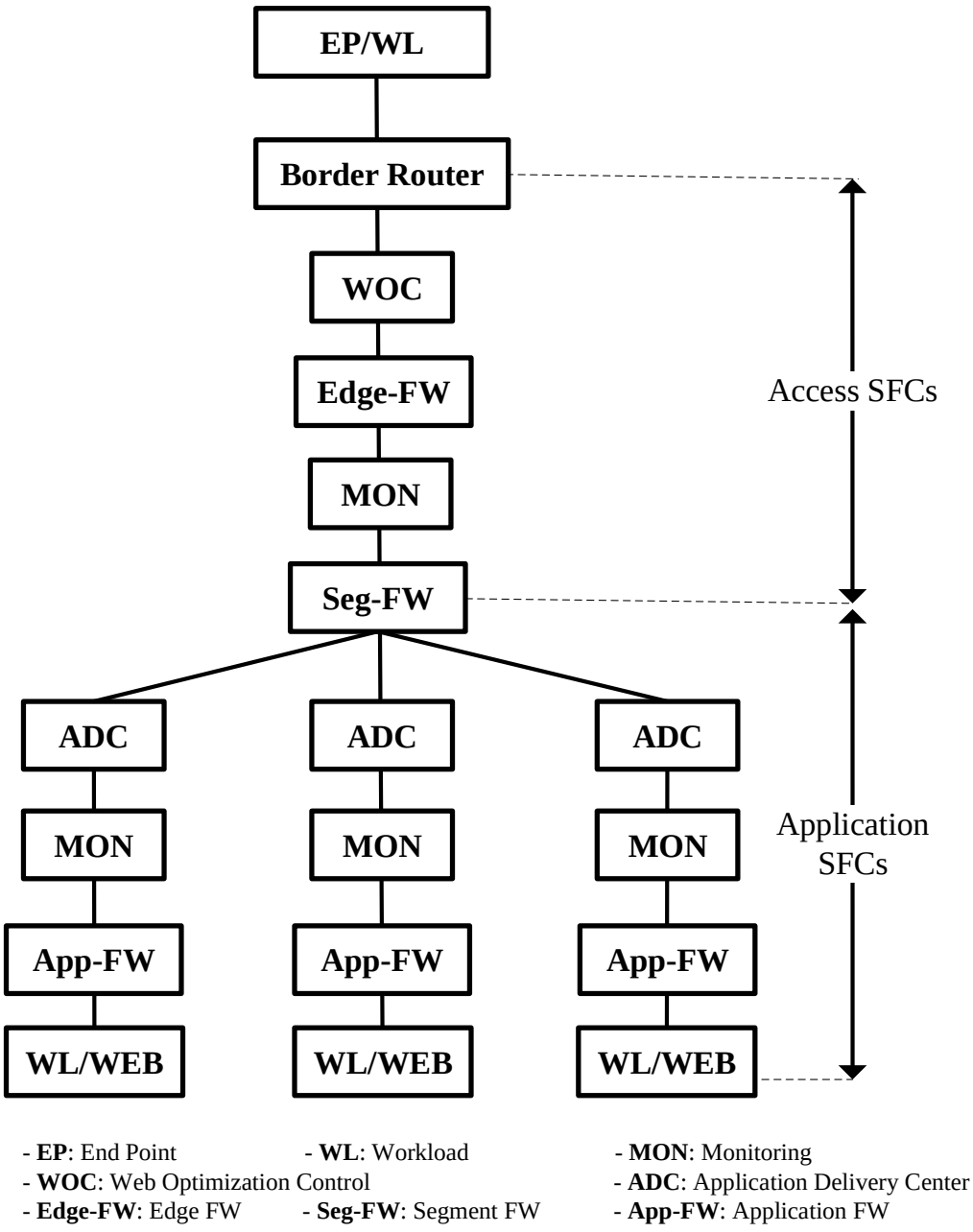


Figure 2.8: Service function chains in data center, reproduced from [68] per the IETF Trust Legal Provisions (TLP).

management and operation; this resulted in difficulties in streamlining operation and zero-touch management [72]. Provisioning SFCs has also been challenged due to restrictions on the number and variety of VNFs in SFCs [52]. The main reason for this drawback is that back in 2012, when operators presented the NFV call to action [44], most cloud-native tools such as Kubernetes, Helm, Terraform and others were generally unavailable. As a remedy, a corrective action has started to turn the wrong direction CSPs started building a *"Teclo cloud"* to the right direction of building a *"Cloud-native Telco"* [72]. That is to adopt a complete cloud-native paradigm and technological tool chain; instead of lifting the network functions software from their specially-built hardware, package them in virtualized containers, and build the Telco cloud to host them. In April 2022, Google and LFN announced the Nephio project that envisions providing cloud-native Kubernetes (K8s)-based carrier-grade platform that simplifies the VNFs provisioning over large-scale edge deployments [2]. Nephio is planned to provide scalability, reliability, and efficiency for all network operations based on true cloud-native automation.

2.2.2 Software-Defined Networking (SDN)

The term Software-Defined Networking was coined in an MIT Technology Review article in 2009 [55]. However, network programmability started before in 1995, with active networks which proposed including the forwarding instructions in a capsule (in-band programming) or using out-of-band programmable switches and routers [51].

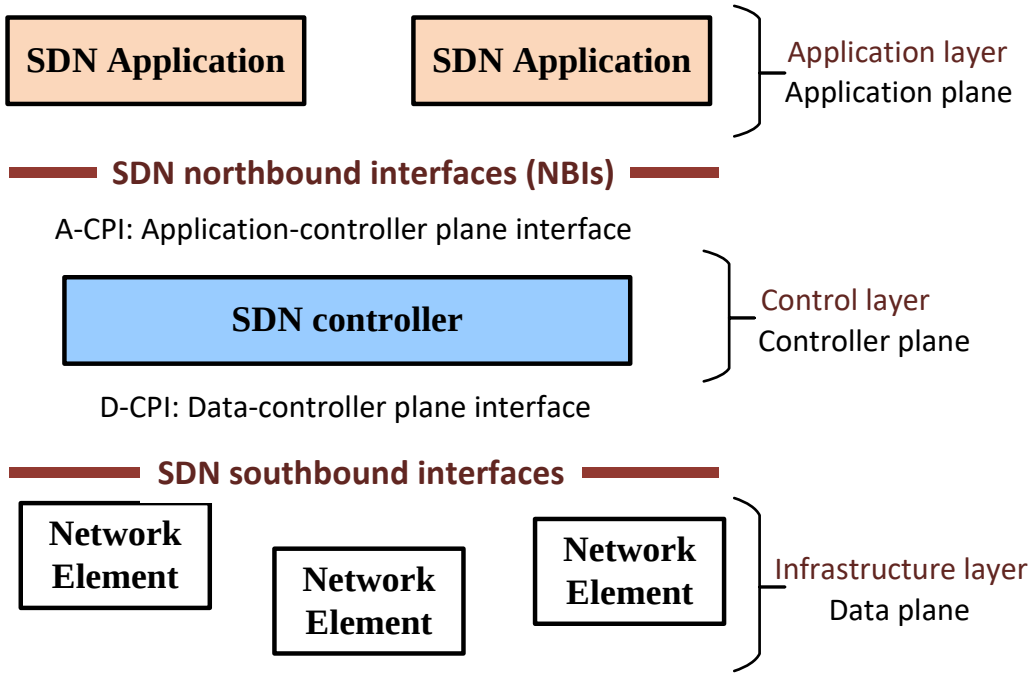


Figure 2.9: Basic SDN components.

In 2008, the control/configuration protocol OpenFlow was introduced in [84], as an experimentation tool for researchers without having the network equipment vendors expose the details of their switches. The authors realized the potential that OpenFlow opened and how it can be used to control and define data flow using software and hence came the SDN concept. SDN makes the network/internet programmable by disaggregating/decoupling the control plane from the data plane and consolidating the network control software in a logically centralized Highly Available (HA) way. The SDN Controller (SDN-C) is where all network control logic is consolidated, as illustrated in Figure 2.9 it has two standardized interfaces, the Northbound Interface (NBI) and the Southbound Interface (SBI). The NBI is to communicate

with SDN/network programming applications and is typically done using RESTful API calls. The SBI is used for communication between SDN-C and data/forwarding plane switches and routers. Several protocols have been used and standardized for SBI, such as OpenFlow, NETCONF, Open Virtual Switch Database (OVSDB) and Cisco's OpFlex. With SDN, the automation of network control, management and operation is much easier and less error-prone. The SDN-C is continuously updated with the status and topology of data/forwarding plane elements. The SDN applications need only to request high-level intents/goals, such as packets from source IP address X to destination IP address Y should be dropped. The SDN-C uses its topology knowledge to translate these high-level intents into low-level forwarding rules and install them into the respective switches. Indeed, SDN is not a necessity when it comes to adopting NFV, however, the integration of SDN in an NFV environment has many benefits, the most valuable is inheriting the flexible programmability of forwarding elements. In an NFV architecture, the SDN-C can be deployed part of the VIM, virtualized and deployed as a VNF, part of the NFVI, part of the OSS, or as a conventional Physical Network Function (PNF) [46].

Several SDN industry consortia and initiatives were formed, Open Networking Foundation (ONF) [12] and Open Daylight (ODL) initiative [10], then many more network vendors and operators joined. Since its inception, SDN has gained popularity in many industries (cloud providers and Telecom providers and vendors) with many successful realizations like Google's wide-area traffic management system [62] and Nicira's network virtualization platform [92]. Progression on the SDN side has never stopped and continued at a fast pace. For example, ONF launched several new SDN projects like Stratum, μONOS and the next generation SDN (NG-SDN) [11]. The

SDN pattern/design philosophy has been adopted in many technological advances such as Software-defined WAN (SD-WAN) and Software-defined RAN (SD-RAN).

2.2.3 Edge Key Requirements and Challenges

In addition to the requirements and challenges facing cloud service providers, due to its distributed nature, edge computing service providers are tasked to address additional requirements. The main requirements an edge computing platform/offering should fulfil are:

- **Efficient resource utilization:** With the rising demand on edge resources, efficient resource utilization will play a crucial role in edge computing offerings.
- **Edge computing resources distribution:** Edge computing location should extend beyond cloud computing and data centres closer to end users and data sources. To provide the promised high performance and low latency, edge locations should be deployed on a large-scale but be small-sized.
- **Converged edge platform:** Edge sites should be a converged platform of compute, storage, networking, and application resources. Which will enable edge sites to provide the edge intelligence that covers use cases requiring real time, optimized, secured and private, and localized processing.
- **Zero-touch provisioning and automation:** Edge sites/locations, will be vastly distributed and hence automating the infrastructure monitoring and trouble shooting, service provisioning, self-healing, upgrading and remote management and operation is necessary to have.

2.3 Edge Computing Platforms

Since edge computing is a distributed version of cloud computing, it is normal to start with the same virtualization technologies that has been used for a decade in cloud computing, in particular VMs or hypervisor-based virtualization. Most early edge computing platforms, adopted the same virtualization technology along with its tool chain. Openstack is the most adopted open source Cloud Management System (CMS). With the introduction of micro-services architecture for designing cloud-hosted applications and utilizing the containers as a lighter packaging compared to VMs, the world witnessed the rise of containers and container orchestration engines, such as K8s [41]. Since the introduction of K8s and service mesh among other tools, the cloud-native term was coined to refer to the culture and design principles of applications designed to be hosted and operate, specifically in the cloud. Containers are the formal best virtualization/packaging mechanism for cloud-native applications. Since edge computing sites are resource limited compared to cloud data centers, the cloud-native ecosystem is the best fit for developing edge computing platforms and workloads. As a result, most of the edge computing platforms are based on container/OS-based virtualization and utilize K8s for orchestrating platforms' micro-services and containerized workloads/edge-native applications. It is worth mentioning that the next wave of VNFs will be the CNFs. CNF is a cloud-native application that implements network functionality. Each CNF consists of one or more micro-services and is deployed and managed using cloud-native principles.

LF-Edge is an umbrella that hosts a variety of edge platforms that fulfil several variations of edge use cases [3], with the aim of building open, interoperable framework for edge computing, see Figure 2.10. Some platforms are tailored towards home

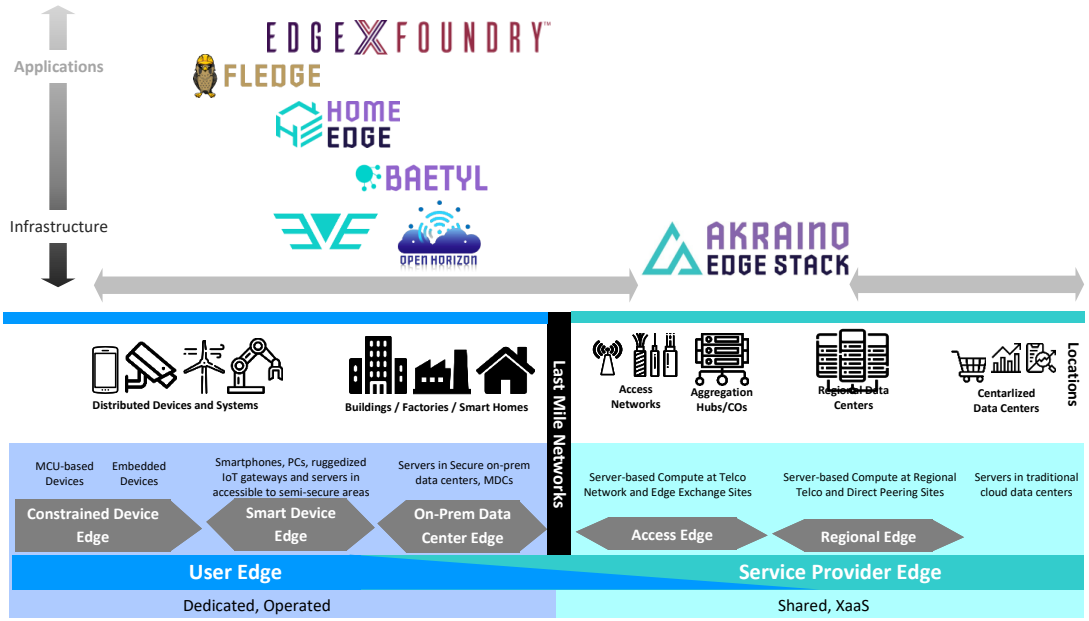


Figure 2.10: LF Edge stack spanning different edge types and covers infrastructure and application layers, reproduced with permission from [74]

and IoT edge solutions, like Baetyl [5], Home Edge [8], and EdgeX Foundry [6]. Fledge is another project that is focused on industrial and mission-critical use cases. Fledge project [7] works and collaborates with another LF Edge project EVE [9]. Fledge is also integrated with the LF Edge flagship project Akraino Edge Stack [4]. Launched in 2018, Akraino project targets edge solutions for Telco, enterprise, and IIoT, and has many blueprint families, each is a declarative configuration of a complete end-to-end edge stack and addresses specific edge use cases with common/relevant specifications and requirements. With the seed code submitted by WindRiver, StarlingX is a fully integrated edge cloud stack that addresses the needs of various use cases [14]. Many general purpose platforms are not tailored to specific set of use cases, they provide

the required software stack for managing the infrastructure of distributed edge locations, each with resource-limited devices such as Raspberry Pi. Rancher K3S [15], is a lightweight K8s distribution built for IoT and edge computing that require a fully-contained cluster at the network edge. KubeEdge [1], is a K8s native edge computing framework that has its control plane located in the cloud and can manage more than one edge location. Virtual Kubelet [18], is positioned between K3s' fully-contained edge clusters and KubeEdge's cloud-based control plane,thus providing a hierarchy of edge servers/locations and controllers. Virtual Kublete enables edge providers to extend their operation and collaborate with external infrastructure providers including user devices.

2.4 Service Provisioning

At the edge, the service provider is closer to the user and has the visibility and the granularity which are not always available at the core cloud. Taking actions to enhance the quality of experience of a user is much easier, more feasible and can be done at a millisecond scale. Compared to the core cloud where all users look the same, taking action is more likely to be seconds after the triggering event took place. Therefore, the edge is the place where service providers can instantly control the quality of the services delivered. Also, it is the place to accurately and properly manage and enhance the user experience.

2.4.1 Service Placement and Resource Allocation

When it comes to service provisioning at the edge, numerous factors are involved such as, optimal placement of services, resource allocation for services [80, 116, 120],

service migration [80,116], service replication [48], user mobility, continuous end user and service monitoring, continuous infrastructure monitoring, to name a few. While most of these factors seem to be independent of each other, they are closely related and affect each other. For example, service placement is dependent on resource provisioning and allocation. Also, service placement, replication, and migration are closely related. User mobility and the ability to profile and then predict user mobility affects placement, replications, and later migration decisions. Service placement algorithms must consider the agreed-on Service Level Agreement (SLA). Furthermore, placement algorithms should consider the trade-off between the SLA requirements and resource utilization (the cost) [30]. Most of the proposed placement algorithms, viz. [19,23,29,94], formulate the placement as an optimization problem and use either integer linear or dynamic programming. A game theoretic algorithm could also be used to produce a solution that addresses the trade-off between SLA requirements and resource utilization/cost.

Since the operational environment is more dynamic at the edge compared to the core cloud, services should be designed for failure, be robust and more reliable [34]. The micro-services architecture is gaining popularity as a software architecture for cloud-hosted applications [41].

Service orchestrator manages the life-cycle of different components that comprise the main service and ensures that the main service is continually available, even is a failure is detected [114]. For example, in reference [30], the authors propose provisioning CDN as a Service (CDNaaS), by decomposing the CDN into four services: virtual transcoders, virtual streamers, virtual caches, and CDN-slice coordinator. Treating the four services as VNFs and utilizing the network slicing technology, they

were able to provision the CDNaaS service over an isolated distributed network of edge nodes across a multi-cloud domain.

According to the work in [111], there is an inevitable convergence of NFV, 5G, and edge computing. On the one hand, the ETSI MEC is considered to be one of the enabling technologies to the next-generation 5G mobile network. On the other hand, ETSI also introduced the NFV to convert the networking infrastructures into commodity servers with VNFs deployed on top. MEC, NFV, and SDN [37], are the technologies through which network slicing is enabled [21, 90]. In addition to NFV and SDN, edge computing, especially MEC, is an integral part of network transformation. Edge computing extends the NFVI to the access network and complements the continuum from COs or Evolved Packet Core (EPC) the whole way to RAN and can extend further to Consumer Premise Equipment (CPE).

2.4.2 Service Migration and Replication

Migration of workloads hosted in VMs and/or containers is an example of a solution that is to be customized to fit the edge's distributed nature. Typically, in the cloud, resource utilization/consolidation and performance requirements are the main triggers for migration, while at the edge, latency, user mobility and data privacy are additional triggers. Moreover, resource-constrained edge nodes and bandwidth-limited links, add challenges to the migration process. At the edge, take MEC as an example, user mobility represents the main challenge that degrades service quality, even in small set-ups [49, 50, 57]. To guarantee service continuity and satisfy service performance requirements, migration is how the service providers can fulfill the performance requirements.

Since the introduction of NFV, many research papers have addressed the service migration and replication problem from different perspectives, with different assumptions, and with many overlooked realistic details. While most of the papers acknowledge the resource-constrained nature of edge servers, the majority are proposing service replication (over-provisioning) as a form of proactive migration [48–50]. Also, some papers, for example [49], are considering a one-to-one cardinality of a user-service relationship. Indeed, triggering a mobile call hand-over is done on a single user basis, this should not be the case when it comes to services at the edge. The number of replicas and the efficient placement of those replicas is another side to investigate, it was briefly discussed as a challenge in [49].

Migration techniques can be divided into two categories, reactive and proactive migration, as summarized in Table 2.1. Proactive migration is further divided into multiple replicas and prediction-based (predicting user mobility). Some factors are critical to the migration process and to which type of migration to choose. First, how the service/function maintains its state data. Stateful services, are those services whose state data is tightly coupled with the service itself. Consequently, state data migration is a crucial part of the service migration process and impacts the duration of service downtime until restarted in its new host. Second, whether the service/function is VM-hosted or container-hosted. Containerized services are more suitable for both reactive and multiple replicas proactive migration.

The authors of [28] propose a service migration of stateful Telecom services. They proposed to migrate a complete service chain by utilizing the service orchestrator and service functions managers. Also, they proved that decoupling the state from the

Table 2.1: Comparison of Migration Techniques & their Relation with Replication

		Reactive Migration	**Proactive Migration**	
			Multiple Replicas	**Mobility Prediction**
Utilization & Overhead		Efficient (least overhead)	Inefficient (Over-provisioning)	Moderate (prediction overhead)
Stateless vs Stateful services		Better with stateless	Good for both	Stateless is better for very short lookahead window
Down time		Stateful (considerable), Stateless (negligible)	Negligible	Depends on lookahead window length
Total migration time		Relatively long	Negligible	Should be negligible to work
Performance	**Containerized**	Promptly responsive	best fit	Better for short lookahead window
	VM-based	Longer migration time	Not practical (overhead)	Requires longer lookahead window

service logic, results in a smooth and more predictable migration time. Service replication as a proactive service migration mechanism is proposed in [50] which is valid only for container-hosted lightweight services. This service migration technique achieves almost zero downtime but at the expense of a slightly higher over-provisioning overhead. Migration based on user mobility prediction represents a compromise between the resource efficient reactive migration and the downtime efficient replication-based proactive migration. The success of this technique depends on the prediction accuracy, which in turn depends on the length of the lookahead window among other things.

Indeed, efficient resource utilization is critical for edge computing; however, we

found there is a shortage of papers addressing this aspect. As well, very few of the surveyed papers addressed sharing VNFs when deploying SFCs considering the varying traffic demand and operation dynamics. Moreover, none of the surveyed papers considered a practical priority-based SFC placement and the time urgency of time-critical premium services. As such, our research is focused on satisfying SFC requests the most resource-efficient way by utilizing VNF sharing and partial fulfilment of SFC requests.

Chapter 3

Improving Edge Resource Utilization

"NFV was Training Wheels,

Cloud-Native is the Real Deal"

Peter Worndle, Ericsson

3.1 Introduction

The next generation of mobile networks (5G) is expected to address the performance requirements of diverse use cases in different industries, massive machine-type communications (mMTC), enhanced mobile broadband (eMBB), and ultra-reliable low-latency communications (URLLC), are the categories of addressed 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) [44]. Control and User Plane Separation (CUPS) of the cellular core is a step to achieving service agility. Control plane functions are split and kept central in the core, while the user plane functions are pushed as close as possible towards end-users. Scaling out will be more flexible for control and user plane functions,

independently. This goes hand-in-hand with the service-based architecture (SBA) adopted by the 3rd Generation Partnership Project (3GPP) for the 5G core and the adoption of cloud-native platforms and tools [53]. On the radio side, Radio Access Network (RAN) components are disaggregated into Radio Unit (RU), Distributed Unit (DU), and Centralized Unit (CU). DU is installed at cell sites, and CU is pooled and deployed in Base-band Unit (BBU) Hotels, or Central Office (CO). Edge Computing is witnessing phenomenal growth [115] and is a major player in 5G and some enterprise use-cases. Edge computing will provide the required platform which will host the separated and virtualized user plane functions [111]. Likewise, edge computing secures the required infrastructure to host the open RAN (O-RAN) disaggregated and virtualized components: DU, CU, and near real-time RAN intelligent controllers (nRT-RICs) [93]. On both core and RAN sides, NFV, Software-Defined Networking (SDN), and edge computing are to play a principal role [111].

A virtualized infrastructure of Telco clouds extending from the network core to the perimeter of the access network, forms a layer that can be used to host delay-sensitive network services and applications of service providers. The layer could even be extended to the cell-sites. With the growing interest in delay-sensitive applications, such as Augmented Reality (AR)/Virtual Reality (VR), telehealth, online gaming, autonomous vehicles, and content delivery services, the demands put on the edge will be high and no doubt will continue to grow significantly as the field deployment of 5G networks progresses.

In 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. 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. Having access to such information would not be viable anywhere else; that is why network edge, especially MEC, is regarded as the monetization arm that Communications Service Providers (CSPs) will utilize to profit and cover the cost of upgrading their infrastructure and services. Edge 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.

Virtual Network Function (VNF)s that are assigned all required compute and bandwidth resources, are expected to operate at their full capacity. However, due to changing operation conditions, some VNFs might be underutilized, that is to receive and process traffic less than its full capacity. With SFC requests continuously arriving, it would be more resource-efficient to utilize/share the unused capacity of deployed underutilized VNFs first and only deploy a new VNF instance if no deployed underutilized VNF of the same type exists. As shown in Figure 3.1, there are three SFCs with different number of VNFs and the total number of CPU cores required per each SFC. SFCs may have common VNFs; for example, in Figure 3.1(b), sfc_1 and sfc_2 have V_3 in common, and V_4 is common between sfc_2 and sfc_3.

The *VNF sharing* should be done while being mindful of the SFC performance requirements. We assume that both transmission and processing delays are negligible. Therefore, we decide to use maximum end-to-end delay as the performance requirement that has to be satisfied.

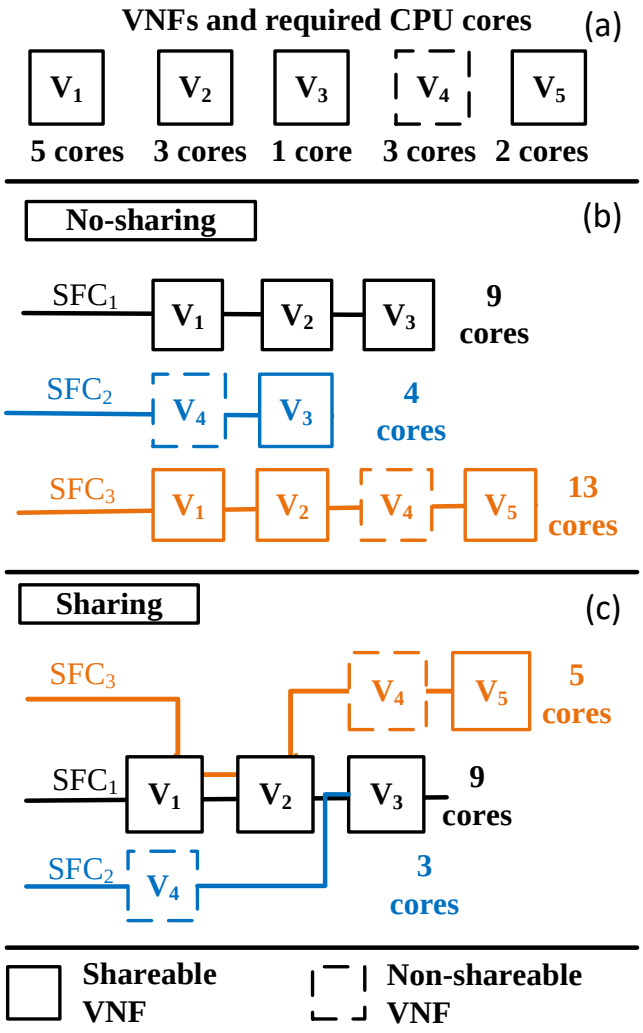


Figure 3.1: Compute resources savings when sharing VNFs among SFCs

To this end, we

- demonstrate the gain in efficient resource utilization and the number of satisfied Service Function Chain (SFC) requests by sharing VNFs across different SFCs.
- introduce *VNF sharing*-based SFC placement. We formulated the SFC placement and sharing as an integer linear program (ILP) model to demonstrate the *VNF sharing* gain. The objective is to minimize the total deployment cost, optimize resource utilization and satisfy QoS requirements/constraints.
- Compare the number of satisfied requests and average required resources per SFC request of our sharing-based placement against no-sharing placement.

