OPTIMIZATION AND ORCHESTRATION IN MULTI-TIER EDGE COMPUTING

by

Ahmet Cihat Baktır

B.S., Computer Science & Engineering, Sabancı University, 2012

M.A., Management Information Systems, Boğaziçi University, 2014

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2021

ACKNOWLEDGEMENTS

I would like to express my gratitude and endless thank to my thesis supervisor Cem Ersoy and co-advisor Atay Özgövde for supporting and guiding me at every stage of this journey. I feel very happy to have the opportunity of benefiting from their vast knowledge in all areas and friendship.

I would like to especially thank Gülfem Işıklar Alptekin and Tuna Tuğcu for their invaluable and constructive contributions throughout the years, which remarkably made my thesis to reach this quality. I would also like to thank to my core jury members Sema Oktuğ and Can Özturan for their time, participation and suggestions.

I would also like to express my thanks to all my colleagues and NETLAB members for always having their support, understanding and friendship. I especially thank Can Tunca and Orhan Ermiş for guiding me and sharing all important moments. I would also like to thank Barış Yamansavasçılar, Meriç Turan, Alper Alimoğlu, Turgay Pamuklu, Çağatay Sönmez, Serhan Daniş, Sinan Işık, Niaz Chalabanlioo, Alp Kındıroğlu, Yiğit Yıldırım and Uras Mutlu for their supports.

I would like to express my sincere thanks to all CMPE members, especially Suzan Üsküdarlı, Mehmet Ufuk Çağlayan, Albert Ali Salah and Lale Akarun.

I would like to thank my family for their support and love throughout my life.

I would like to thank my loving wife and daughter with all my heart. Without them, I couldn't have achieved any of this.

This thesis has been supported by the Turkish Directorate of Strategy and Budget under the TAM Project number 2007K12-873.

ABSTRACT

OPTIMIZATION AND ORCHESTRATION IN MULTI-TIER EDGE COMPUTING

In addition to the efforts in the next-generation cellular networks and traditional network services, the demand for a novel set of services leveraged through smart devices and artificial intelligence (AI) techniques increases tremendously. Pervasive healthcare, online gaming, augmented reality, smart city and many other service types with various performance and functional requirements are supplied with data generated by end-user devices. In this highly dynamic environment, the legacy network infrastructure and operations remain incapable of satisfying the expectations of the users and requirements of the services, especially those demanding real-time interaction with ultra-low latency. Therefore, this thesis focuses on the task offloading operations in a multi-tier edge environment and network slicing optimization problems to enable service-oriented behavior and address the demands of both operators and end-users. In this direction, an extensive literature review is carried out, the requirements are determined, and we provide a formal optimization model for each problem definition. In order to address the scalability issues and finding good quality solutions in a short time, heuristic solutions are proposed. Besides efforts in optimization purposes, two different solution proposals using programmable network paradigms are provided as short-term and long-term for implementing the service-centric behavior. The short-term solution based on Software-defined Networking (SDN) is further evaluated by implementing a fall-risk assessment service with real sensory data. The proposed solutions are novel and provide comprehensive guidance for operators and service providers on implementing a service-centric behavior and optimizing the operations in multi-tier edge systems.

ÖZET

ÇOK KATMANLI UÇ HESAPLAMADA ENİYİLEME VE DÜZENLEME

Yeni nesil hücresel ağlar ve geleneksel ağ servislerine yönelik girişimlerin yanı sıra, akıllı cihazlar ve yapay zekâ teknikleri ile geliştirilen özgün servislere yönelik talepte çok büyük bir artış olmaktadır. Çeşitli başarım ve fonksiyonel gereksinimleri talep eden yaygın sağlık hizmeti, çevrimiçi oyunlar, artırılmış gerçeklik, akıllı şehir ve birçok başka servis tipleri son kullanıcı cihazları tarafından üretilen veriler aracılığı ile beslenmektedir. Böylesine dinamik bir ortamda, eski ağ altyapısı ve operasyonları kullanıcıların beklentilerini ve servislerin gereksinimlerini, özellikle gerçek zamanlı etkileşim talep edenlerin, karşılamakta yetersiz kalmaktadır. Dolayısı ile, bu tezde, servis merkezli yapıyı olanaklı kılmak, operatörlerin ve son kullanıcıların beklentilerini karşılamak amacıyla çok katmanlı uç hesaplama sistemlerinde görev atanması ve ağ dilimleme problemlerinin eniyilenmesine odaklanılmıştır. Bu doğrultuda oldukça geniş bir literatür taraması yapılmış, gereksinimler belirlenmiştir ve her bir problem tanımı için bir eniyileme modeli geliştirilmiştir. Ölçeklenebilirlik sorununa değinmek ve problemlere kısa süre içerisinde kaliteli bir çözüm bulabilmek için sezgisel yöntem önerileri de sunulmuştur. Bütün bunların yanı sıra, servis merkezli yapının gerçekleştirilmesi için programlanabilir ağ paradigmalarını kullanarak kısa ve uzun vadeli olmak üzere iki farklı çözüm önerisi sunulmuştur. Yazılım-tanımlı ağ yapısı üzerine kurulan kısa vadeli çözüm önerisi gerçek veri ile desteklenmiş bir düşme riski analizi servisi ile ayrıca değerlendirilmiştir. Önerilen çözüm önerileri özgün olup, operatörlere ve servis sağlayıcılara çok katmanlı uç hesaplama sistemlerinde servis merkezli davranışın gerçekleştirilmesi ve operasyonların eniyilenmesi konusunda kapsamlı bir yol gösterici olmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a_{ij}	Link between nodes i, j
b_s	Capacity of server s
c_n	Capacity of node n
E	Set of links
$E(T^q_{node})$	Network delay on the shortest path
$E(T^q_{link})$	Overall delay contributed by the links on the shortest path
$E(T_{qs})$	Service execution delay for service type q by server s
E_{us}	Set of links on the shortest path between user u and server s
G	Network topology
l^{req}_q	Network load of the request for service type q
l^{res}_q	Network load of the response of service type q
m_q	Computational load of service type q
N	Set of nodes
N_{us}	Set of nodes on the shortest path between user u and server s
p_{uqs}	Penalty of a task assignment due to SLA violation
Q	Set of service types
r_{uq}	Average number of tasks generated by user u for service q
S	Set of servers
T	Step size of Lagrangian heuristic
U	Set of users
v_{qs}	Service deployment status
w_q	Delay violation for service type q
y	Maximum number of service instances on an edge server
Z	Set of identical fragments of a physical resource
Z_{uqp}	Assignment of requests to a server location
Z_{Lag}	Objective function value of the Lagrangian relaxed problem
α_q	Maximum allowed delay for service type q

δ_q	Minimum required ratio of handled requests for service q
Δ_p	Lagrangian multiplier
η_{uqp}	Lagrangian multiplier
γ_{qs}	Slicing decision for computational resources
γ_{qsz}	Allocation of computational resource fragments
Γ_{qsz}	Substitution variable for linearization
π	Subgradient algorithm parameter
ϕ	Maximum allowed utilization for computational resources
ϕ_{qij}	Slicing decision for a network link
ϕ_{qijz}	Allocation of network link fragments
Φ_{qijz}	Substitution variable for linearization
τ_{uqs}	End-to-end delay
θ_{uqs}	Task assignment decision variable
Θ_{uqs}	Substitution variable for linearization
Υ_s	Utilization of server s
Υ_n	Utilization of node n
φ	Maximum allowed utilization for networking resources
ϑ_{uqs}	Substitution variable for linearization
ζ_{uqp}	Lagrangian multiplier

LIST OF ACRONYMS/ABBREVIATIONS

AUC	Area Under the Curve
CCN	Content-Centric Networking
CCN	Content Delivery Networks
DSCP	Differentiated Services Code Point
DNS	Domain Name System
ECG	Electrocardiography
ECN	Explicit Congestion Notification
eMBB	Enhanced Mobile Broadband
ETSI	European Telecommunications Standards Institute
FIB	Forwarding Information Base
ICN	Information-Centric Networking
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ILP	Integer Linear Programming
IoT	Internet of Things
ITU	International Telecommunication Union
LAN	Local Area Network
LB	Lower Bound
LP	Linear Programming
LR	Lagrangian Relaxation
M2M	Machine-to-Machine
MAN	Metropolitan Area Network
MCC	Mobile Cloud Computing
MEC	Multi-access Edge Computing, Mobile Edge Computing
MI	Million Instruction
MILP	Mixed Integer Linear Programming
MINLP	Mixed Integer Non-Linear Programming
MIP	Mixed Integer Programming