The remainder of this chapter is structured as follows. In Section 3.2, we present related work. Proposed sharing-based SFC placement, network model, VNFs, SFCs and problem formulation are detailed in Section 3.3. Performance evaluation, simulation settings, simulations and results will be covered in Section 3.4, Section 3.5 summarizes chapter findings.

3.2 Related Work

With the introduction of NFV [44], service provisioning became more agile, and the placement of service functions/VNFs as the building block of SFCs started to gain traction. The wheel was not reinvented; researchers started by building on an already existing body of research on cloud computing, VM Placement (VMP), and Virtual Network Embedding (VNE). Due to the differences between VNFs placement and VMP [35], more research into the area was still needed.

Since 2012, considerable research work has been conducted; some are generic VNF/SFC placement [20, 27, 65, 67, 69, 79, 86, 94, 106, 117], while others are more into specific settings and use cases, like VNF placement at the edge-central cloud and service placement and replication in 5G edge [29, 48, 50, 64, 73]. The main goal in [73] is to minimize both end-to-end delay and deployment cost of mission-critical delay-sensitive service chains, and the SFC placement is formulated as a Mixed Integer Program (MIP) and further approximated using the tabu search algorithm. The work in [29] proposes VNF placement on the edge-central cloud in a way that optimizes resource utilization and satisfies QoS requirements using analytic queuing and MIP models. In [64], the authors propose optimizing the QoS of Cloud RAN (C-RAN) by dynamically configuring Remote Radio Unit (RRU) to proper BBU sectors according to the varying traffic conditions. For further details on VNF placement, [70] is a survey on VMP and VNF placement.

3.2.1 VNF Sharing

"*VNF sharing*" is one of the techniques used to reduce the cost and efficiently utilize resources. For example, as shown in Figure 3.1(c), 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% fewer resources.

We remark that in the literature, more than one term is used to refer to the *VNF sharing* concept. For example, the authors of [77] and [85] used the term *"multi-tenancy"* and *"VNF merging,"* in [76] the authors used *"VNF reuse"* (VM reuse),

task and request scheduling are used in [101] and [119], and the most common term was *"VNF sharing,"* used in [39, 81, 87, 88, 118].

There is an increasing interest in *VNF sharing* among CSPs and OTT service providers. For example, deploying evolved packer core (EPC) VNFs on the 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 [87]. In [39], the 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 [118].

In the majority of surveyed papers, the VNF/SFC placement is formulated as an Integer Linear Program (ILP) model. After demonstrating the bottom-line performance, a more practical heuristic placement algorithm is then presented. Sharing of VNFs by more than one SFC flow is triggered by the fact that some VNFs could be shared by SFC flows, such as anti-virus. At the time of this work, with the exception of [86], none of the surveyed papers considered sharing deployed underutilized VNFs while deploying new SFC requests. The work in [86] proposes sharing VNF among SFC flows based on a predefined number of flows that a VNF can handle. The fixed number of flows a VNF can handle does not reflect the changing operating conditions and will still leave some VNFs underutilized. To the best of our knowledge, the sharing/utilization of already deployed underutilized VNFs based on the currently

unused capacity in a way that adapts to operations dynamics was never proposed as a way of satisfying more SFC requests with fewer resources and yet satisfying the performance/QoS requirements.

3.3 Proposed Placement scheme

In our sharing-based placement scheme and upon the arrival of an SFC request, the priority is to use already deployed, underutilized VNFs of the same type as those in the SFC request at hand. At some point in the future, currently underutilized VNFs will get fully utilized even without hosting guest SFCs; hence, any sharing mechanism should take this into consideration. However, for demonstrating the benefits of sharing-based placement, we assume that currently underutilized host VNFs will remain so until the guest SFC(s) are concluded. We do not specify explicit start and termination points for SFC requests; rather, the first and last VNFs of each SFC are regarded as the source and destination, respectively. Moreover, we assume the existence of an SDN Controller (SDN-C), which will take care of configuring forwarding plane switches to forward traffic according to the SFC selected path. An SFC mechanism is assumed to be used in the CSPs' service domain, either using Network Service Header (NSH) or bump-in-the-wire technique that uses port-pairs, port-pair groups, and port chains to configure SFCs.

It is worth mentioning that we mainly address and support sequential SFC requests. However, to add the support of non-sequential SFCs to proposed placement scheme, some preprocessing steps are needed. First, decompose the non-sequential chain into two or more sequential chains. Second, do the placement of sequential chains. Finally, merge the individual placement solutions at the branching VNFs to

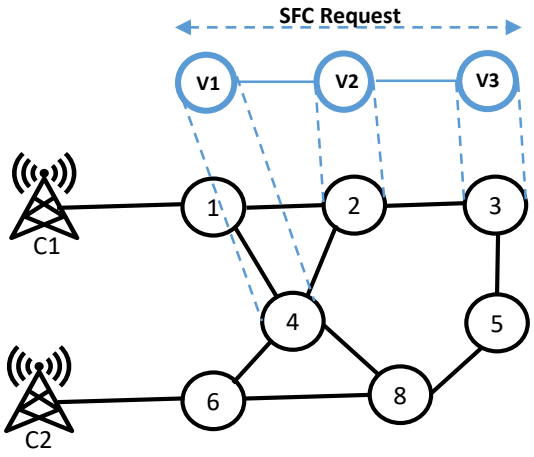


Figure 3.2: SFC request placement and the association of its VNFs with the substrate network nodes.

get the solution for non-sequential SFC.

In the rest of this section, we will explain substrate network model and problem formulation.

3.3.1 Substrate Network Model

The substrate network is modeled as a graph $G(N, E)$, where N is the set of nodes and E is the set of links. To be more generic, we modeled all our network nodes to be capable of hosting VNFs. The links between nodes are directional, as shown in Fig. 3.2, a link between nodes 1 and 2 means there are two links, one link from node 1 to node 2 and another link from node 2 to node 1.

Each node has its own compute resources, CPU cores and RAM. Each link has its bandwidth capacity as well as its propagation delay, which is a function of its length. When created, resources are assigned randomly to nodes and links. The substrate network topology is fixed and described by a connectivity matrix. The description of substrate network parameters is provided in Table 3.1.

Table 3.1: Substrate network parameters.

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 $L_{nn'}$

3.3.2 VNFs and SFC requests

An SFC request consists of an ordered list of VNFs. VNFs are selected from a list V of available already on-boarded VNFs. In the list of available VNFs, each VNF has a type, CPU and memory requirements, and the maximum traffic flow it can handle if assigned the resources required. Some VNFs, like firewalls, drop packets; in that case, outflow should reflect such dropping. The outflow will be equal to the inflow if a VNF does not drop or compress the inflow. As mentioned previously, some VNFs can be shared among SFC flows while others are non-shareable. $S(v_i)$ is a flag to determine whether VNF v_i is shareable or not. Take, for example, request sfc_j shown in Figure 3.3; it consists of 3 VNFs. The resources required as well as other parameters of each VNF, are shown (real numbers from our simulation). As we can see, the max-flow is proportional to the resources assigned to VNFs. For example

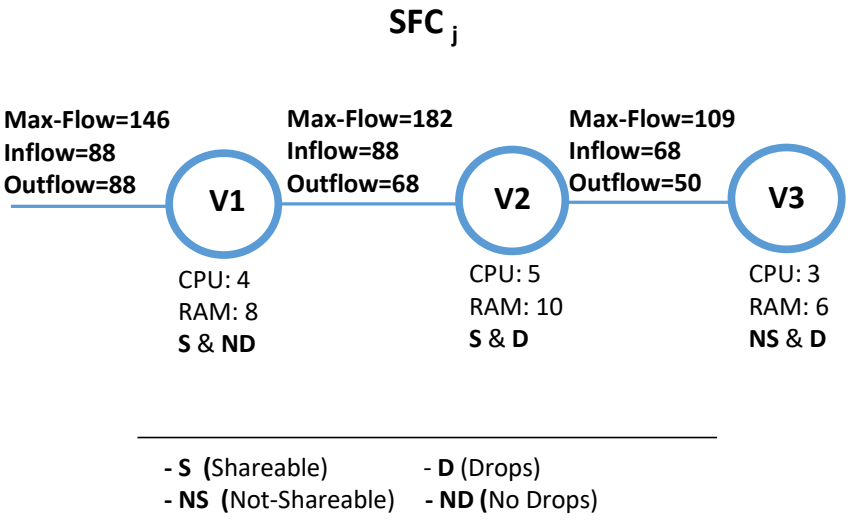


Figure 3.3: VNFs of sfc_i and their required resources, max-flows, inflows and outflows.

v_2, v_1 and v_3 require 5, 4, and 3 CPU cores and can handle 182, 146 and 109 max-flows, respectively. Once selected in sfc_j, the inflow and outflow of VNFs should be determined. As shown in Figure 3.3, the inflow of VNF v^j_{i+1}, $F_{in}(v^j_{i+1})$, is the outflow of the preceding VNF v^j_i, $F_{out}(v^j_i)$, for all $v_i \in sfc_j$. Description of VNF and SFC request parameters are in Table 3.2.

3.3.3 Problem Formulation

We formulate our problem as an ILP model, where all 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} if equals to one means a new instance of VNF v_i of sfc_j is to be placed at substrate node n and R^j_{in} means that VNF v_i of sfc_j is to share its traffic flow with already deployed underutilized VNF of the same type at substrate node n. Decision variables and other parameters descriptions are

Table 3.2: VNF and SFC request parameters.

Parameter	**Description**
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
$\|sfc_j\|$	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

in Table 3.3.

Objective Function

The objective is to select the placement that minimizes the overall cost, hence optimizing resource utilization. The cost of instantiating a new instance of VNF v_i^j includes the total required *cpu*, *ram* and *bw* costs. But when an already deployed VNF of the same type v_i^j exists, the cost only includes total required *bw* cost. The objective function in equation (3.1) is formulated in a way to favor sharing over instantiating and placing new VNF instances. The first term represents the cost of compute resources in case of deploying a new instance of a VNF. The second term

Table 3.3: Decision variables and Constants.

Variable	**Description**
X_{in}^{j}	Binary decision for placing VNF v_i of sfc_j at node n
R_{in}^{j}	Binary decision for sharing the flow of VNF v_i of sfc_j with already deployed VNF of same type at node n
D_n^i	VNF of same type as v_i already deployed at node n
$F_{av}(D_n^i)$	Available unused flow of v_i at node n
$U_c(cpu)$	Unit cost of cpu at all nodes
$U_c(ram)$	Unit cost of ram at all nodes
$U_c(bw)$	Unit cost of bw at all links

is for the cost of bandwidth, either in the case of sharing or the case of deploying a new VNF instance. Using different weights for objective function's cost components, may yield slightly different results. However, using equal weights does not affect the generality of the analysis.

$$min \sum_{i=1}^{|sfc_j|} \sum_{n \in N} [cpu(v_i^j)U_c(cpu) + ram(v_i^j)U_c(ram)]X_{in}^{j} + F_{out}(v_i^j)U_c bw[X_{in}^{j} + R_{in}^{j}] \tag{3.1}$$

Constraints

Constraints are needed to ensure that our model will converge to a feasible solution. A feasible solution must have each VNF of an SFC request mapped only once to a physical node. Moreover, the mapping should be either to place a new instance or to share an already deployed instance, see (3.2).

$$\sum_{n \in N} X_{in}^{j} \; + \; R_{in}^{j} \; = \; 1 \quad , \quad \forall i \in [1, |sfc_j|] \tag{3.2}$$

Constraints (3.3) and (3.4) ensure that, when a sharing decision is to be taken, there has to be an already deployed shareable VNF of the same type as the one in hand, and there is enough available/unused flow that is enough for the current VNF inflow.

$$\sum_{n \in N} X_{in}^{j} \; + \; D_{i}^{j} \; S(v_i) \; R_{in}^{j} \; = \; 1, \quad \forall i \in [1, |sfc_j|] \tag{3.3}$$

$$F_{in}(v_i^j) \; R_{in}^{j} \; \leq \; F_{av}(D_n^i), \quad \forall i \in [1, |sfc_j|] \tag{3.4}$$

For the placement decision X_{in}^{j} to be valid, there must be enough cpu and ram resources at node n, constraints (3.5) and (3.6) ensure the availability of such compute resources.

$$\sum_{i=1}^{|sfc_j|} cpu(v_i^j) \; X_{in}^{j} \; \leq \; cpu_{av}(n), \quad \forall n \in N \tag{3.5}$$

$$\sum_{i=1}^{|sfc_j|} ram(v_i^j) \; X_{in}^{j} \; \leq \; ram_{av}(n), \quad \forall n \in N \tag{3.6}$$

For any two consecutive VNFs v_i^j and v_{i+1}^j 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, constraint (3.7); second, the outflow of first VNF $F_{out}(v_i^j)$ should not exceed available bandwidth at that link $bw_{av}(L_{nn'})$, constraint (3.8). In both constraints, the two terms $(X_{in}^{j} + R_{in}^{j})$ and $(X_{(i+1)n'}^{j} + R_{(i+1)n'}^{j})$ represent the four possible placement decisions our model

might take. First decision, to deploy new instances of v_i^j and v_{i+1}^j at nodes n and n', respectively. Second, to share already deployed instances of v_i^j and v_{i+1}^j at nodes n and n'. Third, deploy a new instance of v_i^j at node n and share a deployed instance of v_{i+1}^j at n'. Fourth, share a deployed instance of v_i^j at node n and deploy a new instance of v_{i+1}^j at node n'.

$$L_{nn'}\,(X^j_{in}\,+\,R^j_{in})(X^j_{(i+1)n'}\,+\,R^j_{(i+1)n'})\,=\,1 \qquad \forall n, n' \in N\ \&\ \forall i \in [1, (|sfc_j|-1)] \tag{3.7}$$

$$\sum_{n\in N}\sum_{n'\in N} F_{out}(v_i^j)(X^j_{in}\,+\,R^j_{in})(X^j_{(i+1)n'}\,+\,R^j_{(i+1)n'}) \leq bw_{av}(L_{nn'}), \quad \forall i \in [1, (|sfc_j|-1)] \tag{3.8}$$

Finally, the performance requirements (end-to-end latency) of sfc_j must be satisfied. Even though end-to-end latency has many components, such as processing delay, queuing delay, propagation delay and virtualization delay [94], for simplicity, the only component we opted for is the fixed propagation delay, see constraint (3.9). The end-to-end delay of an SFC placement solution equals to the summation of propagation delay of individual links.

$$\sum_{i=1}^{|sfc_j|-1}\sum_{n\in N}\sum_{n'\in N} Del(L_{nn'})\,(X^j_{in}\,+\,R^j_{in})\,(X^j_{(i+1)n'}\,+\,R^j_{(i+1)n'}) \leq Del(sfc_j) \tag{3.9}$$

3.4 Performance Evaluation

To evaluate and demonstrate the performance gain of the VNF sharing-based SFC placement scheme, we developed a Java-based simulation environment. The simulation environment generates a substrate network model, creates SFC requests, executes the placement decisions, and tracks the network model's total utilization as well as other measurements. The ILP model is solved using the Gurobi solver [56]. All simulations executed on Dell OPTIPLEX 9020 machine of Intel Core i7@3.6 GHz, 16 GB RAM with Windows 10 Enterprise. We used the NSFNET network topology with 14 nodes and 21 bidirectional links.

Each time a substrate network model is created, the topology is fixed, but the resources cpu, ram, and bw are drawn randomly from the ranges $[8, 64]\, cores$, $[16, 128]\, GB$ and $[100, 1000]\, Mbps$, respectively. These ranges at the time were based on the available compute resources in AWS VM flavors. Moreover, the link length that determines propagation delay, is also random and drawn from the range $[50, 1000]\, m$.

SFC requests are also generated randomly. The SFC request length, resource requirements of each VNF, each VNF shareability and if it drops/compresses inflow, and VNFs inflow and outflow; all these parameters are random and drawn from predetermined ranges, as follows:

- SFC length range $[2, 10]\, vnfs$.
- SFC end-to-end latency $= (|sfc_j| - 1) *$ average-link-delay.

- VNF cpu range $[2, 8]\, cores$.
- VNF ram range $[4, 16]\, GB$.
- VNF $maxflow$ is a function of required/assigned cpu and ram.
- VNF $inflow$ range $[0.15 * maxflow, maxflow]\, Mbps$ (based on sensitivity analysis of the results.)
- If VNF drops/compresses, $outflow$ range $[0.4 * inflow, inflow]$. If not, $outflow = inflow$.

We chose to set $U_c(cpu) = 2.5$, $U_c(ram) = 1.7$, $U_c(bw) = 2$, these arbitrary values have no effect on the results as they are the same on both sides of any comparison.

3.4.1 Simulations

For the purpose of demonstrating the gain, more requests satisfied with less average required resources per request, we design two simulations. In the first, we compare the proposed VNF sharing-based placement versus no-sharing-based placement. In the second, we study the impact of increasing the number of shareable VNFs in the list of on-boarded VNFs, hence the number of deployed shareable VNFs.

Sharing vs No-Sharing

In this first simulation, 10 random network models are generated. For each network model, an identical copy is cloned, one is used to satisfy SFC requests using VNF sharing-based placement and the other with no-sharing-based placement. For each identical pair of network models, 30 SFC requests are generated of the same VNF list. Each SFC request is cloned, and the VNFs of the clone are set to non-shareable,

one used for sharing-based placement and the clone for no-sharing-based placement, respectively. Each network model pair trial is repeated five times, and the averages of the number of satisfied SFC requests and closing *cpu* utilization are recorded. Since the amount of *ram* resources is double the *cpu* resources both in substrate nodes and VNFs, we will only report and compare the *cpu* utilization. The *bw* utilization is ig-

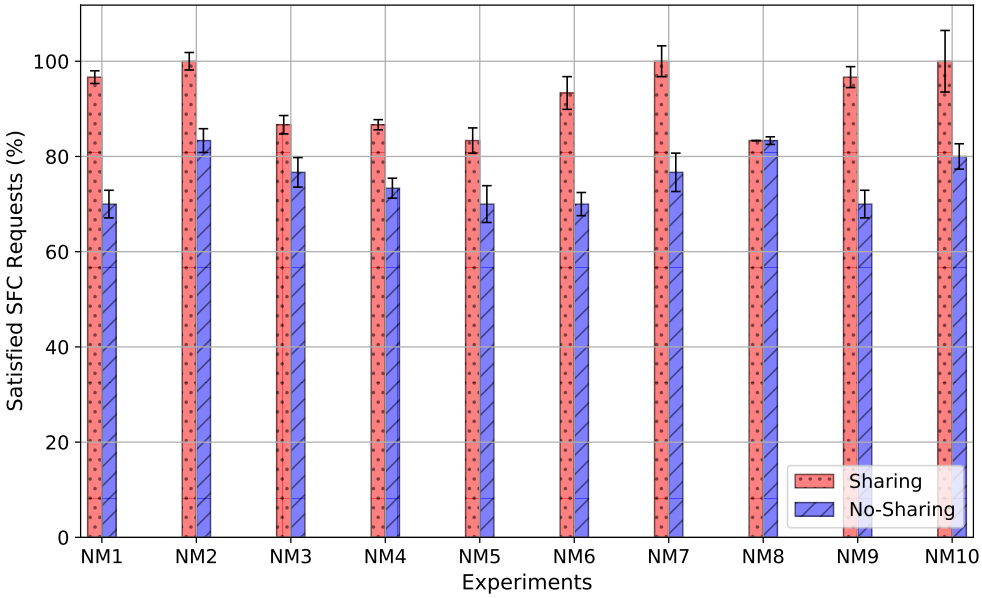


Figure 3.4: Satisfied SFC requests (percentage out of 30 requests) per network model. Simulation for each network model is repeated five times, and the averages are presented.

nored as it will be the same for sharing-based and no-sharing-based placement. As the results reveal, in Figure 3.4, sharing the allocated capacity of already deployed VNFs, resulted in a significant increase in the percentage of satisfied SFC requests ranging from 9% to 47%, yet, as expected, using less compute resources. That is, accepting

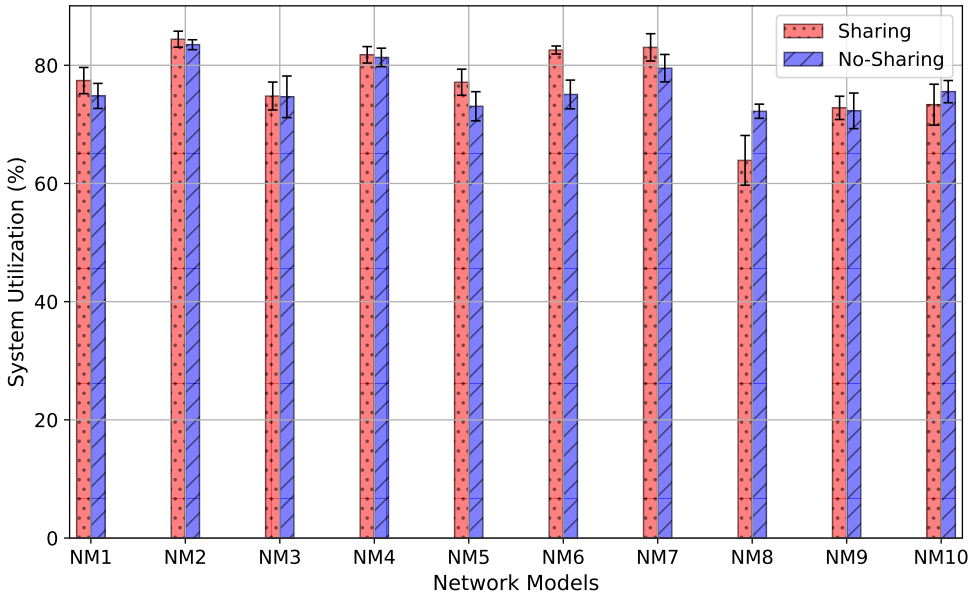


Figure 3.5: Average total utilization per network model. Simulation for each network model is repeated five times, and the averages are presented.

the same number of service requests, the required resources are lower for sharing-based placement compared to no-sharing-based placement. In network models NM8 and NM10, the utilization of sharing is less than that of no-sharing this is attributed to the possibly lower number shareable VNFs in the sharing-based placement and to the factor that link bandwidth and delay play in the feasibility of placement solution. The sharing-based utilizes deployed underutilized VNFs, however, it consumes link bandwidth which may result in making subsequent sharing solutions unfeasible.

System utilization does not tell the true story as in situations where the number of satisfied requests is the same, resource utilization in sharing-based is less than that of no-sharing (see utilization NM8 in Figure 3.5). As a result, the average utilized resources per satisfied SFC requests will tell the true story. To show the average

percentage of required/utilized resources per satisfied SFC request; in each network model trial, we divide the closing utilization by the number of satisfied SFC requests. As shown in Figure 3.6, on average, the required/utilized resources per request for sharing-based placement is 14% to 46% less compared to no-sharing-based placement.

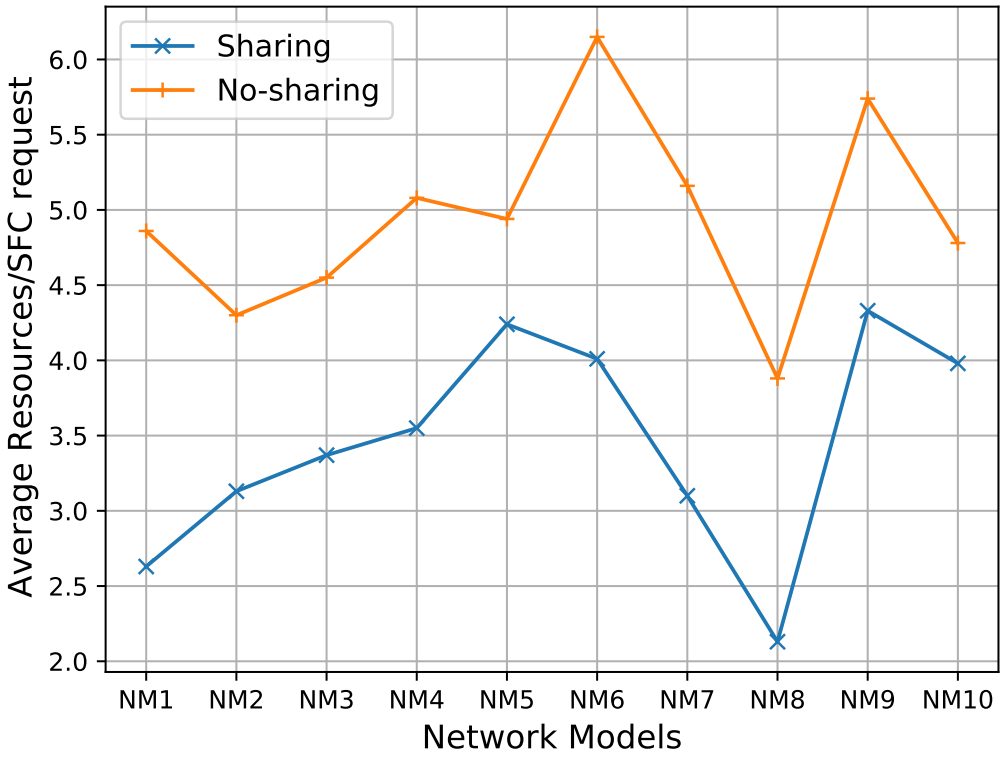


Figure 3.6: Average percentage of required resources per SFC request.