MIPS	Million Instruction Per Second
MTU	Maximum Transmission Unit
mMTC	Massive Machine Type Communication
NESECS	NEtwork Slicing for Edge Computing Services
NDN	Named Data Networking
NDO	Named Data Object
NGMN	Next Generation Mobile Networks
NFV	Network Function Virtualization
ONF	Open Networking Foundation
OXM	OpenFlow Extensible Match
P4	Programming Protocol-Independent Packet Processors
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RF	Random Forest
ROC	Receiver Operating Characteristic
SCN	Service-Centric Networking
SDN	Software-Defined Networking
SFC	Service Function Chaining
SLA	Service Level Agreement
SVM	Support Vector Machine
ToS	Type of Service
UB	Upper Bound
uRLLC	Ultra Reliable and Low-latency Communication
VM	Virtual Machine
VNF	Virtualized Network Function
WAN	Wide Area Network
WLAN	Wireless Local Area Network

1. INTRODUCTION

The ubiquitous computing domain has a long-standing evolution, and Mark Weiser's vision of "computation being integrated into the fabric of our daily lives" is advancing to become a reality [1]. The recent progress is that the smartphone itself has become the primary computational equipment for an end-user. The new edge devices are not just the traditional desktops that are compact systems; they have a lot more. A broad set of multimodal sensors and audio-visual data with different properties and distributions change the scene for the new set of applications.

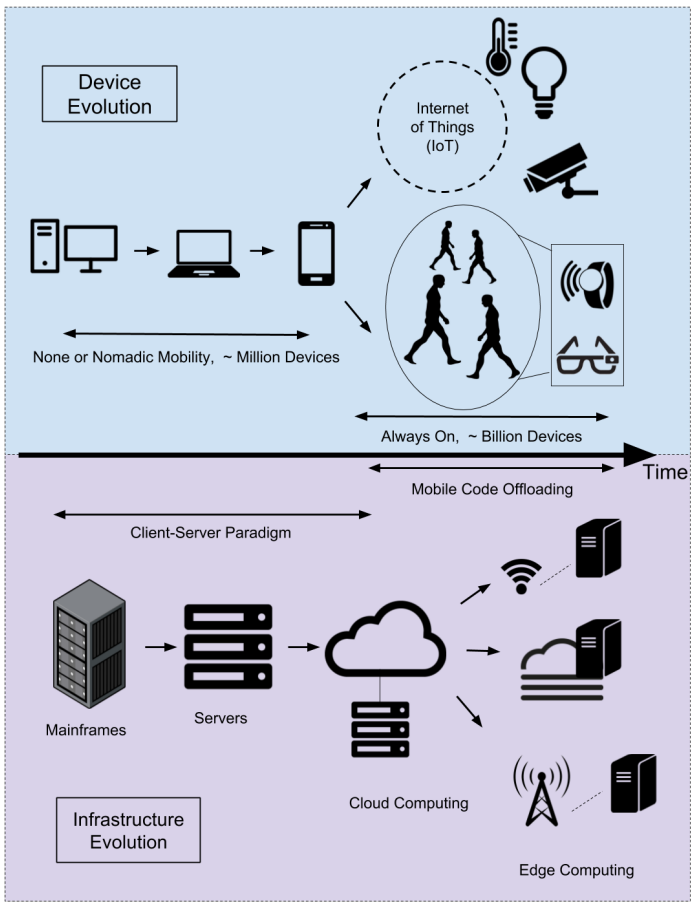

Figure 1.1. Evolution of the end-user devices and servers.

Either coexist with smartphones or in a standalone manner, a wide range of wearable appliances also contributes to the ubiquitous computing vision. A new edge device such as smart glass, smartwatch and smart clothes is becoming commercially available

in the market every day. The evolution of personal devices as the computational resources over the last years is illustrated in Figure 1.1. These devices continuously log data, implement novel services and generate intermittent data traffic. Still, another category falls under the concepts of Internet of Things (IoT) and Machine-to-Machine (M2M) Communications, where machines themselves communicate with each other and participate in different services. IEEE (Institute of Electrical and Electronics Engineers) [2], AllSeen Alliance [3], Thread Group [4], Open Interconnect Consortium [5], and many others have a large set of standards for IoT [6]. Accompanying the ongoing research activities, great effort is put into developing real applications with the help of available standards bodies and specifications.

When we gather all these developments and improvements, we see a pervasive computing environment with a network of devices generating data and offering a large spectrum of novel services. Pervasive healthcare services such as fall-risk assessment, infotainment applications, vehicular communication, augmented reality and many other services can be given concrete examples. However, one problem is implementing services that are integrated with complex machine learning techniques to realize the envisioned use cases via computationally restricted mobile devices. A simple alternative is to improve the computation power of these edge nodes by benefiting from the centralized cloud computing resources. However, even with "unlimited" resource capacities, cloud computing infrastructures cannot solve all problems to the inherent delay constraints of the Wide Area Network (WAN) access [7]. Since most of these services have stringent requirements such as real-time interaction with high Quality of Service (QoS) expectations, this obstacle cannot be underestimated. Considering these limitations in the public Internet, the alternative technical solution that overcomes the problem of indefinite latency is bringing the cloud resources in the proximity of these devices. Recently, we notice a related trend in ubiquitous computing called edge computing, where the computational and storage resources are deployed nearer to the end-user. Eliminating the WAN latency creates the opportunity to meet the requirements of real-time interaction imposed by the latency-intolerant services. Besides, edge computing eases the congestion within the core network and datacenters by keeping the data at

the edge where it is created. It also helps to enable efficient traffic management practices with less effort. It should be noted that expanding the computational capacities of end-user devices through edge computing is not a replacement for the cloud; on the contrary, they are complementary paradigms that need to be in cooperation.

The development of edge computing leads to the proliferation and heterogeneity of the service types provided to the end-users. These services, which are not feasible to be realized so far, are hosted on the distributed edge servers, and the amount of data traffic generated at the edge increases excessively. In order to maintain high service quality and desired level performance, replications of a service type should be provided by the geographically distributed servers.

With the improvements in the computation methodologies and emergence of services enhancing the quality of our daily lives, the focus shifts to the service itself rather than its location. The current IP-based operations remain incapable of orchestrating the interactions between clients and servers efficiently. This problem has higher priority in an edge computing environment, as the service routines may be deployed over many servers. These service instances may also partially reside on local servers and the cloud. A service-centric design is required to resolve all the complexities involved.

When all these cases and requirements are assembled, it is seen that allocating the responsibility of the complex operations such as service discovery to the end-user devices significantly deteriorates the performance and violates the QoS requirements in this dynamic environment. The traditional location-oriented network needs to be remodeled as a service-centric architecture to hide all complex operations from the end-users and implement them in the network. However, leading the focus on "what" instead of "where" is not a smooth procedure for the network operations. Relevantly, there is an ongoing trend for the Information-Centric Networking (ICN), Content-Centric Networking (CCN) and Service-Centric Networking (SCN), which transform the process of the traditional network by shifting the concentration from hosts to the contents, information or services [8]. The "Future Internet" is envisioned as service-

centric [9] and within this context, a service can be requested without any knowledge about server location [10].

However, it is not a straightforward attempt to deploy computational resources at various levels of the network and establish service-oriented operations while letting them be accessible ubiquitously. The envisioned environment and functionality provided by the network and computation infrastructure should be compatible with the TCP/IP protocol stack. Simultaneously, besides its implementation, various optimization problems should be considered, especially in this environment where real-time interaction and QoS have the utmost importance.

1.1. Thesis Contributions

In this thesis, we focus on optimizing the operations in a multi-tier service-oriented structure to address the expectations of both end-users and operators. Based on our previous work [11], an SLA (Service Level Agreement)-aware optimal resource allocation scheme is introduced, in which the main problem components are as follows:

- Undifferentiated set of services where each service type has the same priority
- Fairness among the services with a minimum satisfaction ratio imposed by SLAs.

. For addressing each problem and use-case scenario, two different optimization models for resource allocation and task assignment are designed and implemented: (i) a realistic mixed-integer nonlinear programming (MINLP) model, and (ii) a mixed-integer linear programming (MILP) model obtained by linear approximation of the MINLP model. While the first model adopts a nonlinear queuing model for addressing the latency requirements of the services, the second model utilizes piecewise linear approximation to transform it into an MILP model for resolving the time and space complexity issues. Additionally, a heuristic implementation based on the nearest-fit algorithm is also presented as a quick remedy for the problems where the proposed optimization models remain incapable of obtaining a feasible solution within an acceptable time

limit. Besides, we addressed multi-tier computing system design and proposed an MILP model, a Lagrangian-relaxation based heuristic and a greedy heuristic solution to optimize the server placement and service deployment decisions.

Even though the multi-tier architecture has the inherent capability to enable the service-oriented structure with high performance, it is not solely enough to place the servers at specific locations, deploy the service instances over them, and allocate the available resources optimally. Due to the dynamic nature of the edge environment and diversified characteristics of the service types, it becomes challenging to guarantee a certain level of QoS for particular service types. Therefore, this thesis proposes an optimal network slicing methodology for service-oriented edge access. The proposed Mixed Integer Programming (MIP) model aims to distribute the replications of the services optimally while virtualizing the networking and computational resources for allocating specific capacities for fully isolated slice instances configured for each service type. Additionally, a heuristic algorithm NESECS (NEtwork Slicing for Edge Computing Services) is introduced to find high-quality solutions quickly for cases where the formal optimization tools fail to find feasible solutions.

In addition to the optimization aspect, we also focus on implementing service-oriented operations within the network. With the smooth integration of the edge and cloud servers, the multi-tier computing system handles the offloaded tasks in a service-centric manner. By using the pioneers of the programmable network concept, namely Software-Defined Networking (SDN) and P4 (Programming Protocol-Independent Packet Processors), two different solution approaches are proposed to implement such behavior based on our previous works [11–13].

The original contributions of this thesis are detailed as follows:

- Two problem definitions for SLA-aware resource allocation within a service-centric environment, and necessary actions are discussed in detail and the solutions are designed accordingly. The MINLP and MILP optimization models

are designed with realistic formulations. The complexity issues of the MINLP model are taken into consideration, and nonlinearity is eliminated through linear approximation for targeting the larger-scale environments. A heuristic based on the nearest-fit algorithm is developed to assess the performance of the optimization models and generate a solution by addressing the scalability issues.

- Instead of Network Function Virtualization (NFV) and Service Function Chaining (SFC) operations, the network slicing concept is applied to satisfy the requirements of the novel services that are envisioned to be integrated into the next-generation networks. An MIP model is proposed to minimize the degree of SLA violations. Additionally, a heuristic implementation is provided to eliminate the scalability problem of the MIP model. For achieving the proposed objective, both networking and computational resources are virtualized, individual fractions of the physical resources' capacities are reserved, and isolation is maintained among the slices for each service type to ensure that the requirements are met.
- The proposed SDN method enables service-centricity without any modification to the existing protocol stack by exploiting the TCP port number and DSCP (Differentiated Services Code Point). This method assumes the realistic view that the mobile end-users exploit the resources of the edge servers via computation offloading that can occur at the method/function resolution.
- A long-term solution for implementing the service-oriented behavior by using P4 language is proposed. This proposal focuses on service offload operations with a sub-service resolution and tries to achieve the necessary objectives with a compatible operation with the TCP/IP protocol stack by enabling a fully-programmable network environment.

1.2. Thesis Outline

The content of the thesis is organized as follows: Chapter 2 presents an overview of the related studies in the literature and highlights the contributions of the thesis. In Chapter 3, a background on service-centric networks, Edge Computing, network slicing, and programmable network paradigms are presented.

Chapter 4 provides the problem definitions and optimization model formulation of SLA-aware resource allocation scheme for service-oriented multi-tier computing infrastructure. Additionally, it discusses how the proposed MINLP model is simplified by adopting the linear approximation methodology and presents the operations of the nearest-fit heuristic implementation. Chapter 5 discusses the problem definition of optimal network slicing, formulates the problem with an MIP model, and provides the details of the NESECS heuristic algorithm. Chapter 6 presents the service-centric network implementation by using SDN and P4 concepts as two different solution approaches. Additionally, this chapter presents a use case of fall-risk assessment service and orchestration mechanism as SDN in a multi-tier computing system. The thesis is concluded with Chapter 7, with a brief summary of the content, discussion upon the contributions and possible future works in this research area.

2. RELATED WORKS

In this chapter, the related works in the literature are discussed by highlighting the original contributions of this thesis. The building blocks of the implementation and optimization of service-centric behavior in a multi-tier computation system are analyzed thoroughly, and the literature is reviewed accordingly. The studies fall under the topics of the service-centric networks, SDN, P4, task offloading and resource allocation, multi-tier system design, service placement, and network slicing are discussed by presenting the features of these studies.

2.1. Related Work on the Service-centric Network Implementation

In a study made by El Mougy [14], SDN programmability is asserted as a solution to enable core functionalities of ICN such as content forwarding, in-network caching and multicasting. The study focuses on the architectural modifications that may be applied to SDN and OpenFlow to leverage ICN and manage the dynamic environment. Since the recent version of OpenFlow, currently v.1.5.1, does not support a matching mechanism concerning the named data, carrying out a smooth integration is not a simple operation.

An approach for Software-Defined Content-Centric Networking (SDCCN) is proposed by Charpinel et al. for integrating SDN and CCN to provide a caching policy [15]. The system is designed to involve users interested in contents, CCN switches and a logically centralized controller. The users generate a request for content (i.e., the interest packet), and the network propagates these packets to the nearest provider. Then the content is propagated back to the user using the reverse route. All these routing processes are managed by the controller.

There are solutions presented for supporting the ICN structure with the help of SDN concepts, and Salsano et al. provide both a long-term solution for ICN without

considering the current limitations in SDN and a short-term solution to experiment with the combination of ICN and SDN [16]. Since the SDN still has missing points to support the ICN environment fully, the long-term solutions discuss several choices for the packet format.

Named Data Networking (NDN) [17] is a popular ICN architecture based on CCN. A study proposed by van Adrichem et al. integrates SDN into NDN for leveraging an application-specific forwarding mechanism [18]. In order to distinguish the traffic generated by ICN from traditional IP traffic, this study introduces a layer to OpenFlow. It implements a specific communication channel and controller module that cooperates with the existing OpenFlow communication channel.

There are also solutions for combining the ICN and SDN by utilizing the existing features. One of them is proposed to integrate an ICN solution to the OpenFlow networks, which maps the content name carried in IP options into a tag transported in TCP and UDP port fields that can be used as a matching field by the OpenFlow [19].

Serval is a proposal for SCN, which aims to provide a solution for providing services with multiple servers to the mobile users [20]. Their motivation is that the traditional network structure does not fit the dynamic service environment. In order to mitigate this challenge, they propose the Serval architecture with a Service Access Layer above the network layer. This new layer provides name-based routing, decoupling of control and data planes, and other service-level operations.

In addition to SDN, P4 becomes a promising solution for carrying out the key functionalities to enable a service-centric model. Related work is prepared by Signorello et al. [21], which utilized P4 to program the data plane to support ICN structure. The main objective of the study is implementing the NDN paradigm, which is considered an instance of ICN, through the fully programmable environment provided by P4. The components of an NDN instance are implemented as a P4 program, and the implementation details are explained in detail.

2.2. Related Work on the Task Offload Operations

An optimal resource allocation scheme is crucial for providing user-centric services depicting various requirements in a high-performance manner. In the literature, optimization problems are defined in various contexts, and specific approaches are proposed in task offloading. This section discusses the related works that focus on optimization models of task offloading, service-centric network design and user-centric services with various requirements.

In study made by Wang et al. [22], the authors propose a joint optimization model for MEC-enabled systems, which includes an optimal resource allocation scheme aiming to minimize the energy consumption of access points concerning the latency constraints. JCORAO [23] is an alternative approach for providing a joint optimization of resource allocation and computation offloading methodology for Mobile Edge Computing (MEC). The proposed optimization model provides a policy for wireless channel allocation and computation offload scheduling in heterogeneous network environments. The simulation results depict that JCORAO can decrease the energy consumption of mobile terminals and the task completion time.

As discussed in the literature, MEC will play an essential role in the next-generation cellular networks, 5G. In a study made by Zhang et al. [24], an optimal resource allocation and task offloading scheme are introduced to enhance the users' gains through task offloading. The gain in this context is described as the decrease in the task completion time and energy consumption by mobile devices. This optimization model is designed for MEC environments with multiple servers that are deployed near the base stations. In another study, the authors the optimal offloading problem is solved with a game-theoretic approach [7]. The problem is formulated as a game among multiple users that offload their computational tasks to nearby MEC servers.

The dynamicity of the edge network is taken into account for achieving an optimal traffic assignment. In a study made by Ceselli et al. [25], a framework is devised to

optimize the traffic assignments to MEC servers to satisfy the demands and meet the capacity constraints of the facilities. In this study, a real dataset of mobile traffic is utilized to evaluate the performance of the proposed optimization model.