Number of Shareable VNFs (70%-based vs 30%-based)

In this second simulation, we study the impact of the number of shareable VNFs in the list of on-barded VNFs. To do so, we create one typical pair of network models,

and create a typical pair of on-boarded VNFs list. The first list is with 70% of its VNFs are shareable, and the other list is with 30% shareable VNFs. Using the two VNF lists, we generate two sets of SFC requests, each with 30 requests. Peer SFC requests in the two sets are of the same length, i.e., the same number of VNFs. For each network model pair, we repeat this simulation ten times and reported the averages. The total number of accepted requests and the resources utilized are shown in Figures 3.7 and 3.8. We can observe that the number of shareable VNFs impacts

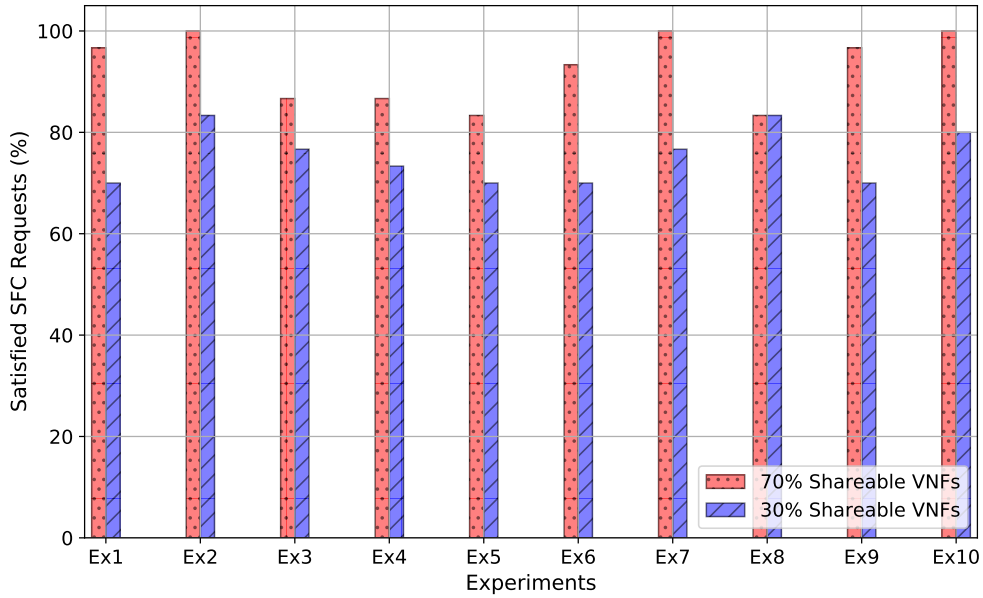


Figure 3.7: Comparison of satisfied SFC requests (percentage out of 30 requests) between 70%-based shareable VNFs and 30%-based shareable VNFs.

number of satisfied requests and the required resources. Over the ten simulations, the number of accepted SFC requests is 13% to 26% higher for the 70%-based requests than for the 30%-based requests. On the other hand, the utilized resources of the 70%-based accepted requests range from 11% lower to only 5% higher than the 30%-based

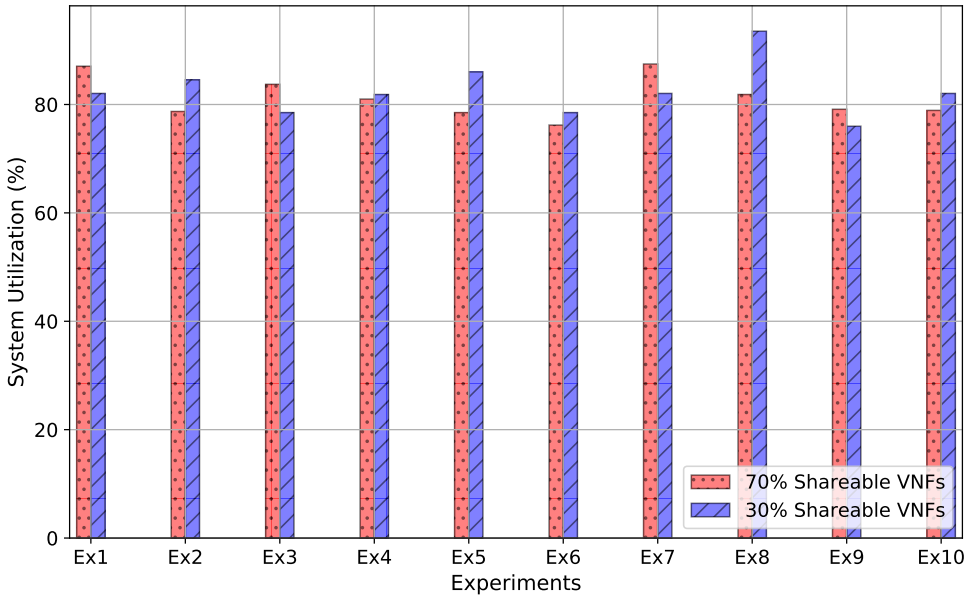


Figure 3.8: Comparison of closing utilization between 70%-based shareable VNFs and 30%-based shareable VNFs.

requests. When the same number of requests are accepted, the amount of resources utilized is 11% less for the 70%-based requests than for the 30%-based requests.

3.5 Summary

To take advantage of the changing operating conditions and the fluctuation in traffic flow over different periods, we demonstrate the performance gain of *VNF sharing*-based SFC requests placement scheme. The demonstrated gain is in the form of an increased number of satisfied SFC requests and a reduction of resources required to satisfy these requests. Indeed, the *VNF sharing*-based placement scheme has to be done with the current network load and individually deployed VNFs' available flow in mind. Unlike previous related work, our *VNF sharing*-based placement scheme

takes the decisions by considering the current utilization status of deployed VNFs, not based on a predetermined number of SFC flow a VNF can handle. The latter may still leave some VNFs underutilized. Our findings will help CSPs efficiently utilize their edge resources. This gain will translate into more earnings and better user satisfaction due to more satisfied/fewer blocked requests.

As a next step, we plan to consider different service categories. For example, a premium category service should not wait for resource availability. Moreover, we will make our scheme more adaptive to operations dynamics which affects the shareable capacity of shareable VNFs. Finally, a migration scheme is needed to handle situations where a host VNF cannot stand a traffic increase of its own or of one of the guest VNFs.

Chapter 4

Priority-based Service Placement

"Edge Is Key to Monetizing 5G, and Telcos Have Beach-Front Property"

Bikash Koley, Juniper Nets CTO

4.1 Introduction

As mentioned in Chapter 3, edge has limited resources compared to the core cloud. Services and applications are not created equal in terms of delay sensitivity and deployment urgency once the service request is received. With the anticipated high demand for edge resources and the importance of the edge being a precious asset, Communications Service Providers (CSPs) should offer service categories to satisfy varying delay requirements and urgency, which is reflected in the cost.

To date, only a few research works with limited scope exist on *Virtual Network Function (VNF) sharing*. Previous studies either consider a predefined number of

traffic flows a VNF can handle, ignoring the operation dynamics or proposing prioritization mechanisms that dynamically assign priorities based on the current system state. Building on findings of Chapter 3, we propose PSVShare, a priority-based SFC placement scheme with *VNF sharing*. PSVShare takes Service Function Chain (SFC) placement decisions at the point-of-presence (PoP) level, and supposed to be part of the Network Function Virtualization Orchestrator (NFVO).

The main contributions of this chapter are the introduction of PSVShare, a priority-based *VNF sharing*-based SFC placement scheme, 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 guest VNFs. Furthermore, we design HPA, a heuristic placement algorithm to overcome the complexity of NP-hard Integer Linear Program (ILP) placement algorithm used in PSVShare.

The remainder of this chapter is structured as follows. Section 4.2 covers related work. Proposed sharing-based SFC placement, system model, and problem formulation are detailed in Section 4.3. Performance evaluation and results analysis are covered in Section 4.4. The HPA and its performance evaluation are explained in Sections 4.5 and 4.6, respectively. Lastly, conclusions and future works are presented in Section 4.7.

4.2 Related Work

4.2.1 Priority-based Placement

In the literature on priority-based SFC and VNF placement, the concept and handling of priority are diverse. The work in [81] utilizes priority that is dynamically

assigned to SFCs, VNFs or flows, depending on the situation, rather than a predetermined priority before deployment. In [105], 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 independent of the required resources.

4.3 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 [91], that services come in two types/classes: Pr service that is provisioned with the highest expected load, not oversubscribed, and served with dedicated high-priority queues; and 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 scheme 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 the varying traffic load that SFCs and their VNFs serve. The capacity of a host VNF is distributed among its guest VNFs. Any increase in 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 also usable for core/cloud-hosted services.

4.3.1 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 the substrate network and SFC parameters is provided in Table 4.1. The placement algorithm and system model have some similarities with those in Chapter 3, for completeness we are listing some repeated details.

4.3.2 Problem Formulation

The PSVShare scheme utilizes an integer quadratically-constrained program (IQCP) model. The objective is to minimize the total deployment cost by optimizing resource

Table 4.1: 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
$ID(v_i^k)$	Unique identifier of deployed instance v_i of sfc_k
$Del(sfc_j)$	Maximum end-to-end delay of sfc_j

utilization while satisfying QoS requirements/constraints. PSVShare prioritizes Pr SFCs over BE SFCs and handles migration situations arising from *VNF sharing* and traffic increase. PSVShare, listed in Algorithm (4.1), 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, the 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 $Run_{pr|be}$ queue to the $Comp_{pr|be}$ list. If traffic flow increase cannot be accommodated either because the host VNF's own flow or due to an increase of any of its guest VNF flows, one or more migrations are unavoidable. If an SFC migration is successful, SFC ends in the *running* state but passes by a *terminated/suspended* 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 (4.2), to return a list of SFCs to be deported to accommodate the traffic increase. If a host VNF of the SFC under

Algorithm 4.1: 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
```

investigation cannot handle the traffic increase, first *migReqrd* 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 completing 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.

IQCP

To satisfy an SFC, new or deported, the $satisfy$ function in Algorithm (4.1): lines[9,30, and 37] are 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} 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 3.3.

Objective Function The objective is to select the placement that minimizes the overall cost, hence optimizing resource utilization, and minimizing the number of migrations of Pr SFCs. Therefore, the objective function in equation (4.1) is formulated to prefer sharing over instantiating new VNF instances.

Algorithm 4.2: migReqrd function

```
// nIF: newInFlow & nOF: newOutFlow
Input  : rs_pr|be, nIF
Output: migList: if migration required, null: otherwise
1  Function migReqrd(rs_pr|be, nIF):
2      foreach VNF v_i in rs_pr|be do
3          if isHost(v_i) = true then
4              if (nIF-F_in(v_i)) > F_av(v_i) then
5                  add(v_i,diagList)
           // v_i is a guest VNF
6          else if (nIF-F_in(v_i)) > F_av(hostVNF(v_i)) then
7              add(v_i,diagList)
8          if (nOF-F_out(v_i)) > bw_av(outLink(v_i)) then
9              add(v_i,diagList)
10     if diagList isn't empty then
11         foreach VNF v_i in diagList do
12             if isHost(v_i) = true then
                   // Add BE SFCs sorted with ↑ inFlow and ↓ TTL
13                 add(SFCs_be,migList)
14                 if Added SFCs' inFlow isn't enough then
                       // Add Pr SFCs sorted with ↑ inFlow and ↓ TTL
15                     add(SFCs_pr,migList)
16             else add(rs_pr|be,migList)
17
18
19         return migList
       // No migration(s) required
20     else return null
21
```

$$min \sum_{i=1}^{|sfc_j|} \sum_{n \in N} [cpu(v_i^j) U_{cpu}^c(n) + ram(v_i^j) U_{ram}^c(n)] X_{in}^j + F_{out}(v_i^j) U_c bw [X_{in}^j + R_{in}^j] \tag{4.1}$$

Constraints First, a feasible solution has to have each VNF of SFC request mapped only once to a physical node. Moreover, the mapping should either place a new or to share a deployed VNF instance, equation (4.2). 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 (4.3), and there is unused capacity enough for new VNF inflow, equation (4.4). Third, for the placement decision X_{in}^j to be valid, there must be enough cpu and ram resources at node n, equations (4.5 & 4.6). Fourth, for any two consecutive VNFs v_i^j and v_{i+1}^j 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 (4.7); second, the outflow of first VNF $F_{out}(v_i^j)$ should not exceed available bandwidth at that link $bw_{av}(L_{nn'})$, equation (4.8). To minimize the 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 (4.9). Finally, the performance requirements (end-to-end latency) of sfc_j must be satisfied, equation (4.10).

$$\sum_{n \in N} X_{in}^j + R_{in}^j = 1 \quad , \quad \forall i \in [1, |sfc_j|] \tag{4.2}$$

$$\sum_{n \in N} X_{in}^j + D_i^j \, S(v_i) \, R_{in}^j = 1, \quad \forall i \in [1, |sfc_j|] \tag{4.3}$$

$$F_{in}(v_i^j)\ R_{in}^j\ \leq\ F_{av}(D_n^i),\ \forall n \in N\ \&\ \forall i \in [1, |sfc_j|] \tag{4.4}$$

$$\sum_{i=1}^{|sfc_j|} cpu(v_i^j)\ X_{in}^j\ \leq\ cpu_{av}(n), \quad \forall n \in N \tag{4.5}$$

$$\sum_{i=1}^{|sfc_j|} ram(v_i^j)\ X_{in}^j\ \leq\ ram_{av}(n), \quad \forall n \in N \tag{4.6}$$

$$\begin{aligned} L_{nn'}\ (X_{in}^j\ +\ R_{in}^j)(X_{(i+1)n'}^j\ +\ R_{(i+1)n'}^j) = 1 \\ \forall n, n' \in N\ \&\ \forall i \in [1, (|sfc_j| - 1)] \end{aligned} \tag{4.7}$$

$$\begin{aligned} \sum_{n \in N} \sum_{n' \in N} F_{out}(v_i^j)(X_{in}^j\ +\ R_{in}^j)(X_{(i+1)n'}^j\ +\ R_{(i+1)n'}^j) \\ \leq\ bw_{av}(L_{nn'}), \quad \forall i \in [1, (|sfc_j| - 1)] \end{aligned} \tag{4.8}$$

$$cat(v_i^j)\ R_{in}^j\ \leq\ cat(D_n^i),\ \forall n \in N\ \&\ \forall i \in [1, |sfc_j|] \tag{4.9}$$

$$\begin{aligned} \sum_{i=1}^{|sfc_j|-1} \sum_{n \in N} \sum_{n' \in N} Del(L_{nn'})\ (X_{in}^j\ +\ R_{in}^j) \\ (X_{(i+1)n'}^j\ +\ R_{(i+1)n'}^j)\ \leq\ Del(sfc_j) \end{aligned} \tag{4.10}$$

4.4 Performance Evaluation

To evaluate and demonstrate the performance of PSVShare, we develop a Java-based simulation environment. The simulation environment generates a 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 [56]. All simulations 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 a network model with 28 nodes and 42 directional links.

All simulations utilized the same network model, with the same topology, nodes resources, and links bandwidth. Unless we are experimenting with the arrival rate λ, simulations 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., the number of SFC requests per TS and the length of each SFC. Unless otherwise stated, SFCs generated are 50% Pr and 50% BE with shuffled order of arrival. The only difference between these simulations 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 simulations, 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.

4.4.1 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 4.1 shows the different queue sizes as a percentage of received SFCs at the end of the simulation. Having "100%" shareable VNFs resulted in satisfying, completed and running, 26% more SFCs than 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 the network and hence satisfies more SFCs.

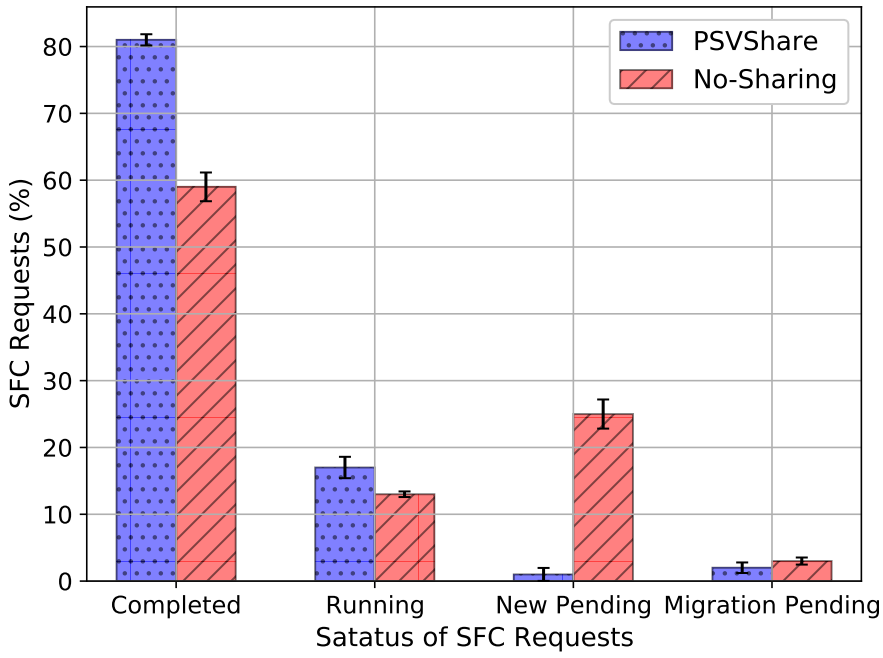


Figure 4.1: PSVShare with 100% shareable VNFs *vs* 0% shareable VNFs. $\lambda = 2$ and shuffled $50:50\%$ Pr-to-BE ratio.

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 capacity). Results in Figures 4.2 and 4.3, 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 schemes. 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 attributed to *VNF sharing* before deploying new VNF instances when satisfying SFC requests.

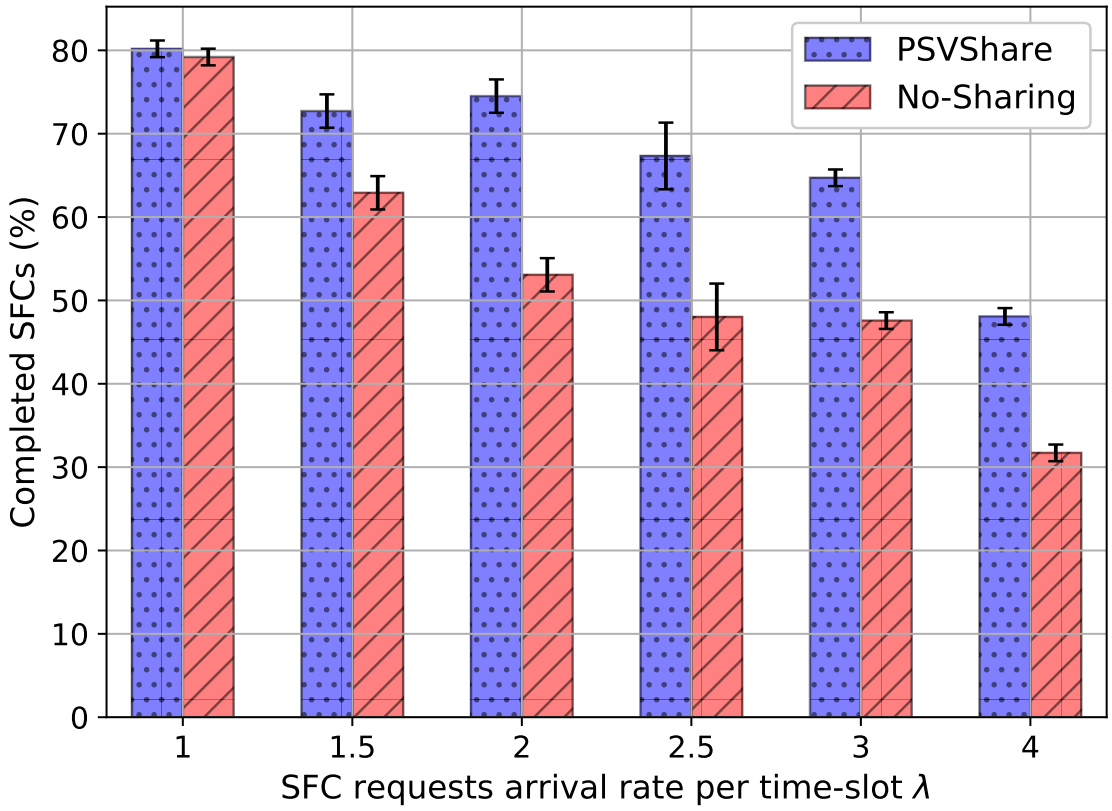


Figure 4.2: Completed SFC at the end of simulation, under different loads for PSVShare scheme vs the No-sharing scheme.

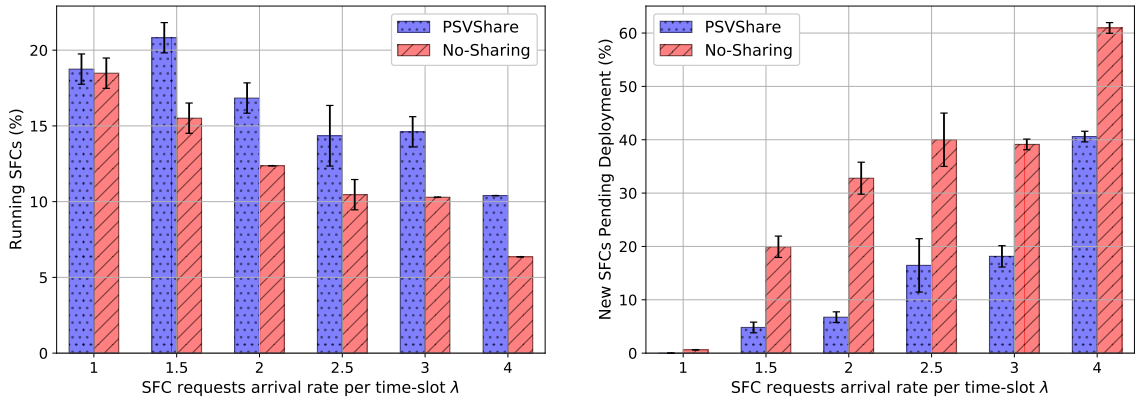


Figure 4.3: Running and new pending SFCs under different loads. All with 60% shareable VNFs, and shuffled $50-50\%$ Pr-to-BE ratio.

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 simulations, the queue sizes were not bounded. We limit the New_{Pr} & New_{be} queues to a finite size equal to a percentage of the total number of received SFC requests for this study. We experiment with queue sizes equal to 0, 2, 4, and 6% of received SFC requests. In Figure 4.4, results show that PSVShare maintained a stable superior performance against the No-Sharing scheme with a 95% confidence interval considered. The rejection rate of PSVShare range from 45% to more than 50% 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 experiment with different Pr-to-BE ratios.

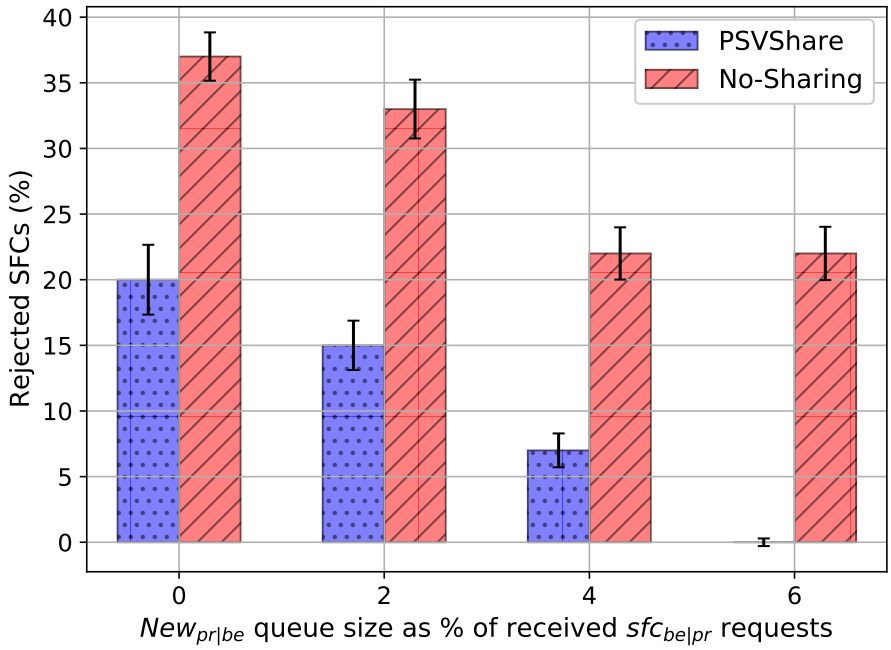


Figure 4.4: Rejected SFC requests (%) for different queue sizes.

Figure 4.5 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 to the '20-80' and '80-20' cases. For the '40-60', '50-50', and '60-40' cases, Pr SFCs have a 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 a large share of network model resources satisfying most Pr SFCs 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, it is still able to satisfy more BE SFCs. This is due to the efficient resource utilization, both when satisfying Pr and then BE SFCs.

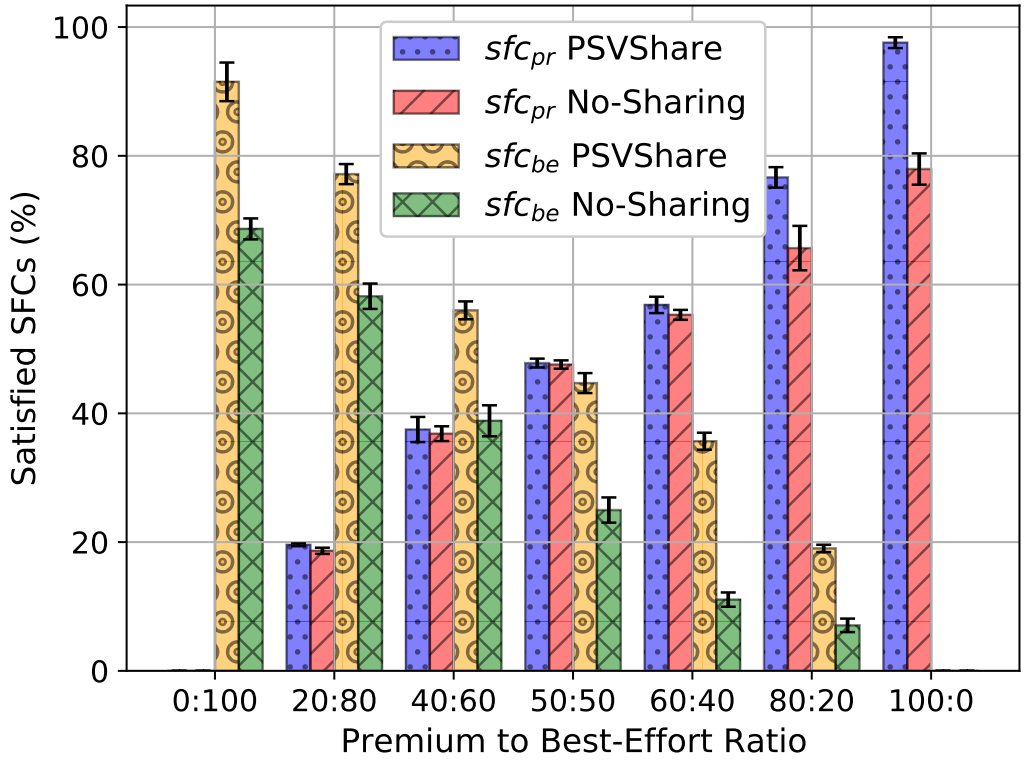


Figure 4.5: Satisfied SFC requests (%) of different Pr-to-BE ratios, shuffled.

4.5 Heuristic Placement Algorithm

The placement algorithm used in PSVShare is modeled as an ILP. It is proven that the VNF/SFC placement is an Np-hard problem [22], hence a heuristic algorithm is needed to overcome the complexity of solving the placement problem at scale. As such, we design a heuristic placement algorithm (HPA), in Algorithm 4.3, to find a feasible placement of SFC request sfc_j. The inputs to HPA are the network model current state (deployed VNFs, VNFs utilization, and available resources in nodes and links), the SFC request at hand sfc_j, and the best solution criterion, i.e., least

Algorithm 4.3: HPA: Heuristic Placement

```
// Unit cost of cpu, ram is node dependent, BW unit cost is the same for all links.
Input  : Net Model, sfc_j, U_c(bw), bestSolCriterion
Output: solPath
1  foreach VNF v_i^j in sfc_j do
2      foreach Node n in N do
           // If a shareable VNF of the same type as v_i^j is deployed at node n, with F_av(D_n^i) ≥ F_in(v_i^j)
3          if shrbleVNFExists(v_i^j , n) then
4              Sol [v_i^j][n] = −1 ∗ ID(v_i^k); // v_i^k is a VNF of sfc_k
5          else if cpu(v_i^j) ≤ cpu_av(n) then
6              Sol [v_i^j][n] = 1
7          else
8              Sol [v_i^j][n] = 0
   // Maximum number of iterations = 7, the maximum |sfc_j|
9  for i ← 0 to |sfc_j| − 1 do
10     foreach Node n in N do
11         foreach Node n' in N do
12             if Sol [i][n]≠0 & Sol [i][n']≠0 then
13                 if isLinked(n , n') then
14                     Links [i].addLink(l_nn')
15 Paths ← getPaths(Links, sfc_j)
16 foreach Path p in Paths do
17     if isFeasible(p,Del(sfc_j)) then
18         feasiblePaths.addPath(p)
19 solPath ← bestSol(feasiblePaths, bestSolCriterion)
```

delay/cost. The HPA starts with an initial solution in a matrix form as shown in Figure 4.6, with rows representing SFC's VNFs and columns representing hosting nodes. For each VNF $v_i \in sfc_j$, HPA checks if a node $n \in N$ has a deployed shareable VNF of the same type as v_i and if that VNF has free capacity that is enough for the inflow of v_i ($f_{in}(v_i)$), the solution cell at $[v_i, n]$ is set to $(-1 * vnf_{id})$. If there is no such deployed shareable VNF or the VNF exists but cannot accommodate

the $f_{in}(v_i)$, HPA checks if the node has enough resources to host v_i. If resources are enough, solution cell at $[v_i, n]$ is set to 1, otherwise it is set to 0. The complexity of this step is $\mathcal{O}(c.n), c \leq |sfc_j|$, which leads to $\mathcal{O}(n)$.

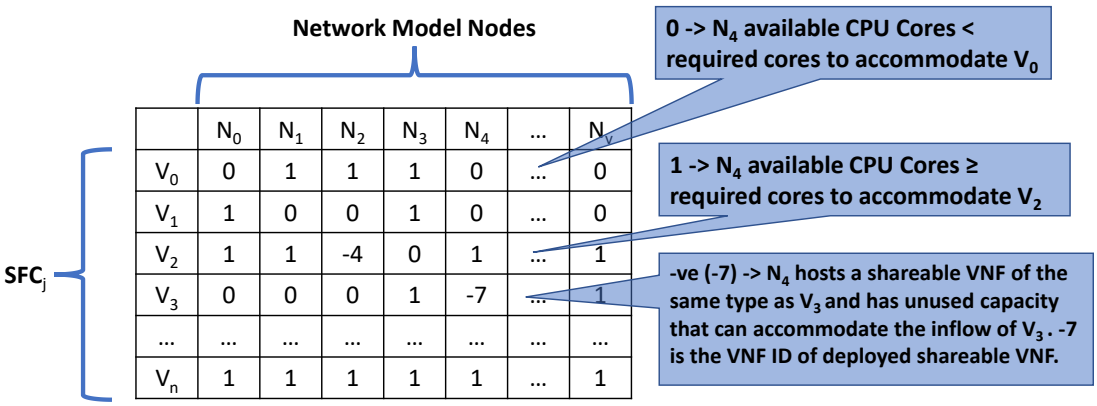


Figure 4.6: Heuristic Algorithm, step 1: initial solution.

In the next step, HPA uses the the initial solution to form possible solutions in the form of possible links. It does so by building another matrix of hops (a link/path that connects a node hosting v_i and v_{i+1}) as rows and nodes as columns, shown in Figure 4.7. For each node n in a row (representing a hop), if a link exists between node n and node n', a link $e_{nn'}$ is added to that row/hop. The complexity of this step is $\mathcal{O}(n^2)$.

Finally, using the matrix of links in the previous step, HPA creates complete paths using all possible combinations of links at each row/hop, shown in Figure 4.8. Then, all formed complete paths are tested for completeness and feasibility according to the given performance/QoS requirements of sfc_j. The best solution is the path that satisfies the defined best solution criterion. The overall complexity of HPA is $\mathcal{O}(n^2)$.

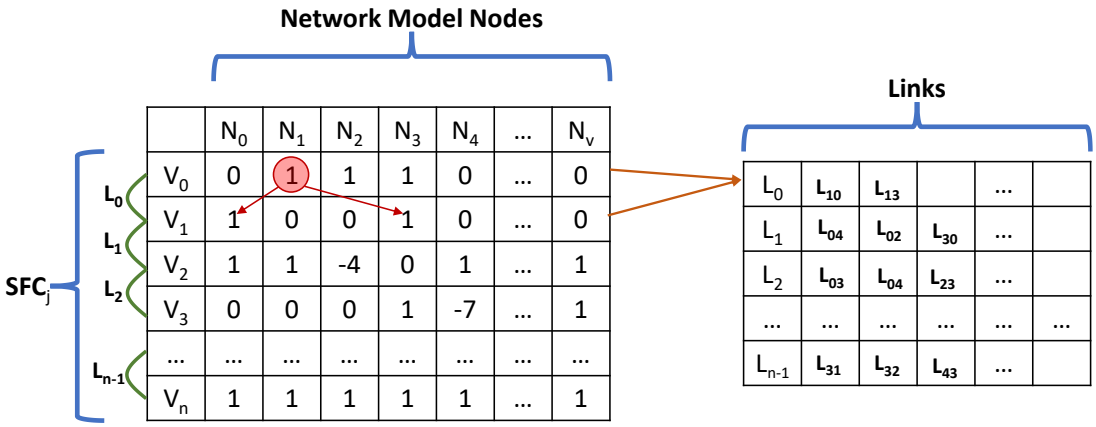


Figure 4.7: Heuristic Algorithm, step 2: possible links.

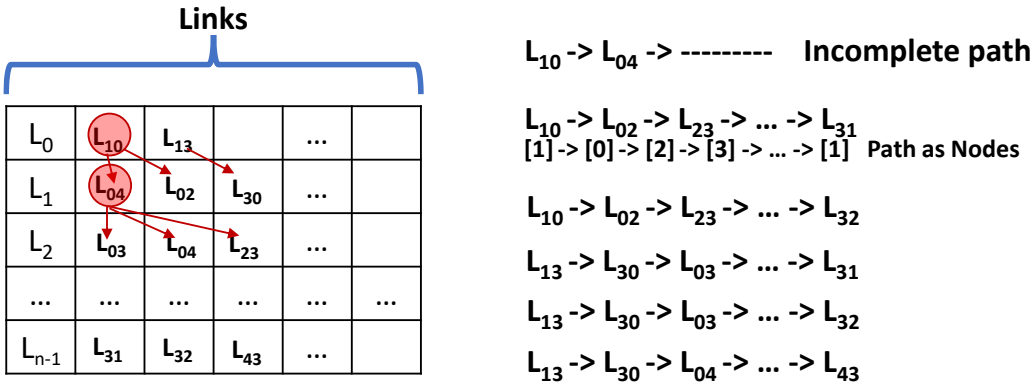
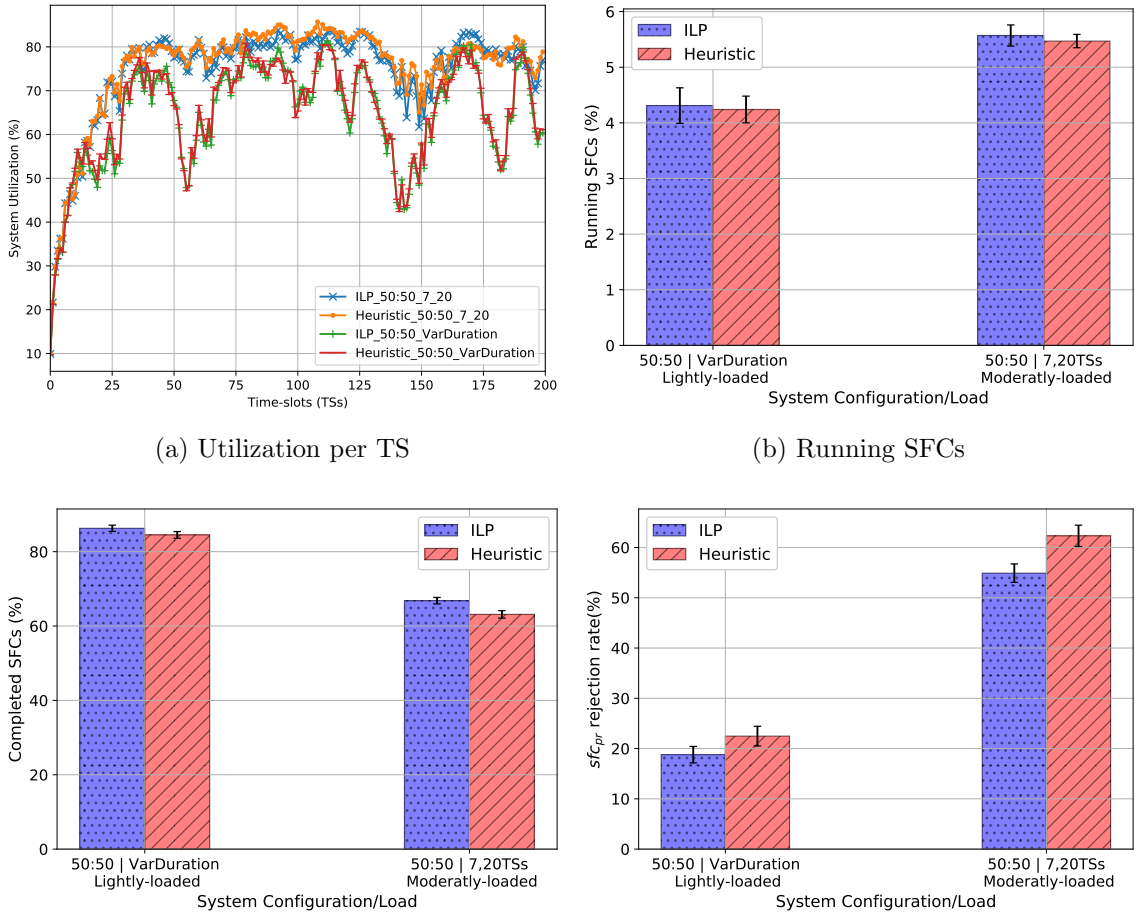


Figure 4.8: Heuristic Algorithm, step 3: paths formation.

4.6 Heuristic Algorithm Results

To evaluate the performance of HPA-based placement scheme vs the ILP-based placement scheme, we use the same simulation setup as in Section 4.4 for lightly and moderately-loaded systems. The results in Figure 4.9, shows an overall similar performance of HPA compared to ILP-based placement schemes. The system utilization of HPA, in Figure 4.9a, is almost the same as that of ILP for both lightly and

moderately-loaded systems. The percentage of running SFCs of ILP is slightly higher than the HPA-based scheme (ILP is 1.6% and 1.8% higher than HPA for lightly and moderately-loaded systems, respectively), shown in Figures4.9b. Similarly, as shown in Figure 4.9c, the percentage of completed SFCs of ILP is $2-5.8\%$ higher than HPA. The percentage of rejected sfc_{pr} requests of ILP is 11% less than the HPA for the moderately-loaded system.



(a) Utilization per TS

(b) Running SFCs

(c) Completed

(d) sfc_{pr} rejections

Figure 4.9: Utilization during simulation time and closing queue sizes.

The performance hit that HPA received in the form of 11% more rejections of

sfc_{pr} requests compared to ILP-based schemes, is balanced by the $28x$ fold reduction in AWT of sfc_{pr}, shown in Figure 4.10.

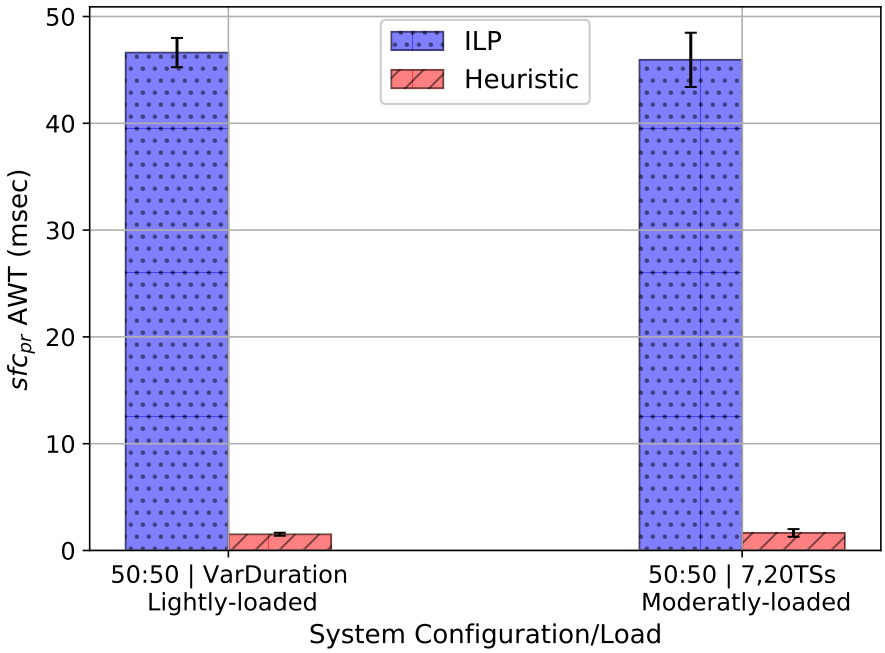


Figure 4.10: Average waiting time (AWT) of sfc_{pr} requests.

4.7 Summary

To take advantage of the changing operations dynamics and common VNFs among services, we design PSVShare, a priority-based SFC placement scheme with *VNF sharing*. PSVShare handles migration situations arising from sharing VNFs and traffic variation. Simulation results show a consistent out-performance of PSVShare over the Non-sharing scheme under different loads, with varying Pr-to-BE ratios, and with different queue sizes. The heuristic placement algorithm (HPA)-based placement

scheme performance is comparable to the ILP-based placement schemes. The HPA-based schemes excel at the execution time compared to the ILP-based, hence the AWT of Pr SFCs is significantly reduced.

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

Chapter 5

Prediction-based Service Placement

"We believe in clouds, not cloud"

Bryn Worgan, JPMorgan Chase

5.1 Introduction

The majority of emerging 5G use cases are time-sensitive in nature, such as real-time media (augmented reality (AR) and virtual reality (VR)), industrial control, remote control, and mobility automation [43]. Time-sensitive, 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 [54,61,78]. The unbounded delay and jitter results in motion sickness and other discomforts to VR/AR users [36, 109]. Edge computing, especially multi-access edge computing (MEC), is the distributed cloud to fulfill the ultra-low latency and high-reliability requirements of use cases mentioned earlier.

The Quality of Experience (QoE) that consumers of Pr SFCs perceive degrades if services have to wait for resource availability. If a Pr SFC (sfc_{pr}) request is received

and no resources are available, the request will be rejected. Conversely, lower-priority best-effort (BE) SFC (sfc_{be}) requests can tolerate waiting for resource availability. Prioritizing sfc_{pr} requests over sfc_{be} requests will help in slightly reducing the rejection rate of sfc_{pr} requests. However, such prioritization will not significantly reduce or even eliminate sfc_{pr} requests rejection.

Hence, we propose a prediction-based SFC placement scheme with VNF sharing (PSVS) to reduce the rejection rate of sfc_{pr} requests. Taking advantage of shareable common VNFs among SFCs and the predictability of sfc_{pr} requests arrival, PSVS will decide to satisfy sfc_{be} requests pending deployment or to defer the deployment to save resources for future sfc_{pr} requests. PSVS uses a *safety margin* to mitigate the unavoidable prediction errors. The main contributions are:

- Introduction of a prediction-based placement scheme that significantly reduces the rejection rate of sfc_{pr} requests.
- Introducing the *safety margin* that provides the desirable resiliency against prediction errors including extreme errors with a high error rate/probability.

The remainder of this chapter is structured as follows. Section 5.2 covers related work. Proposed prediction-based SFC placement, system model, and problem formulation are described in Section 5.3. Section 5.4 details the simulation framework, performance evaluation, and results analysis. Conclusions and future work are presented in Section 5.5.

5.2 Related Work

To better serve customers, service providers utilize prediction in different aspects. The service provider can use models/tools to predict service arrivals which can help

reserve resources for Pr SFCs to avoid rejection/failures [98], traffic demand and user mobility that both help with proactive replication and/or migration [71,97], the resources to allocate to guarantee performance requirements (profiling) [66], finally, to predict the degradation and/or failure of physical nodes hosting SFC's VNFs that help mitigate service interruptions and downtimes.

We utilize service request arrival predictions to reduce the rejection rate of premium services. In addition, we utilize a *VNF sharing*-based SFC placement that considers operations dynamics when taking VNF sharing decisions and prioritizes premium over best-effort services. PSVS, the proposed scheme, considers one priority for all SFC components, and that priority is known prior to receiving SFC requests.

5.3 System Model and Problem Formulation

Similar to the PSVShare scheme in Chapter 4, we utilize two service categories premium (Pr) and best-effort (BE) services. PSVS is nevertheless applicable to service domains with more than two service categories.

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 operation dynamics and shareable VNFs to enhance resource utilization and reduce SFC deployment cost, and rejection rate. To satisfy a new SFC request, the placement scheme searches running VNFs for similar underutilized VNFs which can serve the required load of peer VNF of SFC requests being satisfied. The scheme will only instantiate a new VNF when: considered VNF is non-shareable; there are no similar previously deployed VNFs or a similar VNF(s) deployed but fully utilized.

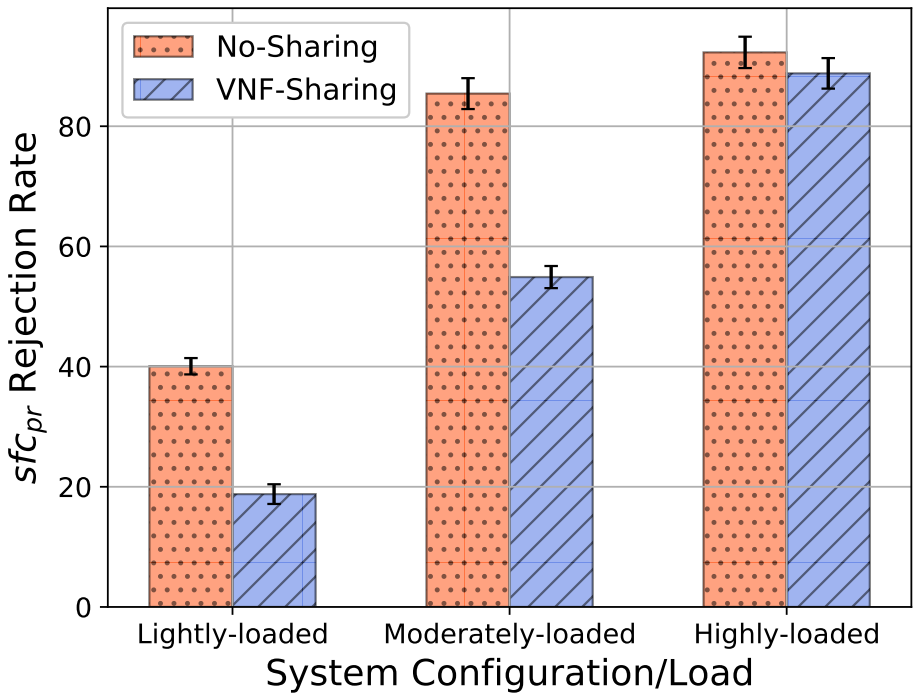


Figure 5.1: Rejection rate of Premium (Pr) SFCs using VNF non-sharing vs *VNF-sharing* for different system configurations/loads (Simulation duration is 200 TSs, arrival rate $\lambda = 2$ sfc requests/TS).

According to the results achieved from our PSVShare extensive simulations [88], SFC placement with *VNF sharing* reduced the rejection rate of sfc_{pr} requests, but it is still concerning. As shown in Figure 5.1, the best-case scenario is *'lightly-loaded.'* Using *VNF sharing*-based placement results in a 50% reduction in sfc_{pr} requests rejection rate. However, the rejection rate of the *VNF sharing*-based placement is still about 19%. It is worse for higher system loads, with longer duration and/or higher number of sfc_{be} requests. These rejections lead to unsatisfied customers and lost revenue for CSPs.

To address such concerning rejections, we propose a prediction-based SFC placement with VNF sharing (PSVS) to help CSPs better the edge resource utilization and minimize lost revenue opportunities. To minimize the situations where resources are not available to satisfy sfc_{pr} requests, PSVS will utilize predicted sfc_{pr} requests to arrive in a lookahead window of length ω to decide to satisfy the pending sfc_{be} requests or not. To the best of our knowledge, prediction-based SFC placement has never been used in the context of SFC placement with *VNF sharing* at the resource-limited edge environments.

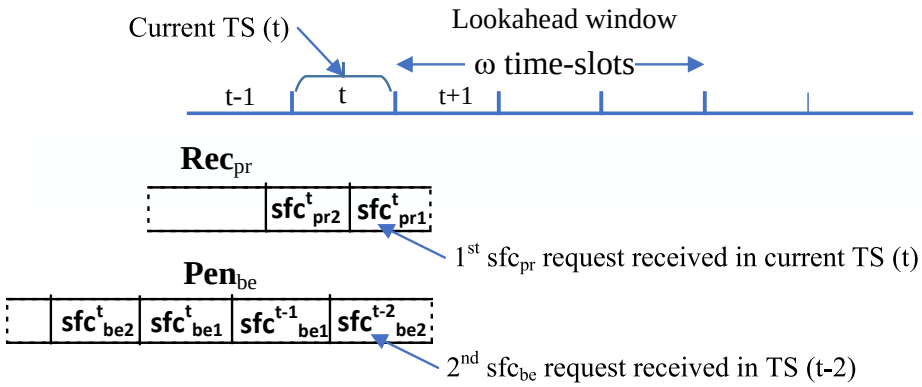


Figure 5.2: PSVS lookahead window, different queues (Rec_{pr} and Pen_{be}), and queued sfc_{pr}/sfc_{be} requests.

As shown in Figure 5.2, in each time slot (TS), arriving sfc_{pr} and sfc_{be} requests are placed in Rec_{pr} and Pen_{be} queues, respectively. While sfc_{be} requests can tolerate waiting for deployment if not satisfied in the same TS they were received, the sfc_{pr} requests cannot tolerate waiting and are rejected if not satisfied (in Figure 5.2, unlike

sfc_{be} requests in Pen_{be}, there are no sfc_{pr} requests older than current TS in Rec_{pr}). We assume that a simple prediction technique/model exists that can predict the sfc_{pr} requests to arrive in the next ω TSs and their required resources with 100% accuracy (in Section 5.4, we demonstrated the performance under different prediction errors and error rates). It is known that the longer the lookahead window, the less accurate the predictions, which is why we experimented with $\omega \in [1, 3]$ TSs.

5.3.1 System Model

Each SFC request consists of a list of VNFs. SFC's VNFs are selected from a list of on-boarded VNFs V. Each VNF type $v \in V$ has resource requirements like *CPU cores* and *memory*. A VNF is expected to operate at maximum capacity/throughput, $F_{max}(v)$, if assigned the required resources. As outlined earlier, not all VNFs are shareable, $S(v)$ is a flag to determine if a VNF v is shareable. The time a VNF becomes part of sfc_j, that VNF's inflow $F_{in}(v_i^j)$ and outflow $F_{out}(v_i^j)$ are determined. Due to operations dynamics, both inflow and outflow are subject to change, and for VNFs that drop or compress, like firewalls, $F_{out}(v_i^j) \leq F_{in}(v_i^j)$. The substrate network of nodes that hosts VNFs 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. Table 5.1 lists a detailed description of the substrate network and SFC parameters.

Table 5.1: System Parameters Description.

Parameter	Description
$cpu_{c\|av}(n)$	CPU capacity | available in cores of node $n \in N$
$ram_{c\|av}(n)$	RAM capacity | available in GBs of node $n \in N$
$e_{nn'}$	A link exists from node n to node n', $n, n' \in N$
$bw_{c\|av}(e_{nn'})$	BW capacity | available in Mbps of link $e_{nn'}$
$Del(e_{nn'})$	Propagation delay of link $e_{nn'}$
$cpu(v_i)$	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
sfc_j	SFC request j
$\|sfc_j\|$	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

5.3.2 Problem Formulation

Algorithm 5.1 describes the proposed PSVS scheme, a flowchart of the scheme is shown in Figure 5.3. The scheme hinges on the placement algorithm and the required resources prediction component. The placement algorithm is modeled as an integer program with binary decision variables and utilizes *VNF sharing*. PSVS utilizes a discretized time scale of TSs. First, at the beginning of each TS, PSVS terminates and releases utilized resources of running $sfc_{pr|be} \in Run_{pr|be}$ whose time-to-live (TTL) is zero and moves them to $Com_{pr|be}$ list. If SFC's TTL is not zero, PSVS decreases the TTL by one. Second, to prioritize sfc_{pr}, PSVS attempts to satisfy received requests

in the Rec_{pr} queue. Finally, if there are requests in the Pen_{be} queue, the predicted required resources in the next ω TSs are utilized to either satisfy some or all of the requests in Pen_{be} or defer deployment to leave room for sfc_{pr} requests to arrive. We utilize no Pen_{pr} queue because sfc_{pr} requests get rejected if not immediately satisfied. Similarly, no Rej_{be} list is used, since sfc_{be} requests can tolerate waiting for deployment.

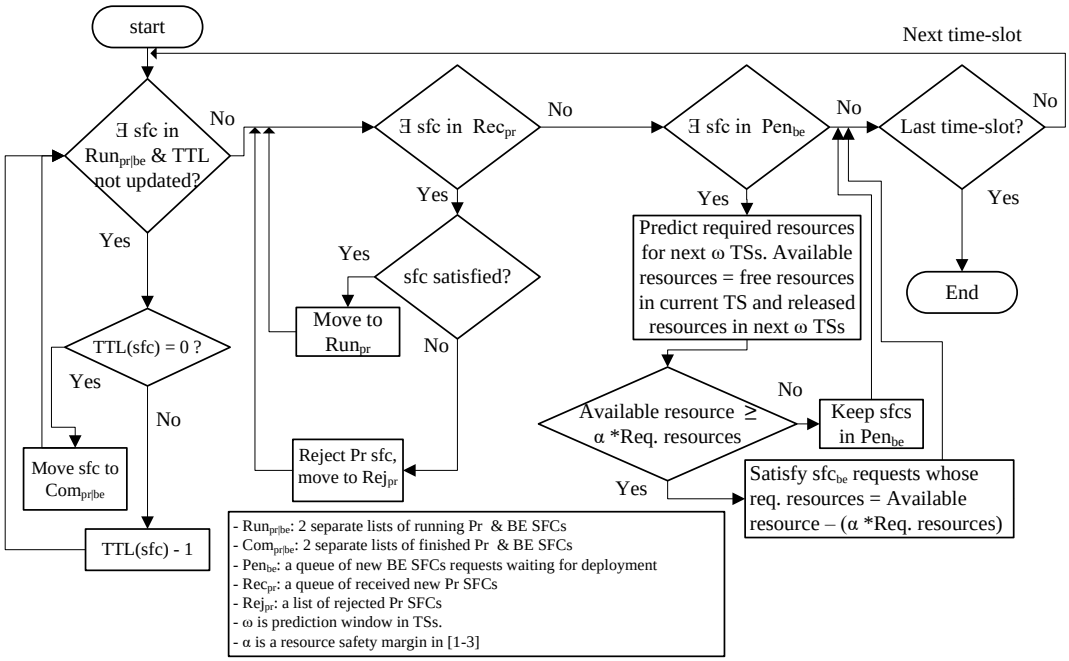


Figure 5.3: PSVS scheme logical flow.

Placement Algorithm

With an SFC request sfc_j consisting of VNFs $v_i, i \in [1, |sfc_j|]$, the decision variables are: χ_{in}^{j}: if a new instance of VNF v_i belonging to sfc_j is to be placed at node

Algorithm 5.1: PSVS

// SimDur: Simulation duration in time slots (TSs), $\omega \in [1,3]$ TSs Lookahead window length, and $\alpha \in [1,3]$ Resource *safety margin* (a scaling factor)

Input : net-Model, SimDur, ω, α
Init. : Rec_{pr}, $Run_{pr|be}$, Pen_{be}, Rej_{pr}, $Com_{pr|be}$
Output: Different queues\lists and collected statistics

for $i \leftarrow 1$ **to** SimDur **do**
  $Rec_{pr} \leftarrow$ received sfc_{pr} requests
  $Pen_{be} \leftarrow$ received sfc_{be} requests
  // Update TTL of running SFCs
  foreach $sfc_{pr|be}$ *in* $Run_{pr|be}$ **do**
    if ttl($sfc_{pr|be}$) $= 0$ **then**
      // Releases resources utilised by finished sfc
      $Com_{pr|be} \leftarrow sfc_{pr|be}$
    else decTTL($sfc_{pr|be}$)
  foreach sfc_{pr} *in* Rec_{pr} **do**
    // Using IQCP & Gurobi solver
    sol $\leftarrow$satisfy(sfc_{pr},net-Model)
    if sol $\neq \emptyset$ **then**
      deploy(sfc_{pr},net-Model)
      $Run_{pr} \leftarrow sfc_{pr}$
    else // sfc_{pr} can not tolerate waiting for deployment
      $Rej_{pr} \leftarrow sfc_{pr}$
  // Required resources/sfc_{pr} requests expected to arrive in ω TSs
  $ReqrdRes_{pr} \leftarrow$ predReqResources(ω)
  // *AvailRes*: free resource in current TS and in next ω TSs
  $AvailRes \leftarrow$ getAvailResources(ω)
  $ExtraRes_{be} \leftarrow 0$
  if $AvailRes > (\alpha * ReqrdRes_{pr})$ **then**
    $ExtraRes_{be} \leftarrow AvailRes - (\alpha * ReqrdRes_{pr})$
  while $(\exists\, sfc_{be}$ *in* $Pen_{be}) \& (ExtraRes_{be} \neq 0)$ **do**
    sol $\leftarrow$satisfy(sfc_{be},net-Model)
    if sol $\neq \emptyset$ **then**
      deploy(sfc_{be},net-Model)
      $Run_{be} \leftarrow sfc_{be}$
      $ExtraRes_{be} \leftarrow ExtraRes_{be}$-usedRes($sfc_{be}$)
    // else sfc stays in Pen_{be}

n; and R^j_{in}: if 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 5.2.

Objective Function The objective function, equation (5.1), is designed to favour the placements that minimize the overall cost and resource utilization by preferring sharing underutilized VNFs over instantiating new ones. The feasible solution has to satisfy constraints 5.2-5.9. Objective function and some constraints are the same as those used in previous chapters, however, they are listing here for completeness.