Besides MEC-based systems, Fog Computing is also a popular approach for providing computation services at the network edge. A multi-objective optimization methodology is proposed by Liu et al. [26], which utilizes queuing models for addressing the problems of energy consumption, execution delay and the unit cost of task offloading to a nearby fog server.

While service latency is considered a constraint in some optimization problems, another set of studies aim to achieve minimized latency within the network. In a study made by Ren et al. [27], the authors suggest a model that minimizes the latency, considering the allocation of both networking and computation resources. In this study, three different task execution schemes are investigated: (i) local execution, (ii) edge execution and (iii) hybrid methodology, where some of the data is processed locally while the rest is transmitted to the edge server. The optimal resource allocation model is tested for the hybrid execution methodology through comparison among the execution models. Similarly, Chen et al. [28] formulate the offloading problem with an SDN-based solution for minimizing the delay while considering the energy constraints of the mobile devices. Since MEC is proposed to serve latency-intolerant and computation-intensive tasks within ultra-dense networks, the location where the task is processed becomes an important criterion. The proposed solution employs controllers at macro cell BSs and decides on the task assignment and resource allocation plans considering the residual energy on the mobile device at that time.

The offloading problem is not only studied for Edge Computing systems but also for multi-tier computing architectures. The study made by Tong et al. [29] considers a hierarchically deployed set of computation resources. An adaptive workload assignment algorithm is proposed to enhance the resource utilization efficiency among different tiers, especially during peak times.

Table 2.1. Summary of the related studies on task offloading optimization.

Study	**Differentiated Service Types**	**Realistic Delay Model**	**Methodology**	**Fairness**
[31]	-	Only Computational, Objective	Simulation	-
[32], [26], [33]	-	Available, Objective	Simulation	-
[34], [23], [7], [30], [35]	✓	-	Simulation	-
[36]	✓	-	SDN Simulation	-
[25], [37]	✓	-	Optimization + Heuristic	-
[38], [39]	✓	Only Network, Objective	Simulation	-
[40]	✓	Available, Objective	Simulation	-
[41]	✓	Available, Objective	Optimization + Simulation	-
[42], [43]	✓	-	Testbed + Simulation	Among Users
[44], [45], [46], [47]	✓	-	Simulation	Among Users
[48]	✓	Only Network, Objective	Testbed	Among Services
[49]	✓	-	Testbed	Among Services
This thesis	✓	Available, Constraint	Optimization + Heuristic	Among Services

The service or user characteristics should also be considered while achieving an optimal task offloading and resource allocation scheme. A multi-objective optimization model is proposed by Malekloo et al. [30] to handle balance among QoS requirements of SLA definitions and the energy consumption of virtual machines. Within this context, a multi-objective Ant Colony Optimization (MACO) is applied to save energy and minimize the possible SLA violations in cloud environments.

Zheng et al. [50] formulate a resource allocation scheme to provide fairness among the users by considering the energy constraints and QoS requirements. The QoS requirements are integrated into the optimization model as a constraint, where a certain level of QoS should be guaranteed. However, the QoS metrics and criteria are not discussed in detail. In a study made by Kolomvatsos and Anagnostopoulos [51], the task allocation problem is divided into two steps where the first step tries to decide

whether the task should be offloaded or not. In the second step, the task is decided to be executed on a group of peers or a fog server. The decision process is directly related to task characteristics such as priority, complexity, and execution requirements.

On the other hand, an SDN-based algorithm is proposed to provide load balancing among the links considering the service types. The northbound applications implemented over the SDN controller can be aware of the service type and extract the load information throughout the network. The proposed system differentiates services through their QoS requirements, including link bandwidth requirements, packet loss rate and acceptable delay values. A similar SDN-based approach is provided by Bahnasse et al. [52], where QoS requirements are considered for VoIP, video streaming, HTTP, and ICMP traffic flows. The main QoS criteria for differentiating the services are jitter, latency, packet loss rate, and delay time.

Healthcare applications are concrete examples of user-centric services with latency limitations. In a study made by Elhoseny et al. [53], the main objective is enhancing the performance of the healthcare systems by addressing the issues of patients' data storage, real-time interaction, and execution time. A Genetic Algorithm, Particle Swarm Optimization and Parallel Particle Swarm Optimization are implemented to optimize the resource allocation through VM selection. The specific criteria that are taken into consideration by the proposed model are the CPU utilization values, turn-around time, and waiting time. Another framework that focuses on the requirements of healthcare applications is proposed by Chen et al. [54]. The edge-based healthcare system addresses the issue of computing resource allocation for monitoring and analyzing the health status of the end-users. To enhance the QoE levels and meet distinct requirements of the healthcare applications, the users are prioritized according to their health risk levels for allocating the limited resources.

The properties of the related studies and approaches utilized there are presented in Table 2.1. As observed, some of the studies do not provide a detailed delay model in order to calculate the end-to-end service latency for each request, considering both

networking and computational resources. On the other hand, there also exist studies that include a realistic delay model by only dealing with the network latency or the delay contributed by all the computational resources. One way or another, these studies formulate an objective function to minimize the service latency, instead of defining the service latency as a constraint defined by the SLAs.

Besides this, different service types and their unique requirements are not examined exhaustively by other studies. Additionally, most of the studies that discuss the concept of differentiated service types do not address the fairness among various services, considering their requirements. In terms of the main approach for solving the problem of resource allocation and task assignment, some of the studies evaluate the performance of the proposed solutions through a simulation environment and do not report an optimality gap.

As a result of examining the properties of related studies and their approaches, the original contributions of this thesis and unique benefits of the proposed techniques are summarized as follows:

- Differentiating service types considering their distinct requirements,
- Providing a detailed delay model including both networking and computational resources,
- Investigating a service-oriented environment with the multi-tier structure of the computational resources,
- Formulating the end-to-end service delay as a constraint to ensure that service requirements defined in the SLAs are satisfied for more realistic use cases,
- Providing two different optimization models and evaluating the performance through randomly generated instances,
- Maintaining the fairness among different service types,
- Implementation of a heuristic approach based on the nearest-fit algorithm for addressing the scalability issues.

2.3. Related Work on the Network Slicing Operations

The network slicing concept, configuration, and optimization of the slice instances currently have an essential place in the literature. Several methodologies may be applied within the network while maintaining the isolation among dedicated virtual resources. Software-defined networking (SDN), NFV, SFC, and other supportive technologies may play a significant role in the process of integration of the slicing concept into a dynamic networking environment [55]. Regardless of the technology or approach adopted to leverage the isolated slice instances, optimization of these operations is of a particular importance for satisfying the high-performance expectations.

PERMIT [56] is a network slice orchestration mechanism for enhancing the personalization of the next-generation cellular networks. It is aimed to replace the uniform services that are currently offered by LTE with a set of services under various requirements. This study proposes to integrate NFV, SDN, Edge Computing, and cloud on top of the network slices to remove the overhead of uniform network services. Slices are configured and instantiated in per user resolution. However, it is stated that there may be cases that necessitate slicing per device or application.

The optimization of the slice management is critical for achieving isolation among slices. In a study by Addad et al. [57], a slice instance is defined as a set of SFCs that are composed of Virtual Network Functions (VNFs). An MILP model and a greedy heuristic approach are proposed to minimize the slice deployment and operation costs. Additionally, assignments and slice mapping operations are also optimized. However, only the servers within the core network are considered as the sliced resources. In order to evaluate the performance, the proposed MILP model is run on the Gurobi [58] optimization solver, while the proposed greedy heuristic is implemented with Python.

Guan et al. [59] consider both networking and computational resources throughout the slicing processes, including wireless access. A mathematical model is proposed for constructing the slice requests and mapping them to the physical infrastructure.

By this mapping, the deployments of the VNFs are optimized, and links are selected for the SFC operations. The proposed solution adopts the complex network theory, in which the primary objective function differs from case to case: enhanced mobile broadband (eMBB), massive machine-type communications (mMTC), and ultra-reliable and low-latency communication (uRLLC). For instance, for eMBB slices, the objective is to maximize the remaining capacities of the resources, An Integer Linear Programming (ILP) model and a heuristic algorithm are proposed, and the performance of the solutions is evaluated with simulations.

While maintaining the isolation among the slices, Kasgari et al. [60] minimize the transmit power between the base stations and users. Two different slices are considered by the proposed framework: reliable low latency (RLL) and self-managed (capacity limited). A stochastic optimization solution based on Lyapunov drift-plus-penalty method is applied to achieve slices for wireless access. The necessary level of isolation is maintained if the number of users to be served does not exceed the maximum value specified in the contract. The proposed solution is evaluated through simulations.