$$min \sum_{i=1}^{|sfc_j|} \sum_{n \in N} [cpu(v^j_i) U^c_{cpu}(n) + ram(v^j_i) U^c_{ram}(n)] \chi^j_{in} + F_{out}(v^j_i) U_c bw [\chi^j_{in} + R^j_{in}] \tag{5.1}$$

$$\sum_{n \in N} \chi^j_{in} + R^j_{in} = 1 \quad , \quad \forall i \in [1, |sfc_j|] \tag{5.2}$$

$$\sum_{n \in N} \chi^j_{in} + D^i_n \, S(v_i) \, R^j_{in} = 1, \quad \forall i \in [1, |sfc_j|] \tag{5.3}$$

$$F_{in}(v^j_i) \, R^j_{in} \leq F_{av}(D^i_n), \; \forall n \in N \; \& \; \forall i \in [1, |sfc_j|] \tag{5.4}$$

$$\sum_{i=1}^{|sfc_j|} cpu(v^j_i) \, \chi^j_{in} \leq cpu_{av}(n), \quad \forall n \in N \tag{5.5}$$

$$\sum_{i=1}^{|sfc_j|} ram(v^j_i) \, \chi^j_{in} \leq ram_{av}(n), \quad \forall n \in N \tag{5.6}$$

$$e_{nn'} \, (\chi^j_{in} + R^j_{in})(\chi^j_{(i+1)n'} + R^j_{(i+1)n'}) = 1 \quad \forall n, n' \in N \; \& \; \forall i \in [1, (|sfc_j| - 1)] \tag{5.7}$$

Table 5.2: Queues, Decision Variables and Constants.

Variable/Queue	**Description**
χ^j_{in}	For placing a new VNF v_i of sfc_j at node n
R^j_{in}	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 is deployed at node n
$F_{av}(v^i_n)$	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
Rec_{pr}	Queue of new Pr SFCs until deployed
Pen_{be}	Queue of new BE SFCs until deployed
Rej_{pr}	List of rejected Pr SFCs
$Run_{pr\|be}$	List of deployed Pr or BE SFCs
$Com_{pr\|be}$	A list of finished Pr or BE SFCs

$$\sum_{n\in N}\sum_{n'\in N} F_{out}(v^j_i)(\chi^j_{in} + R^j_{in})(\chi^j_{(i+1)n'} + R^j_{(i+1)n'}) \leq bw_{av}(e_{nn'}), \quad \forall i \in [1,(|sfc_j|-1)] \tag{5.8}$$

$$\sum_{i=1}^{|sfc_j|-1}\sum_{n\in N}\sum_{n'\in N} Del(e_{nn'})\,(\chi^j_{in} + R^j_{in})\,(\chi^j_{(i+1)n'} + R^j_{(i+1)n'}) \leq Del(sfc_j) \tag{5.9}$$

Required Resources Prediction

The assumption here is that a simple prediction model exists and based on the historical data, it can predict the number of sfc_{pr} requests to arrive in the next ω TSs and their required resources. As can be seen in line 18 of Algorithm 5.1, when comparing predicted required resources ($ReqrdRes_{pr}$) to available resources ($AvailRes$) we utilize a *safety margin* α.

The main purpose of using α is to compensate for the difference in TSs between the size of the lookahead window $\omega \in [1, 3]$ and the duration of sfc_{be} requests (typically 10 or 20 TSs). For example, for $\alpha = 1$, if the $AvailRes$ is greater than predicted $ReqrdRes_{pr}$, a deployed sfc_{be} in the current TS will have a lasting impact on resources availability that will extend beyond the lookahead window, hence the sfc_{pr} requests rejections. The value of *safety margin* α should be empirically selected/tuned. On the one hand, small values of α will lead to a higher rejection rate. On the other hand, large values can cause starvation for the pending sfc_{be} requests as it will result in overbooking resources for future sfc_{pr} requests, resulting in lower system utilization and prolonged waiting times for sfc_{be} requests. It is worth noting that, the $AvailRes$ are those resources free in the current TS in addition to resources that will be released in the next ω TSs as running SFCs are finished, TTL=0, and moved to $Com_{pr|be}$ list. Extra resources ($ExtraRes_{be}$) are those available resources that exceed the α-scaled $AvailRes$ and are utilized to satisfy one or more requests pending in the Pen_{be} queue (see Algorithm 5.1 lines 19-25).

5.4 Performance Evaluation

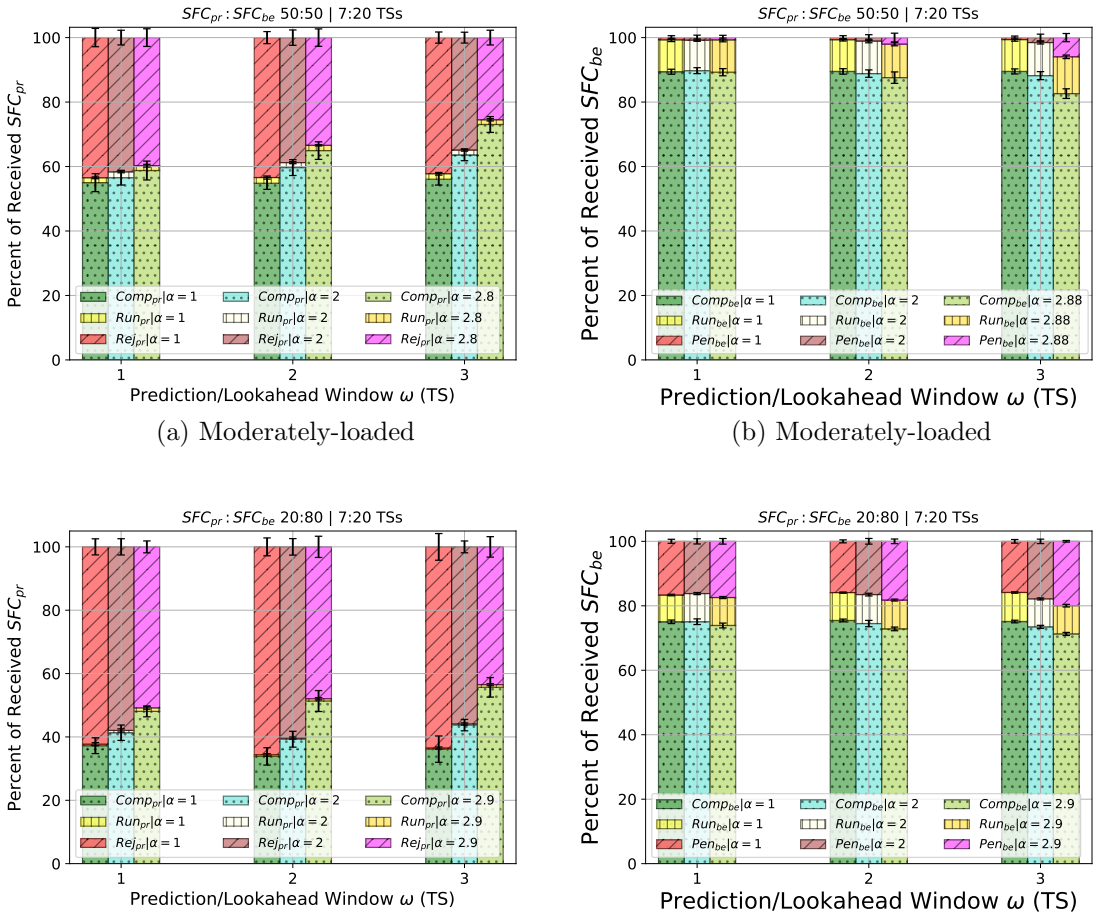
5.4.1 Simulation Framework

We utilize the same Java-based simulation environment used in previous chapter to synthetically generate SFC requests and the network model. The SFC requests arrival rate per TS follows a Poisson distribution with an average rate $\lambda = 2$. SFC length, number of VNFs, is drawn from a uniform distribution $|sfc_j| \sim U[4, 7]$ [58]. The service time, i.e., SFC duration in TSs, is fixed, where $sfc_{pr} = 7$ and $sfc_{be} \in \{10, 20\}$. The ratio of sfc_{pr}:sfc_{be} requests is either *'50:50'* or *'20:80,'* and the order of arrival is

shuffled. Each simulation is repeated ten times where the network model, nodes, and links' resources are the only constant. Across the ten repetitions, the variations are number and type of requests per TS, the length and type of VNFs in sfc requests, and sfc requests performance requirements (maximum end-to-end delay). We utilized a list of onboarded VNFs that contains 16 VNFs, 60% of which are shareable. The placement scheme is solved using the Gurobi solver [56].

5.4.2 Numerical Results and Analysis

To assess the impact of both lookahead window size ω and *safety margin* α on the Rej_{pr}%, we experiment with different values of ω and α, and with different system loads. As can be seen in Figures 5.4a and 5.4b, the *moderately-loaded* system witnesses an increase in completed requests ($Com_{pr|be}$), significantly fewer rejected requests (Rej_{pr}), and about the same percentage of received requests that are in the running state ($Run_{pr|be}$). When comparing the rejection rate of $\alpha = \{1, 2, 2.8, 2.9\}$ and $\omega = 3$, we can conclude that the *safety margin* has a significant impact on reducing the sfc_{pr} rejection rate. The same effect is carried over to the *highly-loaded* system, as shown in Figures 5.4c and 5.4d. Even though the +50% reduction in the rejection rate of sfc_{pr} requests, the Pen_{be} queue did not expand that much (compare the size of Pen_{be} of $\omega = 3$ and $\alpha = 2.9$ for the *highly-loaded* system. The best values that α converged to might seem a magic number. However, we can easily find simpler, more deterministic systems that use the same empirical methodology to determine the best value/range for a probability/factor that maximizes an important *KPI*. For example, how the *'p'* in *p*-persistent carrier sense multiple access (CSMA) is calculated to maximize throughput [107].



(a) Moderately-loaded (b) Moderately-loaded

(c) Highly-loaded (d) Highly-loaded

Figure 5.4: Rejected/Pending, running, and completed (%) of received sfc_{pr} and sfc_{be} requests for two system configurations.

The utilization of both *moderately* and *highly-loaded* systems, shown in Figure 5.5, reveals that PSVS does not increase the completion and reduces the rejection rates of sfc_{pr} requests by totally ignoring the sfc_{be} requests and leaving the system resources idle. Furthermore, PSVS enhances the overall utilization of the *moderately-loaded* system. For the system utilization of different ω and α, we can consult Figure 5.4 to see that the size of $Run_{pr|be}$ of both system loads is almost the same for different

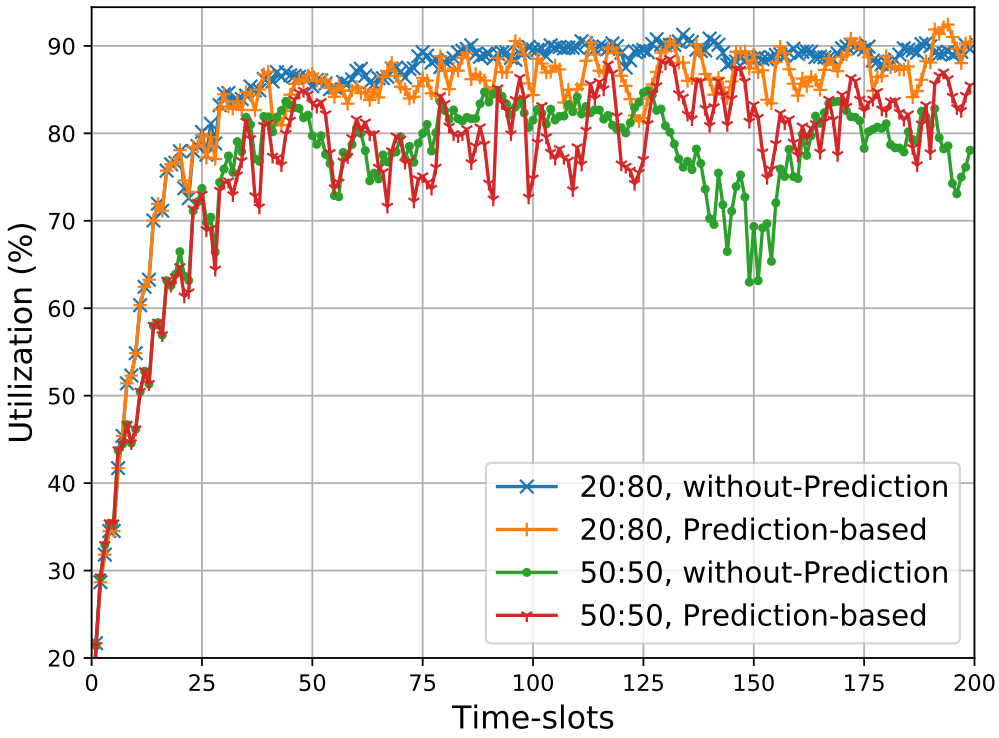


Figure 5.5: Utilization throughout simulation time (200 TSs).

values of ω and α.

There are two faces to the PSVS schemes, the impact on the rejection rate of sfc_{pr} requests and on the AWT and pending sfc_{be} requests. Both the AWT and percentage of pending sfc_{be} requests are reported in Figure 5.6. PSVS increased the average waiting time (AWT) and the percentage of pending sfc_{be} requests by 43% and 38%, respectively.

In addition to compensating for the difference between the short lookahead window (for accurate predictions) and the 20TS-long sfc_{be} requests, the *safety margin* α helps address the uncertainty arising from the VNF sharing (the used resources to satisfy requests varies depending on the current snapshot of deployed SFCs) and increases the

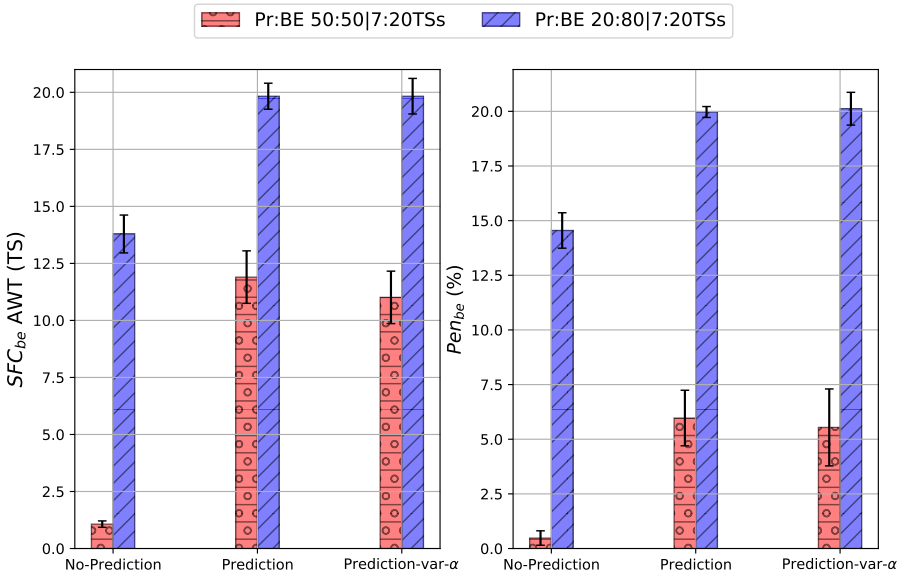
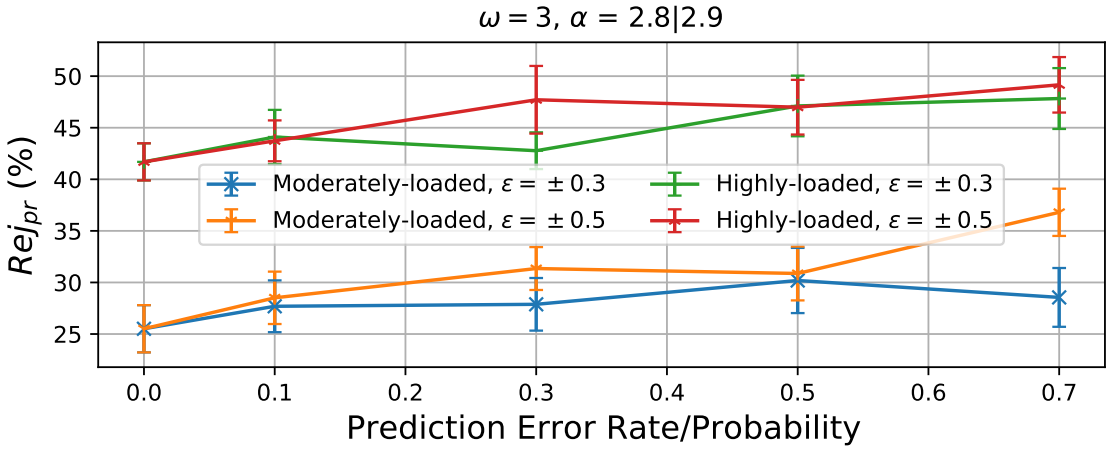


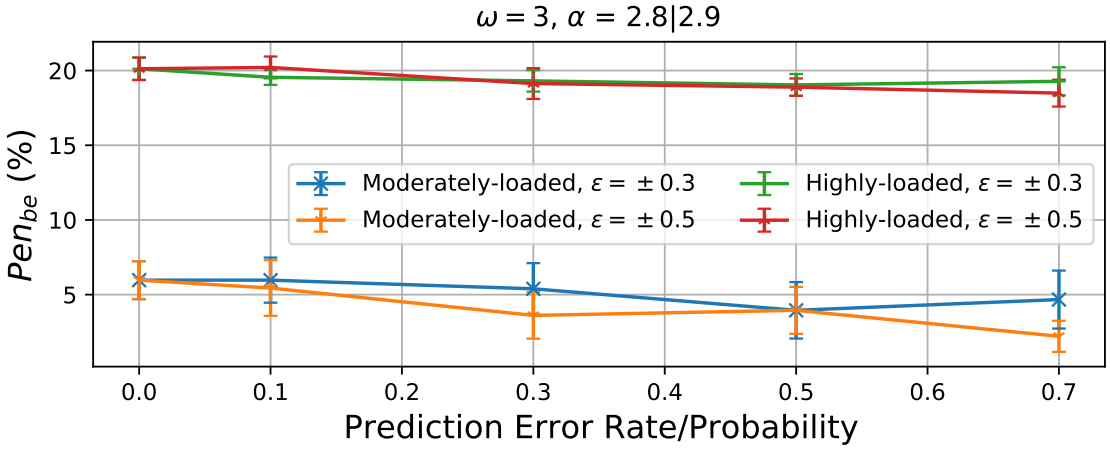
Figure 5.6: Left: sfc_{be} average waiting time (AWT). Right: percentage of pending sfc_{be} requests. *Prediction* is using $\alpha = 2.8$ for 50:50 systems and $\alpha = 2.9$ for 20:80 system. The var-α uses a varying *safety margin* with lower-bound=2.8, upper-bound=2.9, the increase step=0.1 and the decreasing step=0.01.

robustness of PSVS and its resilience to prediction errors. To evaluate the robustness of PSVS results and demonstrate the importance of *safety margin* α, we experiment with error rates/probabilities up to 70% of predicted required resources and with a prediction error value of ε up to $\pm$50%. As shown in Figure 5.7a, the biggest increase in the *highly-loaded* system's rejection rate is only 17% in the face of extreme error values and rates. As shown in Figure 5.7b, the Pen_{be} queue size did not grow.

Figure 5.8 shows the impact of *safety margin* α on sfc_{pr} requests rejection rate under extreme prediction error values and rates/probabilities. For both system loads, the higher the value of α, the less the rejection rate. In the same vein, the increase in the Pen_{be} queue size is not concerning; see Figure 5.8b.

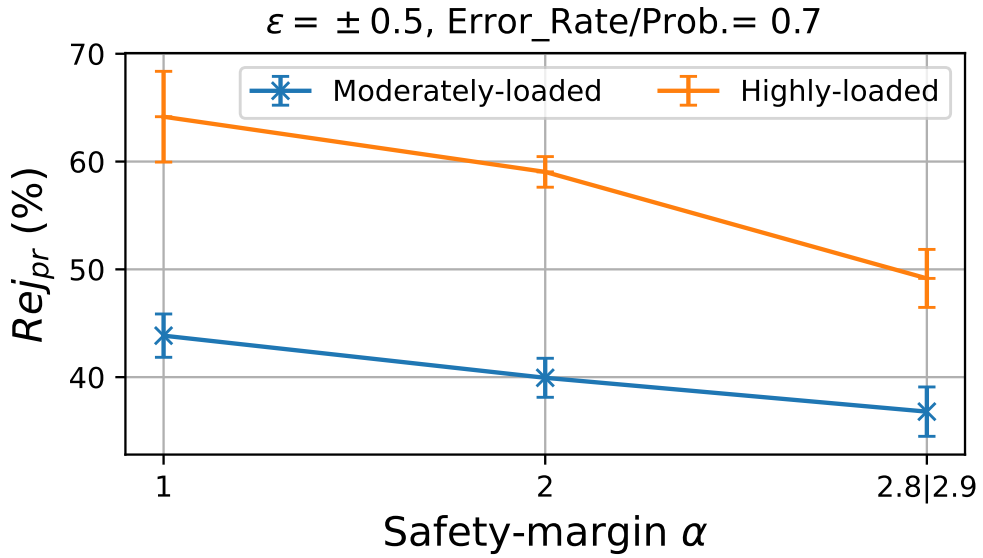


(a) Moderately-loaded

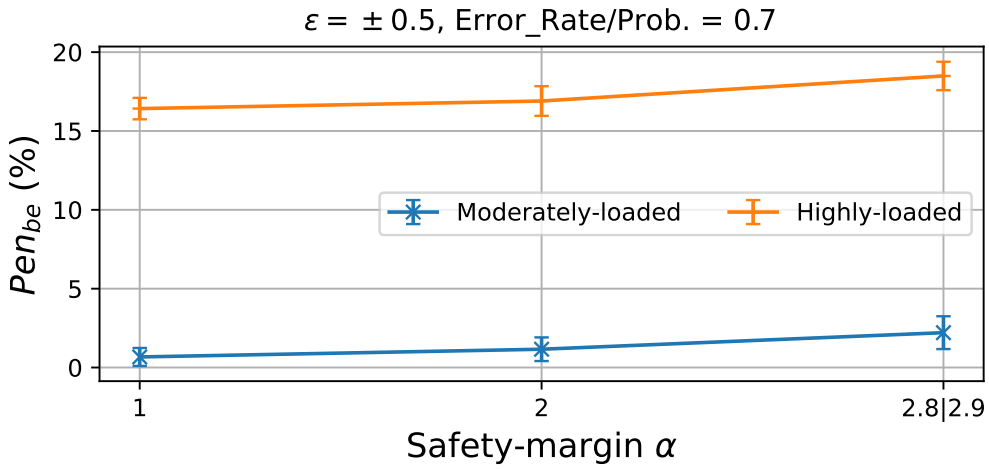


(b) Highly-loaded

Figure 5.7: The impact of prediction error value and rates/probabilities on (a) Rejected Pr SFCs requests and (b) Pending BE SFC requests, for moderately- and highly-loaded systems.



(a) Moderately-loaded



(b) Highly-loaded

Figure 5.8: The impact of *safety margin* on sfc_{pr} requests rejection rate under extreme prediction error values and rates/probabilities, (a) Rejected Pr SFCs requests and (b) Pending BE SFC requests.

5.5 Summary

In this chapter, we present PSVS to reduce the rejection rate of time-sensitive Pr services at the edge. PSVS utilizes both the predicted required resources and a *safety margin* to address the difference between ω and sfc_{be} request duration and to provide resiliency against prediction errors. We experiment and evaluate different lookahead window sizes ω and *safety margin* α and conclude the best values to balance the reduction in sfc_{pr} requests rejection rate and sfc_{be} requests AWT. Finally, more reduction in the rejection rate is attainable; however, the sfc_{be} requests will suffer more starvation. This might be desirable in emergencies where time-critical premium services must be immediately satisfied.

Rejections of sfc_{pr} requests are unavoidable with a prediction window size ω smaller than sfc_{be} requests' average duration. However, to get accurate predictions, we had to continue using small lookahead window sizes. In an environment where lower-priority services can be suspended, and the preemption cost is bearable, a zero sfc_{pr} rejections might be achievable by suspending one or more running BE services and preempting resources to premium services. Future research includes a preemption criterion that preempts resources for sfc_{pr} requests and minimizes the disturbance to BE services.

Chapter 6

Immediate Placement of Time-critical Services

"5G cell networks create a whole new wave of distributed and edge computing"

Michael Dell

6.1 Introduction

Time-critical premium (Pr) services and applications are a class of software that have strict time constraints, that if such constraints were not met a Pr service would fail [54, 61, 78]. 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 the cloud, and its resources are limited compared to the cloud. The demand generated by time-critical applications necessitates the 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.

To address these two needs, efficient utilization of edge resources and immediate

satisfaction of Pr service/SFC requests, we propose an immediate placement of time-critical SFCs with VNF sharing (IPTSV). 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 SFC requests by sharing underutilized shareable VNFs. With limited resources 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. As a remedy to the side effects of IPTSV, we design a preemptive prediction-based placement scheme that only utilizes preemption in cases where the prediction fails to secure the required resources for sfc_{pr} requests.
The main contributions of this chapter are:

- Introduction of the 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.
- Introduction of preemptive prediction-based placement scheme to reduce the side effects of preemptive-based placement on running sfc_{be}.

The remainder of this chapter is structured as follows. Proposed time-critical SFC placement, system model, and problem formulation are detailed in Section 6.2.

In Section 6.3, we detail the simulation framework. Performance evaluation and results analysis are covered in Section 6.4. The preemptive prediction-based placement scheme is detailed in Section 6.5. Conclusions and future work is presented in Section 6.6.

6.2 System Model and Problem Formulation

In the 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, as in previous two chapters, we are utilizing only two service categories, Pr and best-effort (BE) service categories.

Despite using *VNF sharing*, there is a possibility that a received Pr SFC request cannot be satisfied due to insufficient resources. Based on extensive simulations of SFC placement with *VNF sharing*, the rejection rate of Pr SFCs is unavoidable and concerning. As shown in Figure 5.1, the best-case scenario is '50:50—VarDuration' the *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.

We propose the 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 running BE SFC(s). The preemption criterion should strive to balance

between immediately satisfying Pr SFCs and minimizing the number of disturbed BE SFCs. 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 has 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 its lower priority, BE SFCs can be in states: received, running, pending redeployment or completed. The states and actions that trigger state changes of SFCs are detailed in Figure 6.1.

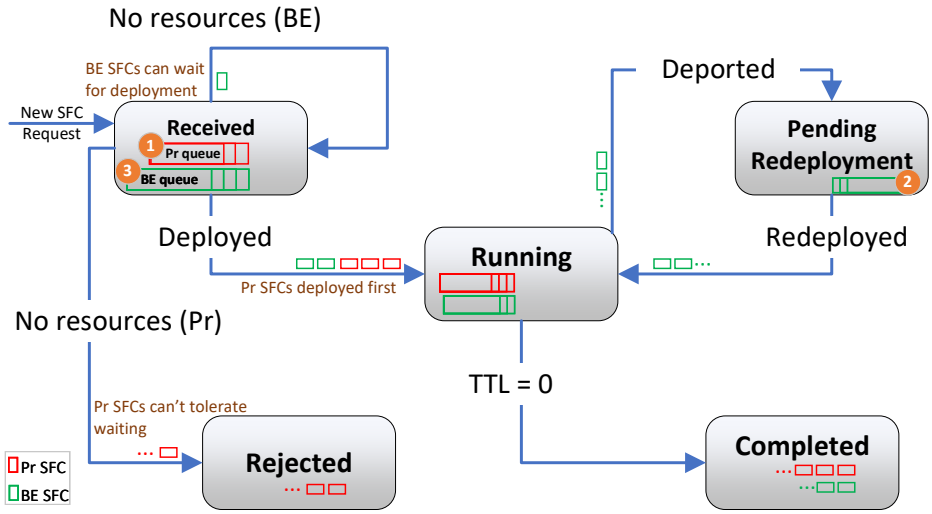


Figure 6.1: States a SFC request can take in IPTSV and order of priority and deployment of queues ($1 \rightarrow 2 \rightarrow 3$).

6.2.1 System Model

The list of on-boarded VNFs list 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 operation 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 its compute resources, *cpu cores*, and *memory*, and each link $e \in E$ has bandwidth capacity bw_c and propagation delay Del. The topology of the substrate network is determined by a configurable connectivity matrix. Table 6.1 lists a detailed description of the substrate network and SFC parameters (very few differences between this chapter's tables and previous chapters, however, they are listed here for completeness).

6.2.2 Problem Formulation

The logic flow the IPTSV scheme is shown in Figure 6.2 and in Algorithm 6.1. The IPTSV scheme hinges critically on two main components, the placement algorithm and the preemption criterion. The different scores used in the preemption criteria are detailed below in the (*Preemption Criteria*) subsection. The *VNF sharing*-based placement algorithm is modelled as an integer quadratically-constrained program (IQCP) with binary decision variables. The IPTSV scheme manages SFC requests arriving at each time slot (TS). First, IPTSV looks for deployed SFCs in list $Run_{pr|be}$ with time-to-live (TTL), in TSs, value equal to zero to terminate and release the resources; otherwise, it decreases TTL value by one. Second, IPTSV considers

Table 6.1: 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

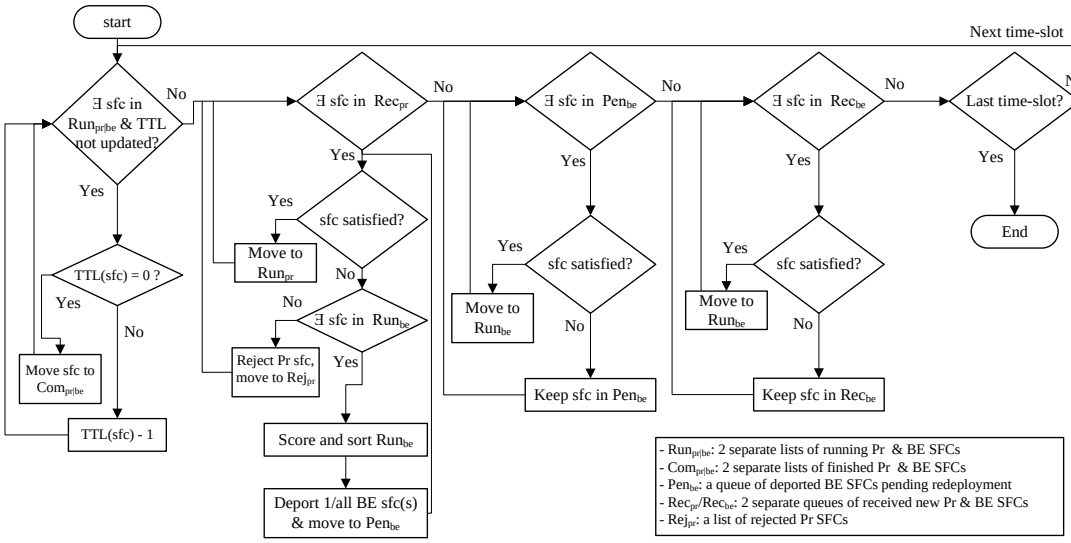


Figure 6.2: IPTSV scheme Logical flow.

requests in the 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 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 indefinitely in queues waiting for deployment or redeployment.

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

Algorithm 6.1: 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 ← 1 to No.TSs do
    Rec_{pr} ← received Pr requests
    Rec_{BE} ← 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} ← sfc_{pr|be}
        else decTTL(sfc_{pr|be})
    foreach sfc_{pr} in Rec_{pr} do
        // Using IQCP & Gurobi solver
        sol ←satisfy(sfc_{pr},netModel)
        if sol ≠ ∅ then
            deploy(sfc_{pr},netModel)
            Run_{pr} ← sfc_{pr}
        else // Check preemptCPU algorithm for details
        // sfc_{pr} is only used with 'similar' preemption criterion
        sol,Pen_{be} ←preemptCPU(criterion,Run_{be})
        if sol ≠ ∅ then
            deploy(sfc_{pr},netModel)
            Run_{pr} ← sfc_{pr}
        else Rej_{pr} ← sfc_{pr}
    foreach sfc_{be} in Pen_{be} do
        sol ←satisfy(sfc_{be},netModel)
        if sol ≠ ∅ then
            deploy(sfc_{be},netModel)
            Run_{be} ← sfc_{be}
        // else sfc stays in Pen_{be}
    foreach sfc_{be} in Rec_{be} do
        sol ←satisfy(sfc_{be},netModel)
        if sol ≠ ∅ then
            deploy(sfc_{be},netModel)
            Run_{be} ← sfc_{be}
        // else sfc stays in Rec_{be}
```

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 6.2.

Objective Function The objective is to select the placement that minimizes the overall cost, hence, optimizing resource utilization. The objective function in equation (6.1) is formulated to prioritize sharing over deploying new VNF instances. Objective function and constraints are the same as those in chapter 5, however, we are listing them here for completeness.

$$min \sum_{i=1}^{|sfc_j|} \sum_{n \in N} [cpu(v_i^j) U_{cpu}^{c}(n) + ram(v_i^j) U_{ram}^{c}(n)] X_{in}^{j} + F_{out}(v_i^j) U_c bw [X_{in}^{j} + R_{in}^{j}] \tag{6.1}$$

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 (6.2). If sharing is the decision, a shareable VNF of the same type must have been deployed, equation (6.3), and its free capacity is a sufficient amount for SFC's VNF inflow, equation (6.4). If a new instance is to be deployed, X_{in}^{j} to be valid, substrate node n must have the required resources, i.e., *cpu* and *ram*, equations (6.5 & 6.6). Equations (6.7) and (6.8) 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_i^j. Finally, equation (6.9) is to guarantee that the quality-of-services (QoS) requirements

Table 6.2: 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_{in}^{j}	Binary decision for placing VNF v_i of sfc_j at node n
R_{in}^{j}	Binary decision for sharing the flow of VNF v_i of sfc_j with already deployed VNF of same type at node n
D_{n}^{i}	VNF of same type as v_i already deployed at node n
$F_{av}(D_{n}^{i})$	Available unused flow of v_i at node n
$U_{cpu}^{c}(n)$	Unit cost of cpu at node n
$U_{ram}^{c}(n)$	Unit cost of ram at node n
$U_{c}(bw)$	Unit cost of bw at all links

(end-to-end latency) of sfc_j are satisfied.

$$\sum_{n \in N} X_{in}^{j} \; + \; R_{in}^{j} \; = \; 1 \quad , \quad \forall i \in [1, |sfc_j|] \tag{6.2}$$

$$\sum_{n \in N} X_{in}^{j} \; + \; D_{i}^{j} \; S(v_i) \; R_{in}^{j} \; = \; 1, \quad \forall i \in [1, |sfc_j|] \tag{6.3}$$

$$F_{in}(v_i^j) \; R_{in}^{j} \; \leq \; F_{av}(D_{n}^{i}), \; \forall n \in N \; \& \; \forall i \in [1, |sfc_j|] \tag{6.4}$$

$$\sum_{i=1}^{|sfc_j|} cpu(v_i^j) \; X_{in}^{j} \; \leq \; cpu_{av}(n), \quad \forall n \in N \tag{6.5}$$

$$\sum_{i=1}^{|sfc_j|} ram(v_i^j) \; X_{in}^{j} \; \leq \; ram_{av}(n), \quad \forall n \in N \tag{6.6}$$

$$e_{nn'}\,(X^j_{in}\,+\,R^j_{in})(X^j_{(i+1)n'}\,+\,R^j_{(i+1)n'})=1 \quad \forall n,n^{'}\in N\;\&\;\forall i\in[1,(|sfc_j|-1)] \tag{6.7}$$

$$\sum_{n\in N}\sum_{n'\in N} F_{out}(v^j_i)(X^j_{in}\,+\,R^j_{in})(X^j_{(i+1)n'}\,+\,R^j_{(i+1)n'}) \leq bw_{av}(e_{nn'}),\quad \forall i\in[1,(|sfc_j|-1)] \tag{6.8}$$

$$\sum_{i=1}^{|sfc_j|-1}\sum_{n\in N}\sum_{n'\in N} Del(e_{nn'})\,(X^j_{in}\,+\,R^j_{in})\,(X^j_{(i+1)n'}\,+\,R^j_{(i+1)n'})\leq Del(sfc_j) \tag{6.9}$$

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 the SFC is supposed to be utilizing. This is why we propose to study a different number of scoring and preemption criteria to produce a recommendation for the best criterion for certain contexts.

As in Algorithm 6.2, 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 the 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 6.2: preemptCPU