Han et al. [61] consider only the networking resources for the slicing operations. An online optimizer based on genetic algorithms is proposed for the network slicing problem and inter-slice resource management. While the proposed approach aims to maximize the utility within the network, the utility can be customized according to the need of slices. The presented utility model and long-term strategy optimization model are non-convex. Thus, a heuristic based on the genetic algorithm is proposed. The performance of the genetic algorithm implementation is evaluated through simulations.

The network slicing concept is also adopted for the vehicular edge computing environments. In the study made by Xiong et al. [74], a slicing scheme is proposed to address the problem of dynamic mobility of the vehicles and allocation of the fog resources. The proposed slicing approach is based on Monte Carlo tree search to accommodate the network traffic in an efficient manner. The main focus is on slicing Radio Access Network (RAN) and fog nodes to minimize the V2V outage probability.

Table 2.2. Related studies on network slicing optimization and their properties.

Study	Objective	Slicing the Netw. Resources	Slicing Comp. Resources	Per Service Slicing	Service Placement	Resource Reservation	Optimization Solver
[57]	Min. the operation costs	✓	✓	✗	✓	✗	✓
[59]	Various objectives	✓	✓	✗	✓	✗	✗
[60]	Min. the power consumption	✓	✗	✗	✗	✗	✗
[61]	Max. the network utility	✓	✗	✗	✗	✗	✗
[62]	Min. the ratio between actual and maximum latency	✗	✓	✓	✓	✓	✗
[63]	Max. the profits generated by slice rentals	✓	✓	✓	✗	✓	✗
[64]	Minimizing the total latency of computing tasks	✓	✓	✗	✓	✓	✗
[65]	Maximizing the total number of linked resource blocks	✓	✗	✗	✗	✗	✓
[66]	Max. the fairness	✓	✓	✓	✗	✗	✗
[67]	Max. the resource efficiency and profit	✓	✓	✗	✓	✗	✗
[68]	Min. the delay	✓	✓	✗	✓	✗	✓
[69]	Max. the overall system utility	✓	✓	✓	✓	✗	✓
[70]	Max. slice utility	✓	✓	✗	✓	✗	✗
[71]	Max. revenue of the service providers	✓	✓	✗	✓	✗	✗
[72]	Max. the total throughput	✗	✓	✗	✓	✗	✗
[73]	Min. the mean response time in the system	✓	✓	✗	✗	✗	✗
[74]	Min. the V2V communication outage probability	✓	✓	✓	✗	✗	✗
[75]	Min. the weighted hops	✓	✗	✗	✓	✗	✗
This thesis	Min. the SLA violations	✓	✓	✓	✓	✓	✓

In the study made by Wang et al. [67], an optimization framework is proposed to both maximize resource efficiency and profit by defining two separate problems from the perspective of a slice provider and a slice customer. A distributed algorithm is proposed for analyzing the tradeoff among the objectives and handle the utility maximization for both operators and customers. Since the proposed approach considers

the dynamic management of the slice instances, the simulations compare the results obtained through the distributed algorithm with a static slicing approach where slice requests are processed without any reconfiguration.

The requirements of the network slices are also recently addressed by the application of learning algorithms [76]. Hua et al. [77] propose a Deep Reinforcement Learning-based approach for useful resource management in network slices. The varying demands within RAN of 5G networks are considered to satisfy the SLA requirements of the services. Similarly, Chergui and Verikoukis [78] apply a Deep Learning methodology to provide a dynamic end-to-end network slicing functionality. The required resources are estimated through the neural network so that the SLA requirements are not violated.

According to the details of the reviewed studies and the analysis of more that is depicted in Table 2.2, it can be deduced that our proposed solution for the optimization of the network slicing operations differs from the related ones in the literature. Most of the studies do not consider the virtualized resources of both networking and computational resources with capacity allocation for each slice. Besides, the slicing concept is generally applied for 5G networks with the proliferation of NFV technology. However, the proposed solution in the thesis is generally applicable in heterogeneous environments, and the realistic SLA requirements of the services are considered with the objective of the minimization of the SLA violation in a network. The proposed MIP model solved with optimization tools, and a heuristic implementation is proposed for the scalability issues. The performance of both solutions is analyzed with an extensive set of test environments.

3. BACKGROUND

The main objective of this thesis is to implement the service-centric behavior through programmable network paradigms and optimizing the operations within this environment. Before going into the details of the corresponding solution approaches and methodologies, it is essential to provide a brief overview of the technologies that are the elementary units of this thesis. In this section, the background information related to service-centric networking, Edge Computing, network slicing and SDN may provide a general understanding of the solution methodologies.

3.1. Service-centric Networks

The ancestor of the service-centric model is the Information-Centric Networks (ICN), which aims to transform the operations of the traditional networks [79]. The current network infrastructure and TCP/IP protocol stack focus on the location of the hosts (i.e., IP addresses). In other words, there is a host-centric mechanism where routing operations are executed through the IP addresses within the packet headers. However, ICN proposes that content itself becomes more important than its location [80]. This transformation requires a technique to access the content independent from its location. Therefore, instead of explicitly identifying a definite server by its IP address, there is a need for a naming scheme that describes the content itself.

The terms ICN [81] and content-centric networking (CCN) [15] become more popular in the literature for addressing the problem of traditional IP-address or location-centric model. Although they share the same objectives and architectural features, there are some differences between them, such as name resolution, routing, caching and Named Data Objects (NDO) granularity [82]. SCN emerges as an extension of ICN as the Internet becomes the pool of novel services. In addition to the contents, SCN states that services are the main elements of the Internet, and they need to be handled with an efficient orchestration scheme to deploy and replicate the instances over the

servers dynamically [83]. It is discussed that content-centric schemes should be generalized towards a service-centric architecture because the Future Internet is envisioned as a supporting mechanism for services such as file storage/retrieval, video streaming and location-based services [9]. As new service types emerge with innovative communication and computation methodologies, service management and orchestration operations, client mobility, and dynamic environment characteristics are not aligned well with the current TCP/IP protocol stack. Correspondingly, SCN is proposed to support effective service management [20].

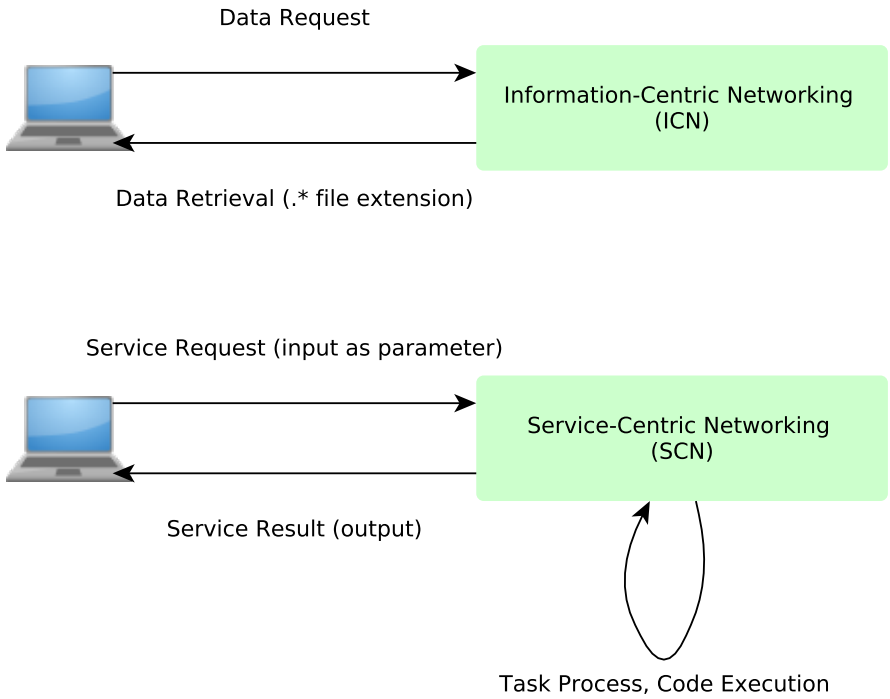

Figure 3.1. Difference between ICN and SCN.

In SCN, clients can request for a service type by sending the input data required for the code execution at the server-side, and output is formed as a result [33]. In order to exemplify this operation, let us consider an end-user requesting the face recognition service with an image as input. The corresponding server hosting this service executes the necessary code routines and generates a response with the execution result. Even though both ICN and SCN possess parallel objectives, the main difference between them is the task execution with input data, which is illustrated in Figure 3.1.