Input : netModel, criterion, sfc_{pr}, Run_{be}
Output: sol, Pen_{be}

```
1  if criterion ≠ Random then
       // sfc_pr is needed for 'similar' criterion
       // sort() sorts 'as.' ↑ or 'dec.' ↓ based-on given criterion
2      Run_be ← sort(score(Run_be,criterion, sfc_pr),[as|dec])
3  sol ← ∅
4  while sol = ∅ and len(Run_be) ≠ 0 do
5      if criterion = Random then
           // Run_be isn't sorted
6          sfc_be ← getRandSfc(Run_be)
7      else
           // Run_be is sorted
8          sfc_be ← Run_be[0]
9      deport(sfc_be)
10     Pen_be ← sfc_be
11     sol ← 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 the number of disturbed BE SFCs (reduce gratuitously deported SFCs) and the 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 depends 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 (6.10), the similarity score is the inverse of the difference score, which includes: the number of different VNFs, Pr-length minus BE-length, Pr-max-outflow minus BE-max-outflow, the difference in required CPU cores, and the total number of SFCs that BE SFC is hosting. Each difference term in equation (6.10) is normalized against its peers

calculated for all BE SFCs before calculating the similarity score.

$$SIM(sfc_{pr}, sfc_{be}) = 1/\{V_{diff}(sfc_{pr,be}) + (|sfc_{pr}| - |sfc_{be}|) + [max(f_{out}(v_i^{pr})) - max(f_{out}(v_i^{be}))] + [\sum_{i=1}^{|sfc_{pr}|} cpu(v_i^{pr}) - \sum_{i=1}^{|sfc_{be}|} cpu(v_i^{be}) + Gst(sfc_{be})\} \quad (6.10)$$

The $deport(sfc_{be})$ procedure, in Algorithm 6.2 line 9, is accomplished with VNF sharing considered. In VNF-sharing, shareable VNFs are either host or guest. For example, in Figure 6.3, 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, and the shared capacity is simply returned 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.

6.3 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 [26,101]. Therefore, we decided to synthetically generate SFC requests per each TS. The arrival of SFC requests per TS follows a Poisson distribution with an average rate $\lambda = 2$.

We utilize the Java-based simulation environment used in previous chapters. In

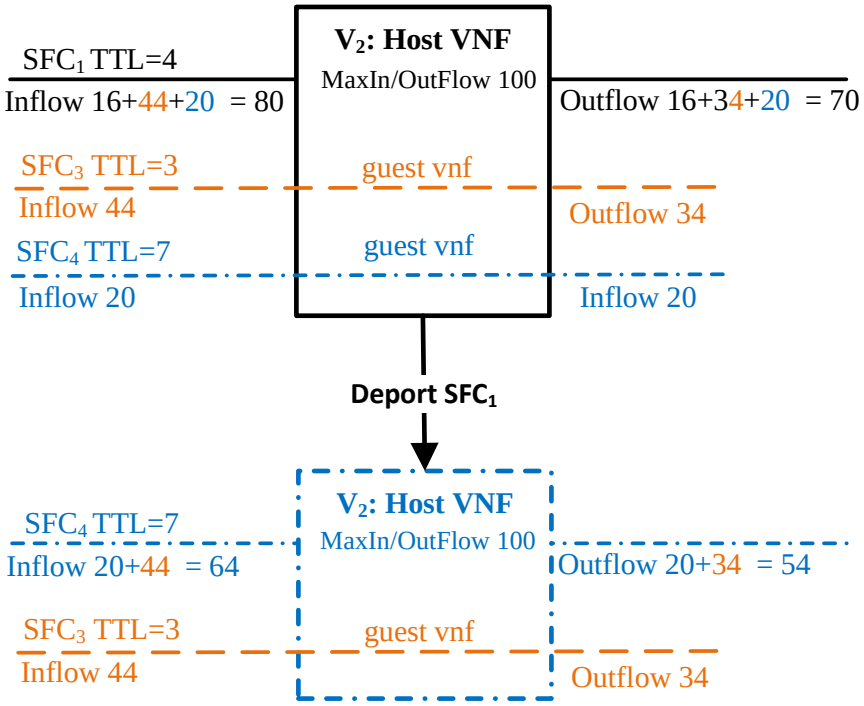


Figure 6.3: Promoting guest VNF to act as a host, part of sfc_{be} deportion

which, we generate a 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 $|sfc_j| \sim U[4, 7]$ [58, 68]. The service time, i.e., SFC duration in TSs, is fixed, where $sfc_{pr} = 7$ and $sfc_{br} \in \{5, 20\}$. 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 is either *'50:50'* or *'20:80,'* and the order of arrival is randomly shuffled.

In our simulations, for all simulations, we used the same network model and the same topology used in previous chapter. 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 were 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 [56].

6.4 Performance Evaluation

6.4.1 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 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 and the 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 simulations 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.

6.4.2 Numerical Results and Analysis

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

As shown in Figure 6.4, 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 that the longer SFCs will have to stay in the system, the higher the rejection rate will be. As illustrated in Figure 6.5, 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

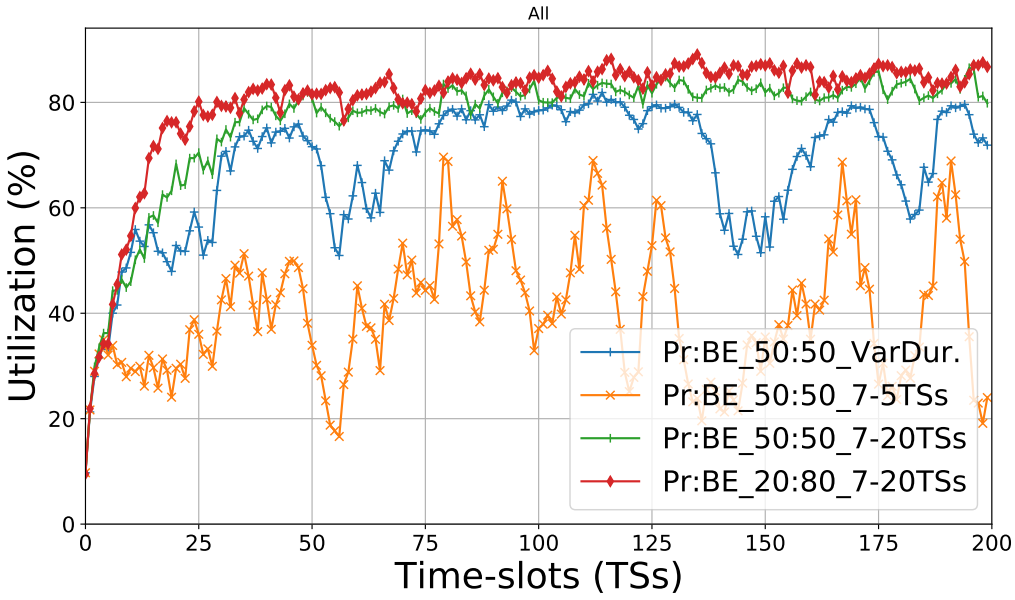


Figure 6.4: Resource utilization for different system loads for '*All*' preemption criterion

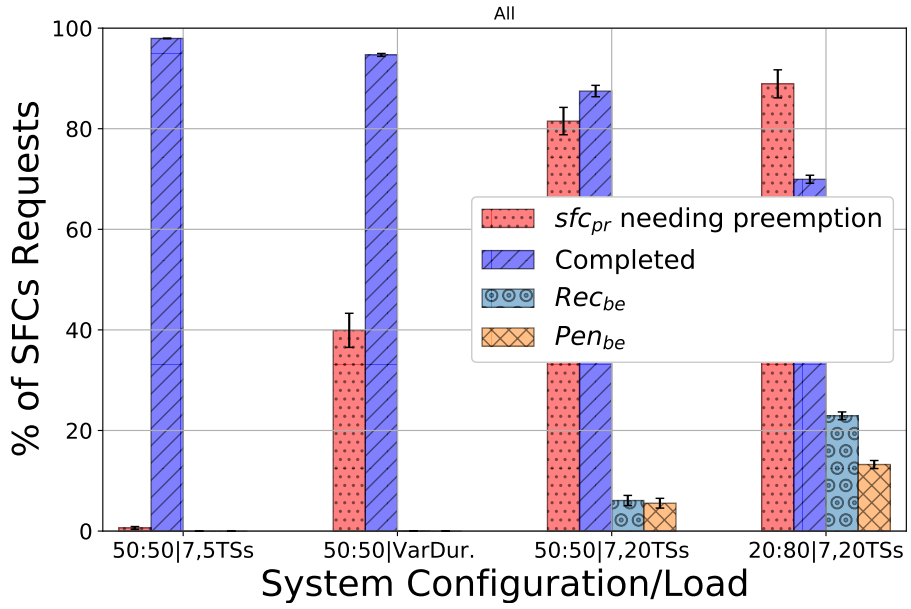


Figure 6.5: End-of-simulation queue sizes for different system loads for '*All*' preemption criterion

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 simulations, 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 measure 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 6.3, 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 a 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 do 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 6.6 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,

Table 6.3: 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

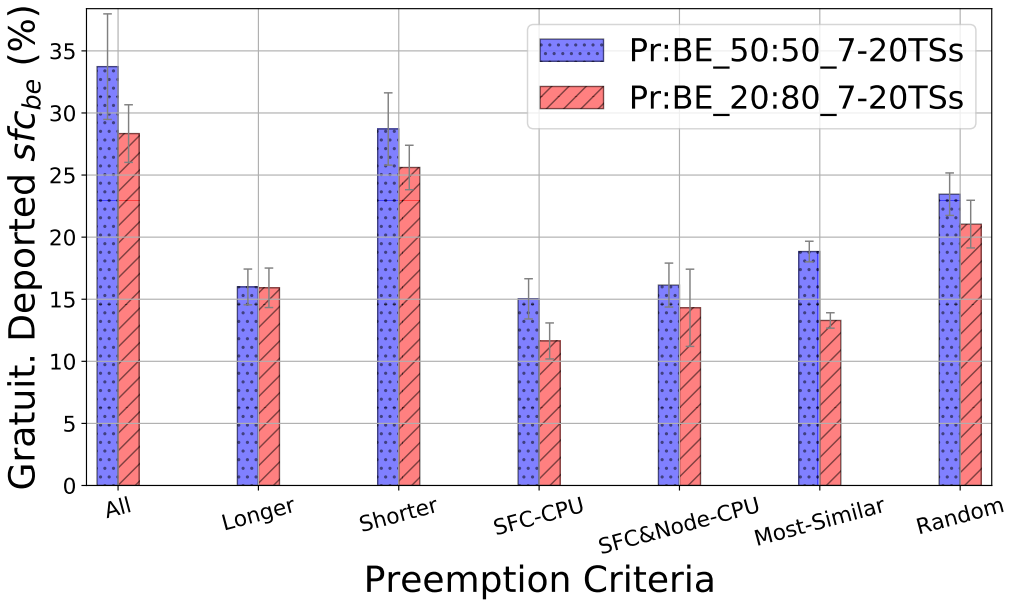


Figure 6.6: Gratuitously deported sfc_{be}.

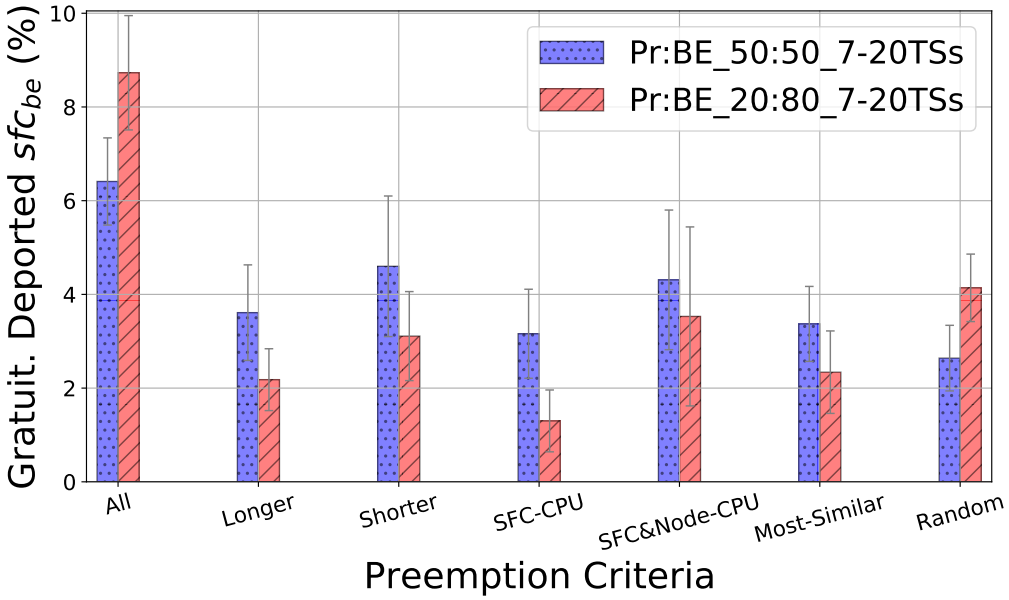


Figure 6.7: Gratuitously deported sfc_{be} (No-Sharing).

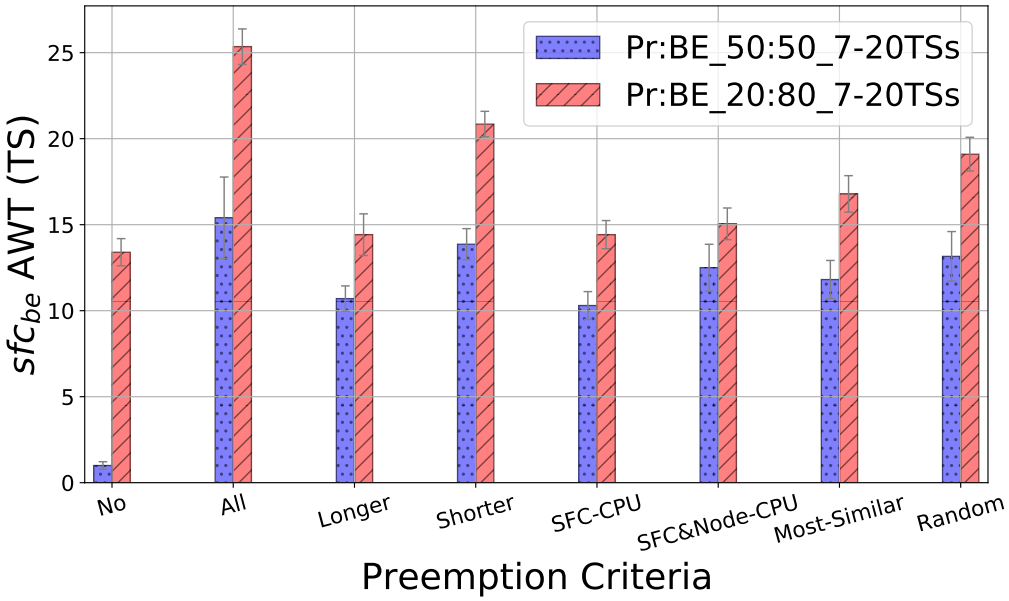


Figure 6.8: AWT of sfc_{be}.

the probability is higher that this is the best-case scenario, i.e., deporting only one BE SFC or the 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 reason is (as explained in subsection(*Preemption Criteria*) and in Figure 6.3) 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 6.7. The overall percentage of gratuitously deported BE SFCs significantly dropped by 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 the 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. The Result of the *'No-preemption'* criterion is used as a reference lower-bound of BE SFC AWT. The concluding results of the simulation are shown in Figure 6.8, 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 the *'20:80'* system, the AWT of the *'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 the average TAT of BE SFCs figure.

We formulate equation (6.11) as the preemption cost function. It is a function of j and k, where $j \in [0, |Run_{be}|]$ is the number of deported BE SFCs to satisfy one Pr

SFC and $k \in [4 * |Run_{be}|, 7 * |Run_{be}|]$ is the 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, $c_1 = c_2 = c_3 = 1$.

$$Cost(j,k) = \begin{cases} c_1.j + c_2(1+j) + c_3.k & ,\text{`}All\text{'}\ criterion \\ c_1.j + 2c_2.j + c_3.k & ,otherwise \end{cases} \tag{6.11}$$

As shown in Figure 6.9, the criteria that has the least preemption cost for *'50:50'*

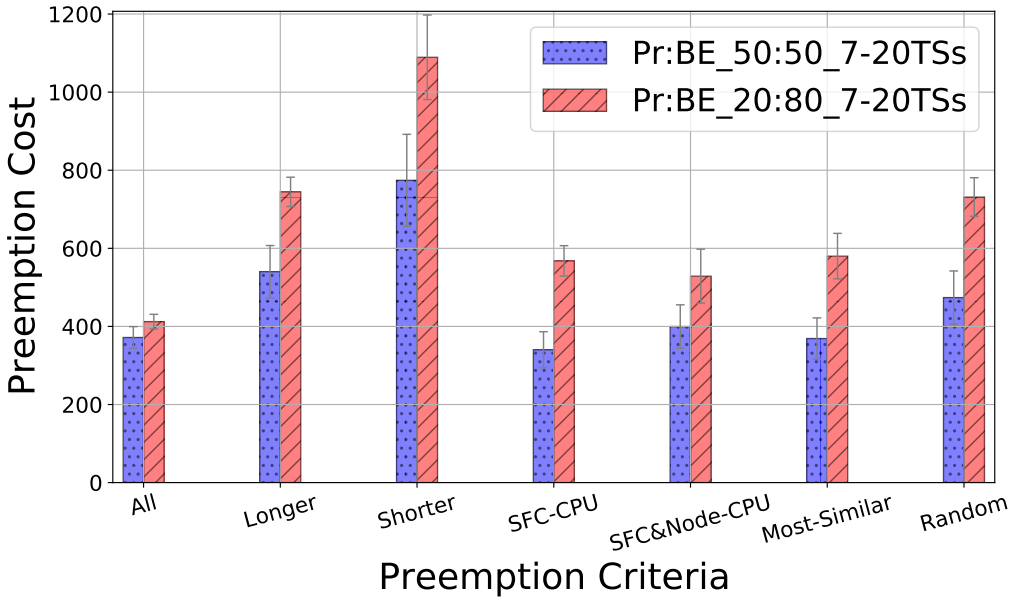


Figure 6.9: preemption Cost of preemption criteria

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. As expected, because of the unpredictable nature

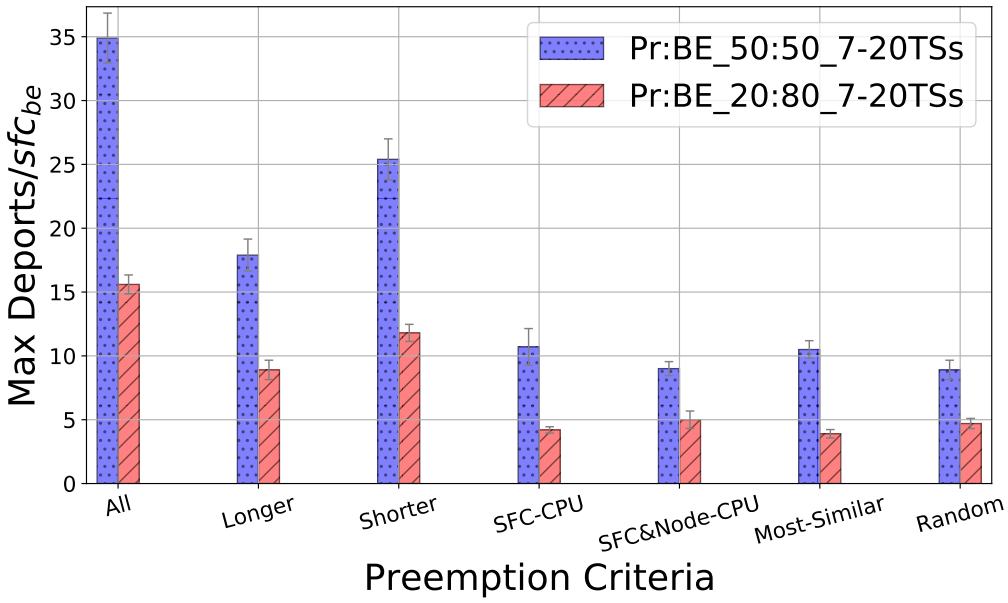


Figure 6.10: Fairness of preemption criteria

of the *'Random'* criterion, it fairly deports SFCs in Run_{be}, as shown in Figure 6.10. Using the *'Random'* criterion as a fairness reference, we can see that *'SFC&Node-CPU-first'* of the *'50:50'* system is as good as the *'Random.'* The overall fairness of *'20:80'* system is way better than that of *'50:50'* system because the number of BE SFCs is 60% more in *'20:80'*.

6.5 Preemptive Prediction-based Service Placement

Both prediction-based and preemption-based service placement suffer from shortcomings/drawbacks. On the prediction-based placement side, due to the restricted length of the lookahead window compared to sfc_{be} duration in TSs, there is no way

to achieve a near zero rejection rate of sfc_{pr} requests without causing severe starvation to the sfc_{be} requests. On the preemption-based placement side, the preemption cost and side effects on sfc_{be} services can be mitigated. As such, taking the best of both worlds, we introduce the preemptive prediction-based placement of time-critical services with VNF sharing. On the one hand, it reduces the rejection rate of sfc_{pr} requests to near zero. On the other hand, it only utilizes preemption in cases where prediction failed to secure the required resources to satisfy sfc_{pr} requests, which inclines the preemption side effects for about 50% of cases compared to the 100% of preemption-based placement scheme.

The proposed scheme is listed in Algorithm 6.3. This scheme is the same as the prediction-based scheme in Section 5.3.2, Algorithm 5.1, but instead of placing the unsatisfied sfc_{pr} request directly into the Rej_{pr} list it utilizes the preemptCPU procedure as in Algorithm 6.2.

6.5.1 Simulations and Results

We use the same set of simulations as in chapter 5 and chapter 6 and compare this scheme against prediction-based and preemption-based placement schemes. The preemption criterion used for the proposed scheme as well as for the preemption-based placement is the best criterion *'SFC-CPU-first.'* For the comparison against the prediction-based scheme, we report the percentage of received SFC requests that are completed, running, and pending for both categories premium and best-effort. Also, we report the average waiting time of best-effort requests. For the comparison against preemption-based scheme, we report preemption related metrics such as average deported sfc_{be} per sfc_{pr} that needed preemption to be satisfied, average deports

Algorithm 6.3: Preemptive Prediction-based Placement (PPSP)

// SimDur: Simulation duration in time slots (TSs), $\omega \in [1,3]$ TSs Lookahead window length, and $\alpha \in [1,3]$

Input : net-Model, SimDur, ω, α
Init. : Rec_{pr}, $Run_{pr|be}$, Pen_{be}, Rej_{pr}, $Com_{pr|be}$
Output: Different queues\lists and collected statistics

for $i \leftarrow 1$ **to** SimDur **do**
  $Rec_{pr} \leftarrow$ received sfc_{pr} requests
  $Pen_{be} \leftarrow$ received sfc_{be} requests
  // Update TTL of running SFCs
  foreach $sfc_{pr|be}$ *in* $Run_{pr|be}$ **do**
    if ttl($sfc_{pr|be}$) $= 0$ **then**
      $Com_{pr|be} \leftarrow sfc_{pr|be}$ // Releases resources utilised by finished sfc
    else decTTL($sfc_{pr|be}$)
  foreach sfc_{pr} *in* Rec_{pr} **do**
    sol $\leftarrow$satisfy(sfc_{pr},net-Model)
    if sol $\neq \emptyset$ **then**
      deploy(sfc_{pr},net-Model)
      $Run_{pr} \leftarrow sfc_{pr}$
    else
      // Check preemptCPU algorithm for details
      sol,$Pen_{be} \leftarrow$preemptCPU(criterion,Run_{be})
      if sol $\neq \emptyset$ **then**
        deploy(sfc_{pr},net-Model)
        $Run_{pr} \leftarrow sfc_{pr}$
      else $Rej_{pr} \leftarrow sfc_{pr}$
  $ReqrdRes_{pr} \leftarrow$ predReqResources(ω)
  // *AvailRes*: free resource in current TS and in next ω TSs
  $AvailRes \leftarrow$ getAvailResources(ω)
  $ExtraRes_{be} \leftarrow 0$
  if $AvailRes > (\alpha * ReqrdRes_{pr})$ **then**
    $ExtraRes_{be} \leftarrow AvailRes - (\alpha * ReqrdRes_{pr})$
  while $(\exists\, sfc_{be}$ *in* $Pen_{be}) \& (ExtraRes_{be} \neq 0)$ **do**
    sol $\leftarrow$satisfy(sfc_{be},net-Model)
    if sol $\neq \emptyset$ **then**
      deploy(sfc_{be},net-Model)
      $Run_{be} \leftarrow sfc_{be}$
      $ExtraRes_{be} \leftarrow ExtraRes_{be}$-usedRes($sfc_{be}$)
    // else sfc stays in Pen_{be}

per sfc_{pr}, and the average preemption cost per sfc_{pr}.

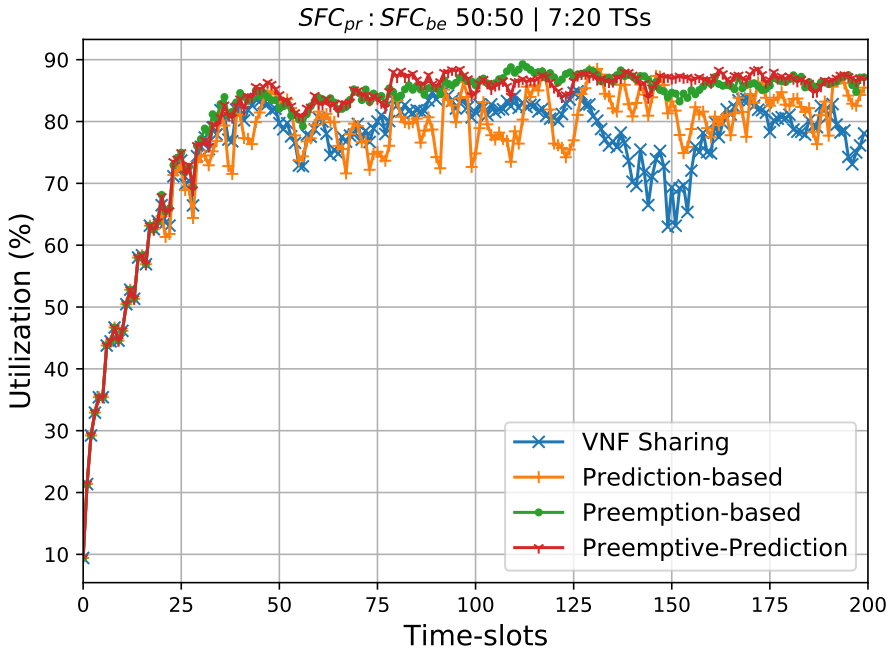


Figure 6.11: Moderate-loaded system Utilization per time-slot

As shown in Figures 6.11 and 6.12, the utilization of the preemptive prediction-based scheme is overall better than that of the prediction-based and as good as that of the preemption-based scheme in both system loads/configurations. Figure 6.13 depicts the status of received SFC requests at the end of the simulation. The moderately-loaded system statistics are in Figures 6.13a and 6.13b; these show that, on the one hand, the preemptive prediction-based placement scheme reduced the premium rejection to zero, even using the smallest *safety margin* $\alpha = 1$. On the other hand, the percentage of pending best-effort requests, with $\alpha = 1$, is almost the same as the best prediction-based placement with $\alpha = 2.88$. The same impact of preemptive prediction-based scheme can be seen in the reported highly-loaded system results in Figures 6.13c and 6.13d. However, it is more impressive as the best prediction-based

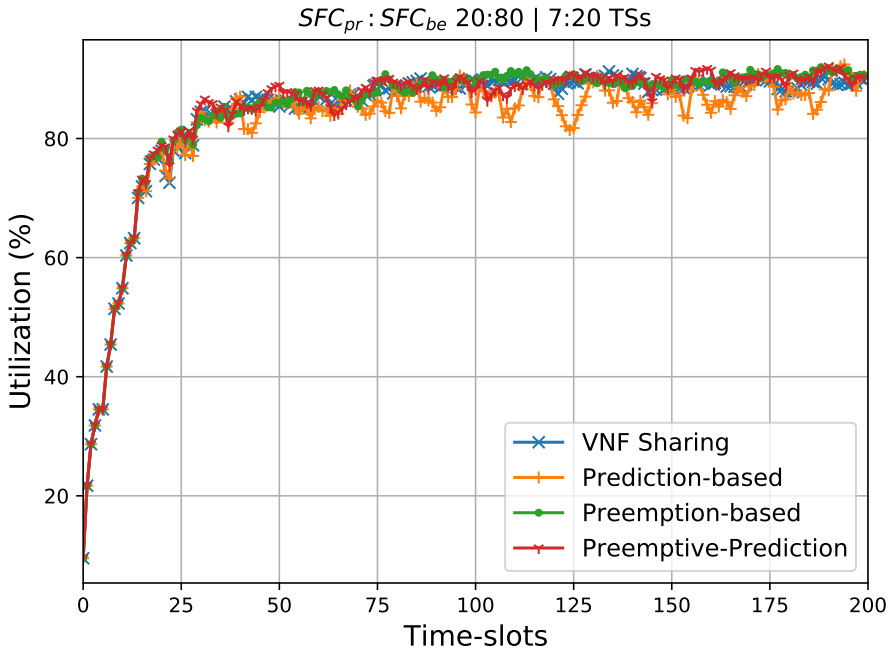


Figure 6.12: Highly-loaded system Utilization per time-slot

placement, with $\alpha = 2.9$, still has around +40% rejected sfc_{pr} requests, yet the preemptive prediction-based scheme is able to totally eliminate premium services rejections with almost no increase to the pending BE SFCs (compare the Pen_{be} of the preemptive prediction-based with $\alpha = 1$ to that of prediction-based with $\alpha = 2.9$). There are two faces to the placement schemes that we are comparing in this section, the impact on the rejection rate of sfc_{pr} requests and on the AWT and pending sfc_{be} requests. Both the AWT and percentage of pending sfc_{be} requests are reported in Figures 6.14 and 6.15, respectively. For reference, we report the AWT and Pen_{be} for VNF sharing-based, preemption-based, prediction-based, and preemptive prediction-based schemes. As can be seen in Figure 6.14, the Pen_{be} of the preemptive prediction-based scheme is better, lower than that of the preemptive scheme ,and almost the same as that of the prediction-based of the highly-loaded system. As

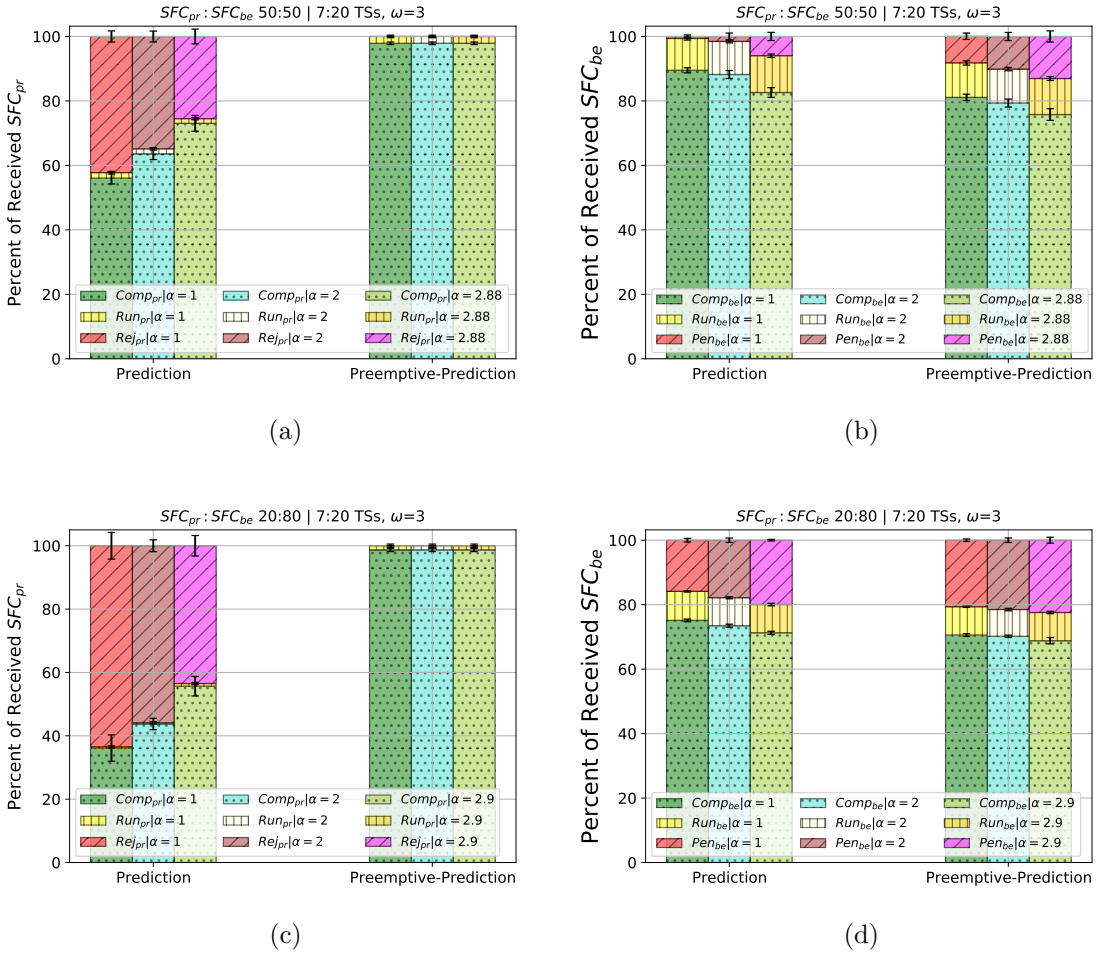


Figure 6.13: Closing premium and best-effort queues and lists ($Comp_{pr}$, Run_{pr}, Rej_{pr}, $Comp_{be}$, Run_{be} and Pen_{be}), for prediction-based compared to preemptive prediction-based placement (reported for different values of the resource *safety margin* α), reported for Moderately-loaded (a)&(b) and Highly-loaded system (c)&(d)

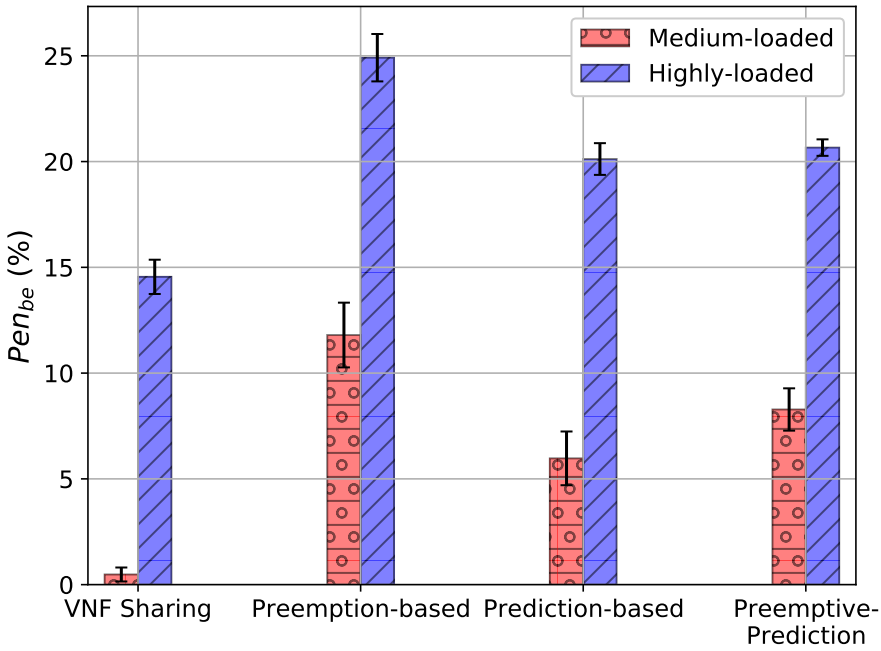


Figure 6.14: Percentage of pending BE requests (Pen_{be})

shown in Figure 6.15, the moderately-loaded system's AWT of sfc_{be} is the same in preemption-based, prediction-based, and preemptive prediction-based schemes. This is not the case with the highly-loaded system because of the higher ratio of sfc_{be} requests,80%, and the compound contribution to the AWT from both prediction's *safety margin* and from preemption's sfc_{be} deports.

If the preemption-based placement achieves zero sfc_{pr} rejections, a valid question would be why use a preemptive prediction-based scheme over a pure preemption-based scheme? To answer this question, we compare the preemption related metrics of both schemes as shown in Figure 6.16. The average number of deported/suspended sfc_{be} to satisfy one sfc_{pr} request of preemptive prediction-based scheme is lower than the preemption-based scheme for both moderately- and highly-loaded systems, see Figure 6.16a. The same applies to the average number of sfc_{be} deports to satisfy one

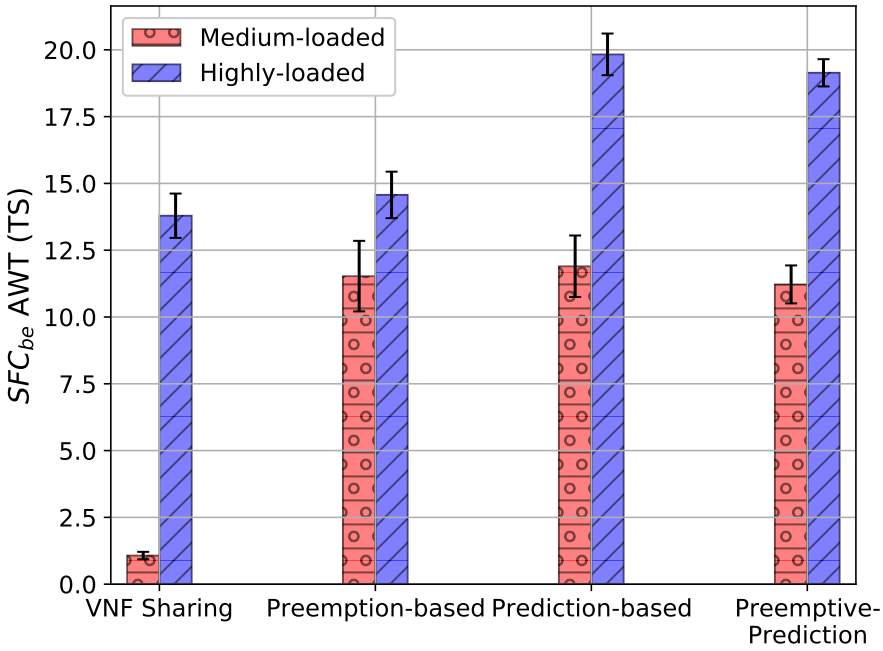
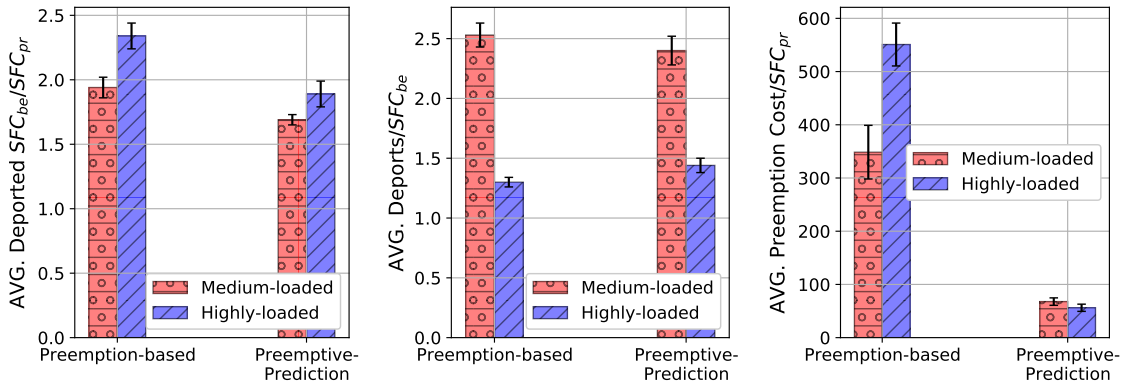


Figure 6.15: Average waiting time (AWT) of sfc_{be} requests



(a) (b) (c)

Figure 6.16: (a) Average number of deported sfc_{be} to satisfy one sfc_{pr} request. (b) Average number of sfc_{be} deports to satisfy one sfc_{pr} request. (c) Average preemption cost to satisfy one sfc_{pr} request.

sfc_{pr} requests (see Figure 6.16b), except for the highly-loaded system, the reason is that having 80% sfc_{be} requests and using the *'SFC-CPU-first'* preemption criterion, deployed BE SFCs with a higher number of CPU cores are more likely to be deported. When it comes to preemption cost (based on Equation 6.11), shown in Figure 6.16c, the preemptive prediction-based scheme is a distant first.

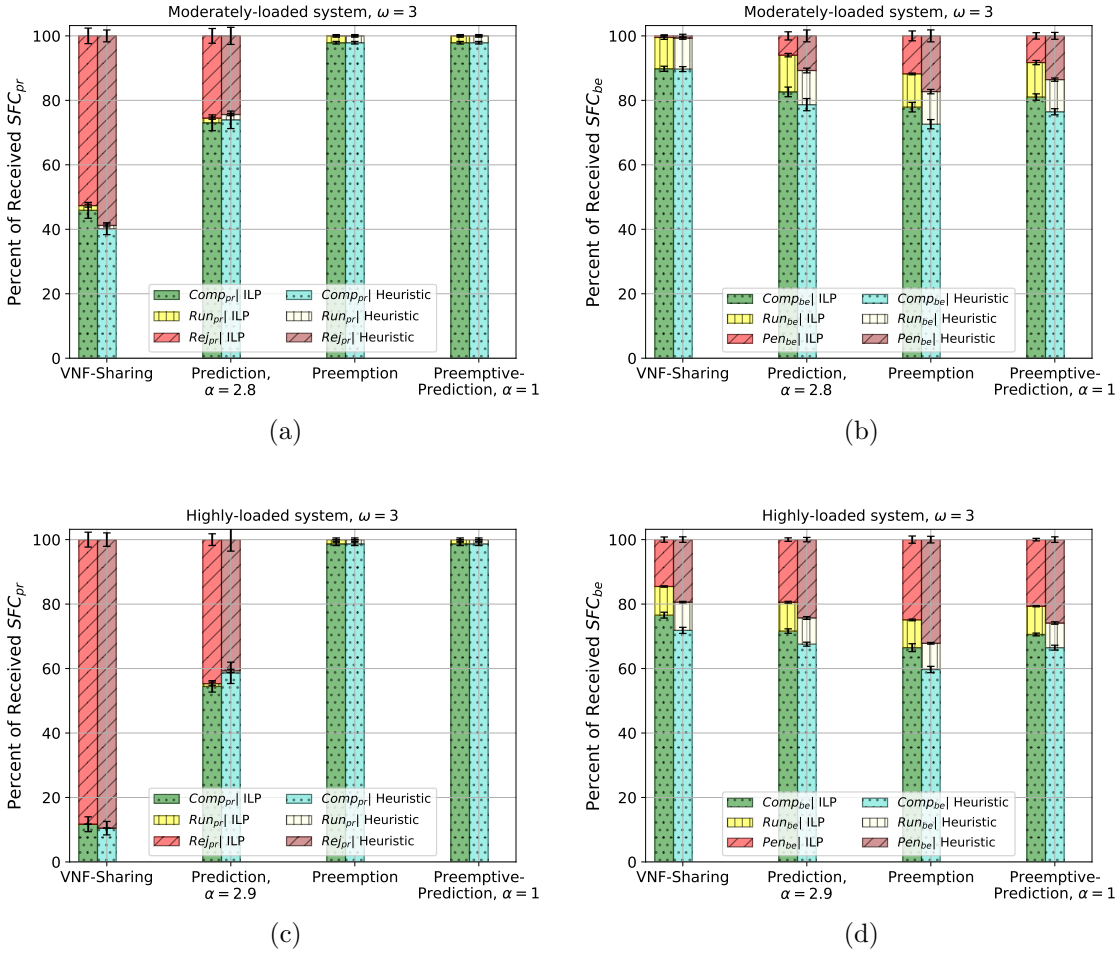
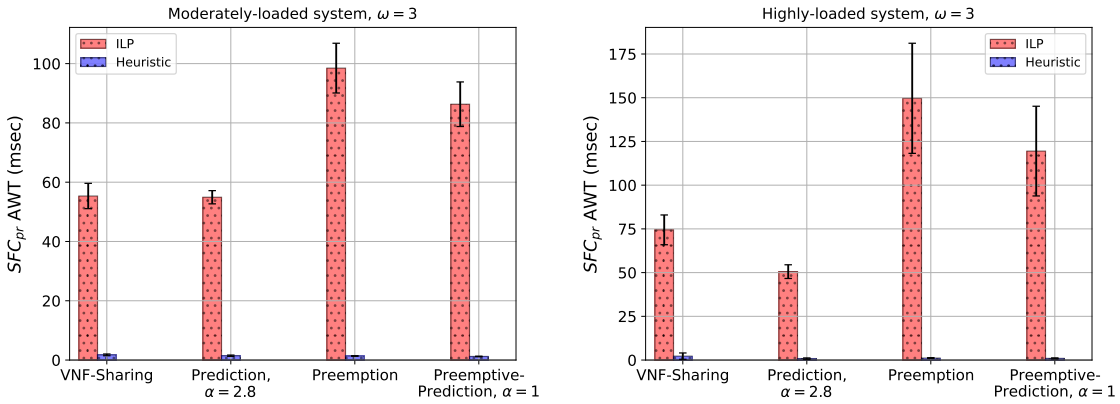


Figure 6.17: Closing premium and best-effort queues and lists ($Comp_{pr}$, Run_{pr}, Rej_{pr}, $Comp_{be}$, Run_{be} and Pen_{be}), for VNF Sharing-based, prediction-based, preemption-based, vs preemptive prediction-based placement schemes, reported for ILP and Heuristic placement algorithms, and for Moderately-loaded (a)&(b) and Highly-loaded system (c)&(d).

As shown in Figure 6.17, the sfc_{pr} statistics are almost the same for ILP and heuristic-based placement schemes for the moderately and highly-loaded systems. On the one hand, we can see that for the highly-loaded system, the ILP-based placement schemes are resulting in $6-11\%$ more completed BE SFC requests $Comp_{be}$, and the heuristic-based placement resulted in $25-33\%$ more pending BE SFC requests Pen_{be}. On the other hand, as shown in Figure 6.18, the heuristic-based placement schemes reduced the AWT of sfc_{pr} requests of the moderately-loaded systems to $31-72$ fold compared to the ILP-based placement schemes, and $37-149$ fold of the highly-loaded systems.



(a) (b)

Figure 6.18: AWT of ILP and Heuristic-based placement schemes for (a) Moderately-loaded and (b) Highly-loaded systems.

6.6 Summary

In this chapter, we propose preemption-based and preemptive prediction-based schemes for immediate placement of time-critical services with VNF sharing. The simulation results of different preemption criteria show that *'SFC-CPU-first'* is the

least 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 possibly-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. In environments where compute resources are scarce and would be preferably used to process actual services/workloads, the *'Random'* criterion works just fine. Finally, we conclude that the preemptive prediction-based scheme is better than both prediction-based and pure preemption-based schemes provided that we have a simple predictor that can produce accurate predictions using a relatively short lookahead window. If such predictor is hard to find except with longer lookahead windows or is not simple, the cost of that predictor should be added to the cost equation 6.11.

We realize that the gratuitous deports of BE SFCs are unavoidable, even when using the best preemption criterion. This is because of VNF sharing and the unknown number of resources released as a result of deporting BE SFCs. Being able to diagnose, in detail, the reason behind failing to satisfy the Pr SFC in the first place would give us some guidance to design a better, less disturbing to BE SFCs, preemption criterion. For future work, we will diagnose the failure of Pr SFC placement and use such diagnosis to relocate single/few VNFs or deport at most a single BE SFC.

Chapter 7

Conclusion and Future Directions

"Reasoning draws a conclusion, but does not make the conclusion certain, unless the mind discovers it by the path of experience"

Roger Bacon

7.1 Summary

In this thesis, we addressed the service placement at the network edge using *VNF sharing* to increase the efficiency of edge resource utilization. This was motivated by the fact that some network functions can serve multiple traffic flows belonging to different services, and the operations dynamics, which sometimes leaves deployed VNFs underutilized. As a result, we proposed service placement schemes that utilized *VNF sharing* and achieved comprehensive improvements to system utilization, service provisioning cost, and significantly reduced premium services rejections.

Chapter 1 gave an overview of the research problem and our contributions to this area of edge resource utilization. Chapter 2 detailed the background and related

work, where we explained the concepts, importance, and types of edge computing. We covered the enabling technologies of edge computing and existing edge platforms. Finally, we reviewed several service provisioning schemes, including service placement and resource allocation, service migration, and replication.

Chapter 3 presented our proposed scheme to improve edge resource utilization, taking into consideration the performance/QoS requirements of service requests. With limited edge resources, system utilization and the number of satisfied service requests are the main metrics used to measure the performance gain of our proposed scheme. The proposed placement schemes achieve such gains by taking advantage of common functions among services, shareable functions, and operations dynamics. The results showed an increase in the number of satisfied services using the same amount of edge resources indicating better resource utilization and reduced service provisioning cost.

In Chapter 4, we broadened our focus to include different service QoS categories/priorities, specifically premium and best-effort priorities. We designed PSVShare, a priority-based service placement with *VNF sharing* scheme. PSVShare handles migration situations arising from sharing function and traffic variation. Simulation results show that PSVShare is consistently outperforming the non-sharing service placement scheme under different loads, and with different queue sizes with varying premium to best-effort ratios.

Chapter 5 expand on the foundation laid in Chapter 4. We introduced PSVS, a prediction-based service placement scheme to reduce the rejection rate of real-time/time-sensitive premium services at the edge. PSVS utilizes the predicted required resources and a *safety margin* to address the difference between lookahead window size and premium services duration and resiliency against prediction errors.

We experimented and evaluated different lookahead window sizes and *safety margin* and concluded the best values to balance the reduction in premium service requests rejection rate and the best-effort service requests average waiting time. We found that more reduction in the rejection rate is attainable; however, the best-effort service requests suffer more starvation. This outcome is desirable in emergencies where time-critical premium services must be immediately satisfied.

Chapter 6 builds on the findings of chapter 5 to immediately satisfy time-critical services. To mitigate the consequences of a failure to satisfy a time-critical service request, we proposed IPTSV for the immediate placement of time-critical services with *VNF sharing*. First, we proposed a pre-emptive service placement that pre-empts CPU by deporting deployed best-efforts services in situations where no resources are available to satisfy the premium services. We experimented and found the best criterion that is the least likely to unnecessarily deport best-effort services even when *VNF sharing* is not used. Second, to soothe the side effects of pre-emption on deployed best-effort services, we combined prediction and pre-emption by taking the best of both worlds and designed a pre-emptive prediction-based placement of time-critical services. The default is to use prediction-based placement, and the pre-emption kicks in when the prediction fails to secure the required resources to satisfy a premium service. The results showed zero rejections of premium services and reduced the side effects pre-emption has on best-efforts services.