Predictably, modifying the existing protocol stack that is around for decades is not an easily applicable solution. Therefore, a supportive mechanism that is interoperable with the existing infrastructure and technologies is mandatory. From this perspective, the orchestration of the services requires a set of processes, including a scheme of service identification, service discovery, sub-service resolution and service mobility.

3.1.1. Service Identification

Since the legacy TCP/IP protocol stack was initially designed for providing end-to-end communication through IP addresses and port numbers, it remains incapable of handling the information-centric functionalities. The emergence of specialized gadgets and an increase in the number of services requested by the devices are reforming ICN as a service-centric model. Services that are infeasible to be executed on the device due to capacity restrictions can be deployed over the servers and executed there [10]. The service replications may be distributed all over the network, and requesting them via a specific IP address cannot be the desired procedure to satisfy the user experience. Therefore, a model that uniquely identifies the services, including their sub-services, and eliminates the obligation of prior knowledge about the destination server's IP address is still necessary.

The naming scheme should be unique for each service type. Like DNS (Domain Name System) operations, the user application embeds the information of service identity into the request, and the service orchestrator resolves it. Then, this request is forwarded to a server, which is determined by the orchestrator capable of handling and executing the particular service procedures. All these complex operations need to be hidden from the end-user for an enhanced experience with real-time interaction.

3.1.2. Service Discovery

The service components (i.e., code and instructions) may be distributed over a wide area to improve the user experience and fulfill the requests as soon as possible. As

mentioned, the user application has information only about the requested service and its identification. The orchestration mechanism inspects the request and maintains the set of servers that can execute the corresponding service routines. It achieves these resolving operations through a mapping table that is frequently updated with the information supplied by the servers after each service deployment and replication.

Including the user application in these operations not only increases the complexity but also deteriorates the user experience. Therefore, the only action that a user ought to take is to create a new request by specifying the service identity, and the rest should be isolated from the user.

3.1.3. Sub-Service Resolution

As proposed by Mobile Cloud Computing (MCC), the computation offloading can occur at the method/function resolution [84]. In other words, each part of the code can be executed independently from the remaining parts. While some of the methods can be executed on the device itself, the remaining can be offloaded to a server with an RPC/RMI fashion to conserve energy and decrease the execution time.

Similar approach can be applied to the service context. For example, a face recognition service consists of several procedures such as noise reduction, face detection and feature matching. The implementation needs to follow a certain way to divide these methods into segments to be executed separately in a parallel or sequential manner. Hence, we need to define a sub-service resolution where the user does not need to request the service as a whole. Instead, the application should be able to request only an intended subset of the procedures. This methodology provides a realistic environment and models the actual granularity between the devices and the servers.

3.1.4. Service Mobility

When the demand for a service increases to a level where the server fails to meet the requests, it is possible to deploy its replication on another server. Applying this feature to all services within the network creates a dynamicity that is difficult to handle. Tracking the new service deployments and keeping the mapping table updated is critical in the service-centric model. In addition to this, service or virtual machine (VM) migration may be realized for energy efficiency and performance enhancement. Providing a seamless migration without disrupting the service continuity for the end-user is essential for the multi-tier orchestration mechanisms.

3.2. Edge Computing

The proliferation of edge devices, including smart glasses and smartwatches, creates a necessity to process at least some of the data at the edge rather than carrying them to the remote data centers. It is a vital operation for minimizing not only the service delay but also energy consumption. Recently, there have been many proposals for the operation and architectural design of the Edge Computing systems. It is no coincidence that the terms Mobile Cloud Computing (MCC) [85], Cloudlet [86], Fog Computing [87], Edge Computing [88], and Multi-access Edge Computing (MEC) [89] are all hot topics in the literature. All these proposals define various practical implementations for Edge Computing. These approaches have common grounds when carefully inspected but differ in their target use cases and deployment models. Edge Computing is an umbrella concept that covers a range of practical schemes.

3.2.1. Edge Computing Paradigm

Although intended for different parts of the overall network, edge computing and cloud computing are interrelated paradigms that complement each other. Analyzing the features of various edge proposals [90, 91], a comparison between cloud and edge computing technologies is depicted in Table 3.1.

Table 3.1. Differences between cloud and edge computing.

Requirements/Features	**Cloud Computing**	**Edge Computing**
Latency	High	Low
Network Access Type	Mostly WAN	LAN(WLAN)
Server Location	Anywhere within the network	At the edge
Mobility Support	Low	High
Distribution	Centralized	Distributed
Task/Application Needs	Higher computation power	Lower latency
User Device	Computers, mobile devices (limited)	Mobile-smart-wearable devices
Management	Service Provider	Local Business
Number of Servers	High	Low
State	Soft and hard state	Soft state

Figure 3.2 compares the various approaches for providing the necessary computation power to the users at the edge. Conventional datacenter design is highly regular with identical servers and networking hardware aligned in a grid-like fashion. Although at the edge of the network, MEC still depicts a more rigid and well-defined structure than other edge computing proposals. The main reason for this behavior of MEC is its envisioned existence in a telecommunications infrastructure, which is inherently regulated. Functionality served over MEC will not be some individual services locally available to the edge users but will be highly controlled and orchestrated in accordance with the overall state of the 5G network.

Compared with MEC, Cloudlets and Fog Computing solutions have less stringent hardware and application execution model constraints. A Cloudlet hardware can be a micro-sized server in a coffee shop where WLAN allows users to carry out code offloading. Fog servers can be co-located on a networking device to handle the IoT traffic at the edge. In that respect, Cloudlet and Fog have much wider design spaces allowing irregularity.

The utilization of various edge computing proposals and traditional cloud servers is illustrated in Figure 3.3. Smart devices, vehicles and IoT-related appliances can

offload their tasks to the edge servers, accessible at one hop through different access technologies. Simultaneously, a subset of requests can be directly forwarded to the traditional cloud datacenters through WAN, or the edge servers may operate as an intermediate computation layer to pre-process the offloaded tasks.

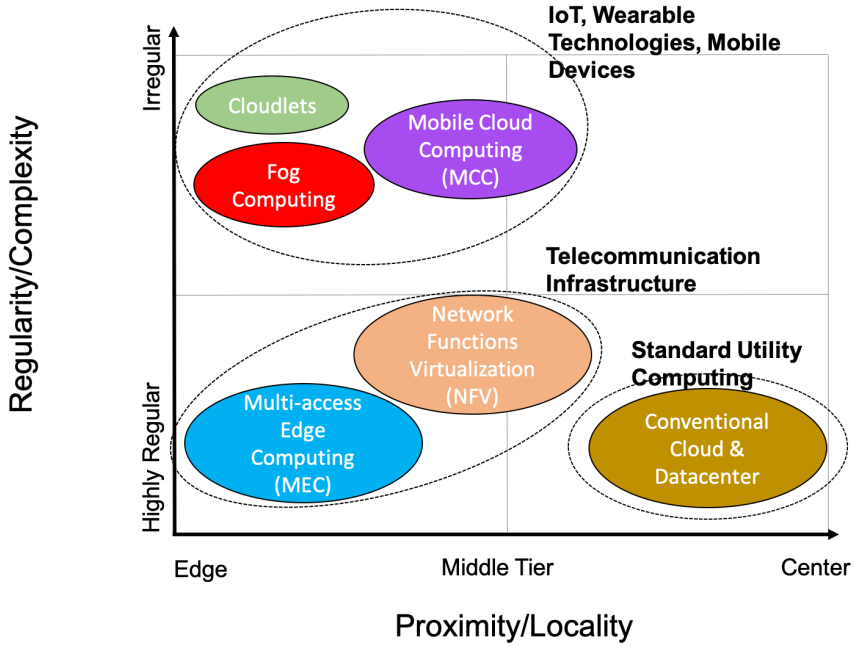

Figure 3.2. Comparison among the edge computing spectrum.

3.2.2. The Need for Edge Computing

The causes that lead to the emergence of Edge Computing are discussed briefly to form a basis for the edge server proposals.

3.2.2.1. Real-time QoS & Delay Sensitiveness. Although these end-user devices are as powerful as they have never been so far, most of them still lack enough capacity for accomplishing real-time use cases with the pre-defined QoS requirements. Cloud computing is acknowledged as a remedy for limited-capacity devices by providing a pool of computation and storage resources. However, wearable devices and IoT are designed for delay-sensitive use cases. Since most of these devices demand high QoS requirements because of the mobility, interactive environment and real-time requirements, legacy cloud servers cannot be the sole solution because of the indefinite WAN delay.