7.2 Limitations

Our priority in system modelling is to mimic realistic edge computing scenarios. Although several efforts have been made, including simulation platforms for cloud

computing, NFV, and SDN, these are still simulation environments that may not be able to replicate the exact real-world settings. We could not find MEC or edge traces for realistic service requests and traffic traces. We decided to use synthesised service requests and traffic traces to replicate possible real edge environments. Each use case, IIoT and AR/VR, has unique requirements and constraints. We designed a solution that can be customized to fit each of these use cases.

7.3 Future Directions

This thesis has presented a novel direction in service provisioning using *VNF sharing* at the network edge. Our proposals achieve comprehensive improvements both for end-users and service providers. However, there are still open challenges and potential implementation opportunities that can be explored to enhance some of the proposed schemes further and complete the knowingly abstracted details. A few of these avenues are listed as follows.

7.3.1 Diagnosis of failed service placement

Using *VNF sharing*-based placement has its advantages and some drawbacks. The slightly complicated interrelation between host and guest VNFs has difficulties in situations where the host VNF cannot tolerate the aggregate traffic load. When addressing the immediate placement of premium time-critical services, diagnosing the exact reason the scheme failed to satisfy a specific request would help provide the right solution. Without such diagnoses, the proposed solutions, such as prediction-based scheme with *safety margin* and preemption-based scheme, will have undesirable yet avoidable drawbacks.

If the placement scheme fails to satisfy an SFC request, this mean none of the found placement solutions is feasible. A placement solution is unfeasible if one or more nodes on the solution path does not have the required resources or the end-to-end delay of solution path is greater than the maximum delay of the SFC request. There is nothing we can do for the latter. Taking into account resources availability, we can inspect the solution path nodes whose available resources are less than the required and migrate/relocate one or more of the deployed BE VNFs on that node. Relocating a single VNF should be easier and less disturbing to deployed SFCs compared to the whole SFC deportation.

To evaluate such scheme, we would use metrics like Pr rejection rate, the average number of relocated VNFs to satisfy one Pr SFC, and the AWT of BE SFCs. The comparison of evaluation metrics should be of the proposed scheme against plain VNF sharing-based, prediction-based, and preemption-based placement schemes.

7.3.2 Benchmarking-/profiling-aware placement

As mentioned in Section 7.2, all our proposed schemes are evaluated using simulation based evaluation. With such promising results, the next logical step is to utilize benchmarking, profiling, test-beds, and real SFCs and VNFs. Both profiling and benchmarking will help us ensure that VNFs accurately satisfy the performance/QoS requirements when assigned a specific amount of resources. Profiling and benchmarking will help in determining the required resources (varies by edge node) to satisfy VNF's performance requirements.

We will need a data set that correlates the resources required to fulfill certain performance requirements. This is achievable by using a test-bed with varying hardware

configurations of compute nodes, or by utilizing previous findings in the literature, if exist, about the resources required to address the performance requirements of specific workloads on specific hardware configuration. Moreover, we plane to use readily available open source VNFs of different flavors and requirements, i.e. computationally-intensive VNFs, input/output intensive VNFs, and VNFs of the same type/flavor but with different resources and/or performance requirements. Using the data set, we will train an ML model with the goal of being able to accurately determine the required resources to achieve a certain performance requirements, or the expected achievable performance if a VNF is assigned a certain number of resources.

Using the trained ML model and the test-bed, we will re-evaluate and report the results of all the proposed VNF sharing-based placement schemes in previous chapters, with real VNFs that are deployed in a real compute nodes and processing varying traffic loads. The moral of this proposal is to report the performance of VNFs, guest or host, while sharing the resources with others, and to determine the factors that control the maximum achievable performance with VNF sharing (is it only the unused capacity, the number and load of guest VNFs, hosting node hardware configuration, or a combination of these factors?). The metrics used to evaluate the performance of the proposed placement schemes in previous chapters can be used here.

Bibliography

[1] "KubeEdge, a Kubernetes Native Edge Computing Framework," https://kubernetes.io/blog/2019/03/19/kubeedge-k8s-based-edge-intro/, accessed: 04-03-2022.

[2] "Linux Foundation , Nephio Project," https://nephio.org/, accessed: 04-15-2022.

[3] "Linux Foundation Edge (LF Edge)," https://www.lfedge.org/, accessed: 04-03-2022.

[4] "Linux Foundation Edge (LF Edge), Akraino Edge Stack Project," https://www.lfedge.org/projects/akraino/, accessed: 04-03-2022.

[5] "Linux Foundation Edge (LF Edge), Baetyl Porject," https://baetyl.io/en/, accessed: 04-03-2022.

[6] "Linux Foundation Edge (LF Edge), EdgeX Foundry Porject," https://www.edgexfoundry.org/, accessed: 04-03-2022.

[7] "Linux Foundation Edge (LF Edge), Fledge Porject," https://www.lfedge.org/projects/fledge/, accessed: 04-03-2022.

[8] "Linux Foundation Edge (LF Edge), Home Edge Project," https://wiki.lfedge.org/display/HOME/Home+Edge+Project, accessed: 04-03-2022.

[9] "Linux Foundation Edge (LF Edge), Porject EVE," https://www.lfedge.org/projects/eve/, accessed: 04-03-2022.

[10] "Open Daylight," https://www.opendaylight.org/, accessed: 04-03-2022.

[11] "Open Networking Foundation, Next-Generation SDN," https://www.opennetworking.org/ng-sdn/, accessed: 04-03-2022.

[12] "Open Networking Foundation, SDN Projects," https://www.opennetworking.org/onf-sdn-projects/, accessed: 04-03-2022.

[13] "OpenFog Reference Architecture for fog computing," https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf, accessed: 04-03-2022.

[14] "Openstack Foundation, StarlingX Project," https://www.starlingx.io/, accessed: 04-03-2022.

[15] "Rancher K3s, Certified Kubermetes distro for IoT & Edge Computing," https://k3s.io/, accessed: 04-03-2022.

[16] "Service Function Chaining in Openstack," https://docs.openstack.org/newton/networking-guide/config-sfc.html, accessed: 04-03-2022.

[17] "State of The Edge Report," https://project.linuxfoundation.org/hubfs/LF%20Edge/StateoftheEdgeReport_2021.pdf, accessed: 04-03-2022.

[18] "Virtual Kubelet," https://virtual-kubelet.io/docs/architecture/, accessed: 04-03-2022.

[19] M. Abu-Lebdeh, D. Naboulsi, R. Glitho, and C. W. Tchouati, "On the placement of vnf managers in large-scale and distributed nfv systems," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 875–889, Dec 2017.

[20] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *IEEE 4th International Conference on Cloud Networking (CloudNet)*, 2015, pp. 171–177.

[21] I. Afolabi, M. Bagaa, T. Taleb, and H. Flinck, "End-to-end network slicing enabled through network function virtualization," in *IEEE Conference on Standards for Communications and Networking (CSCN)*, Sep. 2017, pp. 30–35.

[22] S. Agarwal, F. Malandrino, C. F. Chiasserini, and S. De, "Vnf placement and resource allocation for the support of vertical services in 5g networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 433–446, 2019.

[23] A. Alleg, T. Ahmed, M. Mosbah, R. Riggio, and R. Boutaba, "Delay-aware vnf placement and chaining based on a flexible resource allocation approach," in *13th International Conference on Network and Service Management (CNSM)*, Nov 2017, pp. 1–7.

[24] Anthos For Telecom, https://cloud.google.com/blog/topics/anthos/anthos-for-telecom-puts-google-cloud-partners-apps-at-the-edge, accessed: 20-10-2021.

[25] AT&T and Azure, https://www.business.att.com/learn/top-voices/at-t-integrates-5g-with-microsoft-azure-to-enable-next-generatio.html, accessed: 20-10-2021.

[26] 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.

[27] M. T. Beck, J. F. Botero, and K. Samelin, "Resilient allocation of service function chains," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 128–133.

[28] P. Bellavista, A. Corradi, A. Edmonds, L. Foschini, A. Zanni, and T. Bohnert, "Elastic provisioning of stateful telco services in mobile cloud networking," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[29] F. Ben Jemaa, G. Pujolle, and M. Pariente, "Qos-aware vnf placement optimization in edge-central carrier cloud architecture," in *IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–7.

[30] I. Benkacem, T. Taleb, M. Bagaa, and H. Flinck, "Optimal vnfs placement in cdn slicing over multi-cloud environment," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 616–627, March 2018.

[31] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA:

Association for Computing Machinery, 2012, p. 13–16. [Online]. Available: https://doi.org/10.1145/2342509.2342513

[32] F. Brockners, S. Bhandari, S. Dara, C. Pignataro, J. Leddy, S. Youell, D. Mozes, T. Mizrahi, A. Augado, and D. Lopez, "Proof of transit, draft-ietf-sfc-proof-of-transit-02," IETF, Internet-Draft, Tech. Rep., 2019.

[33] G. Brown, S. Analyst, and H. Reading, "White paper: Service chaining in carrier networks," 2015.

[34] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things," in *IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2016, pp. 1–6.

[35] J. Cao, Y. Zhang, W. An, X. Chen, J. Sun, and Y. Han, "Vnf-fg design and vnf placement for 5g mobile networks," *Science China Information Sciences*, vol. 60, no. 4, p. 040302, Mar 2017. [Online]. Available: https://doi.org/10.1007/s11432-016-9031-x

[36] P. Caserman, M. Martinussen, and S. Göbel, "Effects of end-to-end latency on user experience and performance in immersive virtual reality applications," in *Joint International Conference on Entertainment Computing and Serious Games*. Springer, 2019, pp. 57–69.

[37] J. Chen, X. Zheng, and C. Rong, "Survey on software-defined networking," in *Second International Conference on Cloud Computing and Big Data in Asia*. Springer, 2015, pp. 115–124.

[38] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec 2016.

[39] 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 *IEEE 7th International Conference on Network Softwarization (NetSoft)*. IEEE, 2021, pp. 348–356.

[40] Dave Greenfield, "The Pains and Problems of NFV," https://www.catonetworks.com/blog/the-pains-and-problems-of-nfv/, accessed: 04-12-2022.

[41] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, *Microservices: Yesterday, Today, and Tomorrow*. Cham: Springer International Publishing, 2017, pp. 195–216. [Online]. Available: https://doi.org/10.1007/978-3-319-67425-4_12

[42] Y. Elkhatib, B. Porter, H. B. Ribeiro, M. F. Zhani, J. Qadir, and E. Rivière, "On using micro-clouds to deliver the fog," *IEEE Internet Computing*, vol. 21, no. 2, pp. 8–15, Mar 2017.

[43] 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

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

[45] ETSI GS MEC 003 V1.1.1 (2016-03), "Mobile Edge Computing (MEC) framework and reference architecture," https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf, accessed: 04-03-2022.

[46] ETSI, GSNFV, "Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework," https://www.etsi.org/deliver/etsi_gs/NFV-EVE/001_099/005/01.01.01_60/gs_NFV-EVE005v010101p.pdf, accessed: 05-06-2022.

[47] ETSI ISG NFV, "Network Functions Virtualisation (NFV) architectural framework," https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf, accessed: 04-03-2022.

[48] I. Farris, T. Taleb, M. Bagaa, and H. Flick, "Optimizing service replication for mobile delay-sensitive applications in 5g edge network," in *IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[49] I. Farris, T. Taleb, H. Flinck, and A. Iera, "Providing ultra-short latency to user-centric 5g applications at the mobile network edge," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 4, p. e3169, 2018.

[50] I. Farris, T. Taleb, A. Iera, and H. Flinck, "Lightweight service replication for ultra-short latency applications in mobile edge networks," in *IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[51] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[52] Fergus Wills, "Why Problems With Service Chaining Are Stalling NFV," https://www.sdxcentral.com/articles/contributed/problems-with-service-chaining-stalling-nfv/2018/08/, accessed: 04-12-2022.

[53] A. Ghosh, A. Maeder, M. Baker, and D. Chandramouli, "5g evolution: A view on 5g cellular technology beyond 3gpp release 15," *IEEE access*, vol. 7, pp. 127 639–127 651, 2019.

[54] 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.

[55] K. GREENE, "TR10: software-defined networking. MIT Technology Review (March/April)," http://www2.technologyreview.com/news/412194/tr10-software-defined-networking/, accessed: 04-03-2022.

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

[57] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, and M. Satyanarayanan, "Adaptive vm handoff across cloudlets," *Technical Report CMU-CS-15-113*, 2015.

[58] 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.

[59] J. Halpern, C. Pignataro *et al.*, "Service function chaining (sfc) architecture," in *RFC 7665*, 2015, pp. 1–32.

[60] J. L. Hardcastle, "sdxCentral: Juniper CTO Says Edge Is Key to Monetizing 5G, and telcos have 'beach-front property'," https://www.sdxcentral.com/articles/news/juniper-cto-says-edge-is-key-to-monetizing-5g-and-telcos-have-beach-front-property/2019/02/, accessed: 04-03-2022.

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

[62] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[63] N. Kazemifard and V. Shah-Mansouri, "Minimum delay function placement and resource allocation for open ran (o-ran) 5g networks," *Computer Networks*, vol. 188, p. 107809, 2021.

[64] M. Khan, R. S. Alhumaima, and H. S. Al-Raweshidy, "Qos-aware dynamic rrh allocation in a self-optimized cloud radio access network with rrh proximity constraint," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 730–744, 2017.

[65] S. Khebbache, M. Hadji, and D. Zeghlache, "Virtualized network functions chaining and routing algorithms," *Computer Networks*, vol. 114, pp. 95 – 110, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128617300087

[66] H.-G. Kim, D.-Y. Lee, S.-Y. Jeong, H. Choi, J.-H. Yoo, and J. W.-K. Hong, "Machine learning-based method for prediction of virtual network function resource demands," in *IEEE 5th International Conference on network softwarization (NetSoft)*, 2019, pp. 405–413.

[67] T. Kim, S. Kim, K. Lee, and S. Park, "A qos assured network service chaining algorithm in network function virtualization architecture," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp. 1221–1224.

[68] S. Kumar, M. Tufail, S. Majee, C. Captari, and S. Homma, "Service Function Chaining Use Cases In Data Centers," Internet Engineering Task Force, Internet-Draft, Feb. 2017, work in Progress.

[69] 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.

[70] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, 2018.

[71] S. Lange, H.-G. Kim, S.-Y. Jeong, H. Choi, J.-H. Yoo, and J. W.-K. Hong, "Machine learning-based prediction of vnf deployment decisions in dynamic networks," in *20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2019, pp. 1–6.

[72] Larry Peterson, "Overtaken By Events: Whatever Came of the NFV Initiative?" https://systemsapproach.substack.com/p/overtaken-by-events?s=r, accessed: 04-12-2022.

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

[74] LF Edge, "Sharpening the Edge: overview of the lf edge taxonomy and framework," https://www.lfedge.org/wp-content/uploads/2020/07/LFedge_Whitepaper.pdf, accessed: 04-13-2022.

[75] LFN ONAP, "Open Network Automation Platform (ONAP) architecture overview white paper," https://www.onap.org/wp-content/uploads/sites/20/2019/07/ONAP_CaseSolution_Architecture_062519.pdf, accessed: 04-03-2022.

[76] 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.

[77] 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.

[78] 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.

[79] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 543–553, 2017.

[80] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 140–147, February 2018.

[81] 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.

[82] E. Marín-Tordera, X. Masip-Bruin, J. García-Almiñana, A. Jukan, G.-J. Ren, and J. Zhu, "Do we all really know what a fog node is? current trends towards an open definition," *Computer Communications*, vol. 109, pp. 117–130, 2017.

[83] 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: 03-30-2022.

[84] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

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

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

[87] 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.

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

[89] Nati Shalom, "70% Of NFV And Digital Transformation Projects Fail: Cloudify's 5 Key Lessons To Success," https://cloudify.co/blog/70-of-nfv-and-digital-transformation-projects-fail-cloudifys-5-key-lessons-to-success/, accessed: 04-12-2022.

[90] V. Nguyen, A. Brunstrom, K. Grinnemo, and J. Taheri, "Sdn/nfv-based mobile packet core network architectures: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1567–1602, thirdquarter 2017.

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

[92] Nicira Inc., "NICIRA NETWORKS: DISRUPTIVE NETWORK VIRTUALIZATION 2012," https://web.stanford.edu/class/ee204/2012/Nicira%20Networks.pdf, accessed: 04-13-2022.

[93] O-RAN Alliance, "O-RAN Architecture Description v03.00," O-RAN Alliance, Tech. Rep., 2021.

[94] D. B. Oljira, K. Grinnemo, J. Taheri, and A. Brunstrom, "A model for qos-aware vnf placement and provisioning," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2017, pp. 1–7.

[95] Openstack, "Cloud Edge Computing: beyond the data center," https://www.openstack.org/assets/edge/OpenStack-EdgeWhitepaper-v3-online.pdf, accessed: 03-30-2022.

[96] OSM End User Advisory Group, "OSM White Paper osm scope, functionality, operation and integration guidelines," https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Scope_and_Functionality.pdf, accessed: 04-03-2022.

[97] S. Pandey, J. W.-K. Hong, and J.-H. Yoo, "Gru and edgeq-learning based traffic prediction and scaling of sfc," in *IEEE 7th International Conference on Network Softwarization (NetSoft)*, 2021, pp. 124–132.

[98] S. Park, H.-G. Kim, J. Hong, S. Lange, J.-H. Yoo, and J. W.-K. Hong, "Machine learning-based optimal vnf deployment," in *21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2020, pp. 67–72.

[99] L. Peterson, T. Anderson, S. Katti, N. McKeown, G. Parulkar, J. Rexford, M. Satyanarayanan, O. Sunay, and A. Vahdat, "Democratizing the network

edge," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 2, pp. 31–36, 2019.

[100] P. Quinn, U. Elzur, and C. Pignataro, "Network service header (nsh)," in *RFC 8300*. RFC Editor, 2018.

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

[102] M. Satyanarayanan, "Edge computing for situational awareness," in *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, June 2017, pp. 1–6.

[103] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[104] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

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

[106] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted nfv service chain deployment based on affiliation-aware vnf placement," in *IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–6.

[107] H. Takagi and L. Kleinrock, "Throughput analysis for persistent csma systems," *IEEE transactions on communications*, vol. 33, no. 7, pp. 627–638, 1985.

[108] 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.

[109] M. Torres Vega, C. Liaskos, S. Abadal, E. Papapetrou, A. Jain, B. Mouhouche, G. Kalem, S. Ergüt, M. Mach, T. Sabol *et al.*, "Immersive interconnected virtual and augmented reality: a 5g and iot perspective," *Journal of Network and Systems Management*, vol. 28, no. 4, pp. 796–826, 2020.

[110] T. X. Tran, M.-P. Hosseini, and D. Pompili, "Mobile edge computing: Recent efforts and five key research directions," *IEEE COMSOC MMTC Commun.-Frontiers*, 2017.

[111] 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.

[112] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[113] Verizon and Wavelength, https://www.verizon.com/about/news/verizon-private-mobile-edge-computing-enterprise-aws-outposts, accessed: 20-10-2021.

[114] M. Villari, A. Celesti, G. Tricomi, A. Galletta, and M. Fazio, "Deployment orchestration of microservices with geographical constraints for edge computing," in *IEEE Symposium on Computers and Communications (ISCC)*, July 2017, pp. 633–638.

[115] VMware Telco Cloud Blog, "Innovation at the Telco Edge," https://blogs.vmware.com/telco/innovation-at-the-telco-edge/, accessed: 04-03-2022.

[116] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, June 2019.

[117] Z. Ye, X. Cao, J. Wang, H. Yu, and C. Qiao, "Joint topology design and mapping of service function chains for efficient, scalable, and reliable network functions virtualization," *IEEE Network*, vol. 30, no. 3, pp. 81–87, 2016.

[118] 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.

[119] 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 *IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 731–741.

[120] H. Zhu and C. Huang, "Cost-efficient vnf placement strategy for iot networks with availability assurance," in *IEEE 86th Vehicular Technology Conference (VTC-Fall)*, Sep. 2017, pp. 1–5.