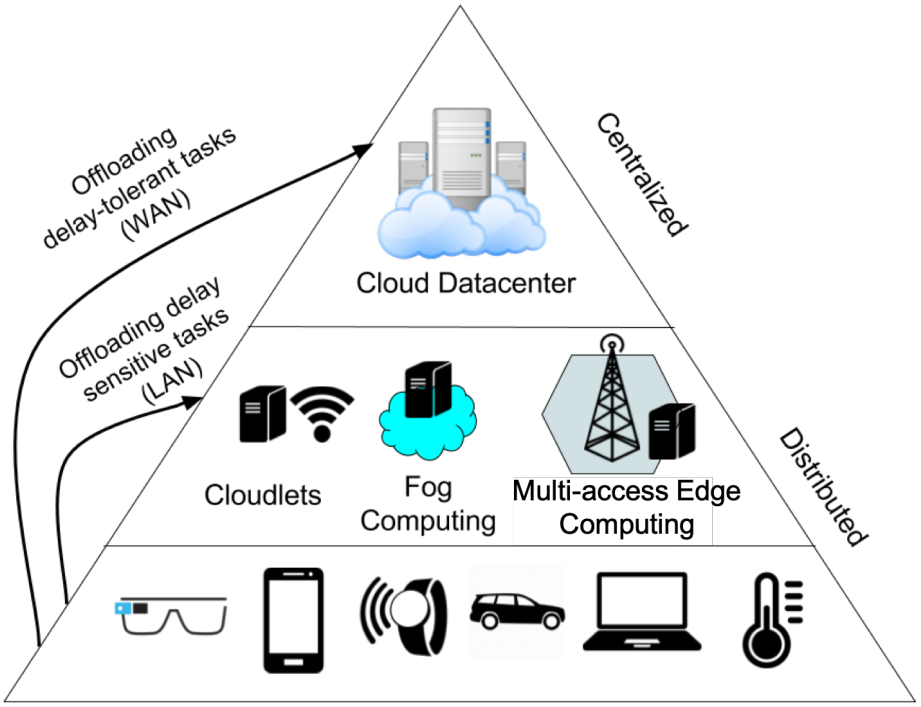

Figure 3.3. The possible cooperation of edge technologies and cloud computing.

3.2.2.2. Battery Lifetime. Energy consumption is one of the most critical parameters for mobile devices [92]. Although smartphones' processing capabilities enhance steadily, their battery life does not improve at the desired rate.

One of the main objectives of task offloading is to decrease energy consumption. The related studies show that offloading reduces the total energy consumption [93–95]. Offloading can be achieved through two approaches: (i) cloud servers and (ii) edge servers. Although the offloading operations inherently decrease the energy consumption, utilization of edge servers decreases it further. Ha et al. [96] analyzes energy consumption rates for applications such as face recognition and augmented reality. It is stated that offloading tasks to the edge servers results in lower energy consumption than cloud servers. Unsurprisingly, executing these applications on the device leads to the highest energy consumption among all methodologies.

3.2.2.3. Regulating Core Network Traffic. The limited bandwidth of the core network makes it vulnerable to congestion. In 2023, it is expected that there will be 13.1 billion

mobile devices [97]. As a result, operators face difficulties in managing cumulative data traffic with varying sizes and characteristics.

In the traditional approach, the traffic generated by the edge devices flows through the core network to access cloud servers. If we keep the traffic at the edge, the burden on the core network is relieved, and the bandwidth utilization is optimized [98]. This shift in operations prevents the consumption of the limited bandwidth of the core network by billions of devices at the edge. Therefore, the core network traffic becomes manageable in size and the orchestration functions are simplified.

3.2.2.4. Scalability. The tremendous increase in the number of end-user devices creates a significant scalability problem [99]. In order to support the dynamic demands and continuously varying expectations, the cloud can be scaled accordingly [100]. However, sending tremendous volumes of data to cloud servers create congestion within the datacenters [101]. The changing characteristics and massive amount of the data traffic generated by IoT and wearable devices make the operators' duties more difficult. With these Cloud computing's centralized structure falls short of providing a scalable environment for the data and applications with these rates.

3.3. Network Slicing

Deploying the replications of the service code throughout the edge of the network and offloading the end-user application tasks to the resources in the vicinity solely is not useful in practice. Assuming that various services coexist in a heterogeneous setting and the offload operations are handled by the same physical infrastructure, the demand or load increase for a service may degrade the adjacent services' performance. The congestion caused by the services and introduction of a new application in this environment may affect the performance of all previously deployed services. Therefore, it becomes necessary to provide isolation among the service types and share the physical resources fairly.

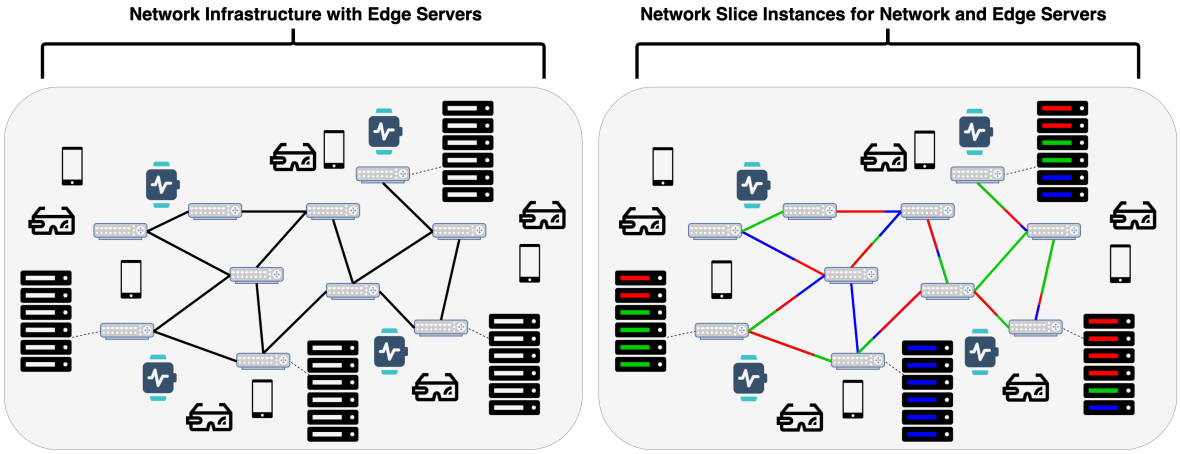

Figure 3.4. Different colors representing isolated slice instances for each service type within an edge computing environment.

The network slicing approach, which is proposed initially by the NGMN (Next Generation Mobile Networks) Alliance, aims to achieve the desired level of flexibility and scalability by enabling the isolation among the slice instances through logical networks [102]. By partitioning the physical resources and assigning the allocated capacity of the virtual resources to the slice instances, the isolation can be preserved for accommodating high-performance services. According to Open Networking Foundation (ONF) [103], the bandwidth of a network link, the forwarding tables and processing capacity of a networking node, and the computation capacity of a server can be partitioned into virtual resources. By reserving a specific physical resource capacity, either networking or computational, through the virtualization approaches, a slice instance may not be affected by the fluctuations or faults within an adjacent slice. Thus, the projected performance can be maintained continuously. An example scenario of slicing the network into isolated partitions is depicted in Figure 3.4. In an environment where emergency applications share the same physical infrastructure with other edge computing use cases, different colors on the network links and edge servers correspond to isolated slices for different service types.

The network slicing paradigm is initially targeted for 5G networks where the verticals' diversified requirements should be satisfied over the same networking infrastructure [104]. A slice instance may be composed of VNFs and the operations of SFC

for traffic steering. As envisioned by the International Telecommunication Union (ITU) and 5G-PPP, there are be three different application classes that should be handled in the next-generation cellular networks: eMBB, mMTC (e.g., smart transportation and autonomous driving [105]), and critical services as uRLLC [106].

For example, in a vehicular communication scenario, to provide reliable communication between the vehicles and road side units, the envisioned applications for autonomous driving, vehicular emergency, and infotainment should be allocated with adjacent slice instances on the same physical infrastructure [107]. As 5G networks become a catalyst for innovative vehicular scenarios, the network slicing concept takes an essential place in providing the necessary QoS level required for ultra-reliable vehicular communication, and safe transportation [108]. Considering a vehicular emergency scenario, an ambulance or an emergency team should arrive at the place of an accident immediately [109]. The network and computation resources should be allocated to an emergency service slice to maintain high connectivity and QoS for the related applications (e.g., path recommendation and emergency call).

These concepts apply not only to the operators but also to the service providers. In order to ensure all these divergent demands in the form of performance requirements (e.g., data rate, latency), end-to-end slices should be initiated through dedicated virtual resources with optimized capacities, both within the network for traffic routing and on the edge servers for the execution of the complex application routines.

3.4. Programmable Networks: Software-Defined Networking & P4

3.4.1. Software-Defined Networking and OpenFlow

It is envisioned that SDN can offer a remedy for a broad set of challenges encountered within the scope of the traditional networking approach and operations. Similarly, when the intrinsic properties of SDN are considered, fruitful cooperation with the edge computing framework can be foreseen.

SDN is proposed for using the limited network resources optimally and enabling flexible network management by separating the control layer from the data layer [110]. Since the main logic is extracted from the networking nodes that are no longer capable of making decisions on their own, it is concentrated on the software-based controller with a general view of the underlying network [111–114].

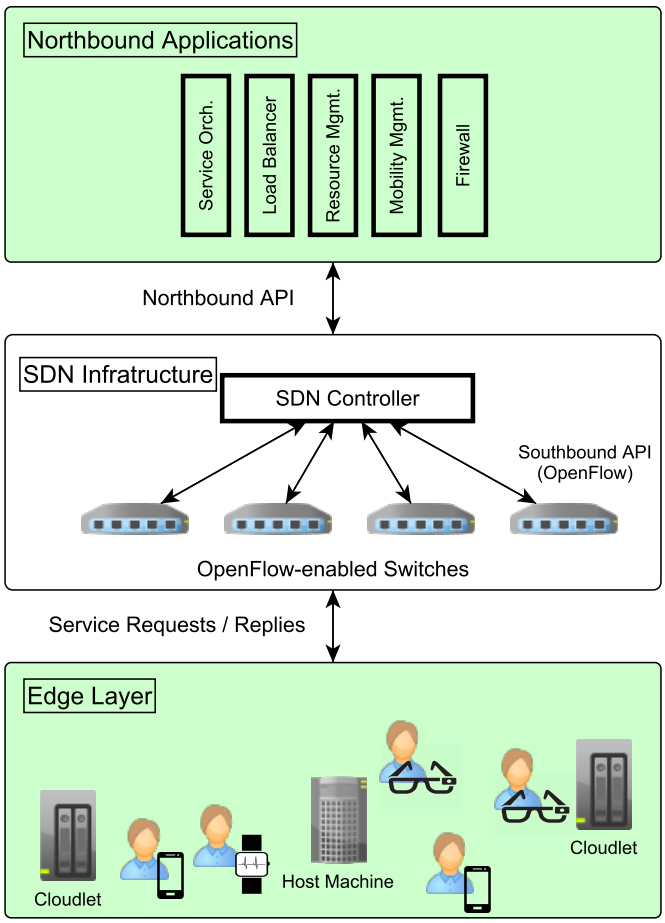

Figure 3.5. A view on SDN architecture for edge computing.

Routing and forwarding behavior of the network elements can be inquired and modified upon customized policies defined by the SDN controller. These policies are applied to modify and populate the flow tables of the network nodes the with match/action flow rules. When there is an incoming packet, the corresponding switch initially checks its flow table to match this packet with any rule. If found, it applies the action defined by the matched flow rule. Otherwise, the switch buffers this packet for a limited time and forwards an encapsulated replication to the controller for determining

the most appropriate action according to the policy [115]. The SDN controller then decides on an action to be applied on this packet and installs the necessary flow rule on the corresponding switch for applying the same action on similar packets in the future.

All operations of SDN and the flexible communication between the controller and the switches are carried out with the OpenFlow protocol [115,116], which is developed by ONF [117]. OpenFlow represents the main functionalities of SDN, such as managing the flow tables on the forwarding nodes, populating them, defining flow rules, gathering statistics, and many other managerial operations.

The layers of the envisioned SDN architecture for orchestrating a multi-tier edge system is depicted in Figure 3.5:

- Edge layer
- SDN infrastructure
- Northbound applications

There are servers positioned at the bottom layer to handle the user requests and various edge devices demanding services through offload operations. At the intermediate level, the traditional SDN infrastructure composed of OpenFlow-enabled switches and the SDN controller is organized. Although SDN considers them as separate planes (i.e., data and control planes), it forms the typical SDN infrastructure where the responsibilities of both the controller and forwarding nodes are clearly defined. The novelty of this adapted SDN architecture is specified by the top layer, which consists of customized and virtualized northbound applications that define the behavior of the control mechanism and policy to be applied on the managed network. Service management and orchestration, optimal resource allocation and mobility management can be concrete examples of orchestration applications for an edge environment. These applications that define the network behavior can communicate with the controller through the northbound interface API implemented by the controller [113]. The high-level commands as policy rules generated by the northbound applications are forwarded to

the controller via the northbound API, which is not standardized yet [114]. Then, the controller transforms these commands into low-level OpenFlow messages to be sent to the data plane [118]. In distributed control plane where several controllers are deployed over geographically distant servers, a mechanism is necessary to provide communication between the controllers for synchronization through east/west interfaces.

3.4.2. P4 Language

The fixed-function chips available in the networking nodes and TCP/IP protocol stack remain infeasible to accommodate the functionalities of the service-centric approach. Handling and routing the network packets according to a new protocol is challenging for the traditional network infrastructure. Integrating new protocols without replacing the networking devices or upgrading the firmware is not a practical approach. Therefore, there should be a certain level of programmability and flexibility to realize the desired functionalities of an innovative operation, such as implementing the service-centric model.

Although the current pioneer of the programmable networks in the form of SDN and OpenFlow provides a solution alternative for enabling a service-centric approach, the level of provided programmability is a restricting factor in the long term.

P4 is a language that provides a fully programmable environment, instead of the partial programmability brought by OpenFlow, which is the primary mechanism utilized for our initial solution for the service-centric model [12]. Due to its nonrestricted operations, P4 has proved itself a promising candidate to perform innovative operations within the network.

P4 (Programming Protocol-Independent Packet Processors) was proposed in 2015 by the P4 Consortium, in which the pioneers of SDN also have an essential role [119]. It promises the fully programmable data planes through a high-level language developed explicitly for defining the behaviors and characteristics of the networking nodes. With

P4, any protocol can be implemented from scratch, and how switches process each packet can be customized. By enhancing the degree of flexibility, P4 mitigates the limitations of SDN, enables protocol-independency and paves paves the road for vendor independence by eliminating the need for specific firmware updates or chip upgrades.

Table 3.2. Comparison of SDN/OpenFlow and P4.

Property	**SDN/OpenFlow**	**P4**
Proposer	Open Networking Foundation (ONF)	P4 Consortium (Common members with ONF)
Programmability	Partial	Full
Centralized Controller	Available	Available
Gathering Statistics	Available	Available
Populating Tables	Available	Available
Creating Custom Tables	Not Available	Available
Matching Fields	Fixed (40+ fields Layer2/3/4)	User-defined
Possible Actions	Fixed	User-defined
Topology Detection	Available	Not Available
Protocol-independency	Not Available	Available
Fault Detection	Available	Not Available

The recent version of OpenFlow (v1.5.1) [120] supports 40+ packet header fields (Layer 2/3/4) for the matching mechanism. Although this number was lower in the previous versions, there is still protocol-dependency that constructs a barrier for accelerating innovation. The limited progress in OpenFlow and restricted set of matching fields are the main obstacles to implement the SDN-based service-centric model. The capability of defining custom headers is the main argument for applying P4 to leverage service-centricness. P4 is not proposed as a replacement for OpenFlow, but it addresses the limitations in the SDN. While OpenFlow supports partial-programmability, P4 enhances the degree of flexibility and creates the opportunity to define customized behavior for each packet processor. Similar to the OpenFlow, P4 Runtime API provides functions to populate and organize the match/action tables. The detailed comparison of SDN/OpenFlow and P4 is summarized in Table 3.2. As observed, there are common features, but each technology is specialized in specific responsibilities and objectives.

4. SLA-AWARE OPTIMAL RESOURCE ALLOCATION

Each of the services provided by the multi-tier computing architecture (e.g., healthcare applications, infotainment through video streaming, vehicular communication and augmented reality) has its own set of requirements and characteristics. Maximum tolerable service delay, code complexity, generated load within the network, and the minimum required satisfaction level can be given concrete examples. Considering these properties, the network infrastructure and orchestration mechanism should treat them accordingly to comply with the SLA definitions. Within this context, a fine-tuned SLA-aware resource allocation and task assignment scheme is a key component to handle the service requests generated at the network edge.

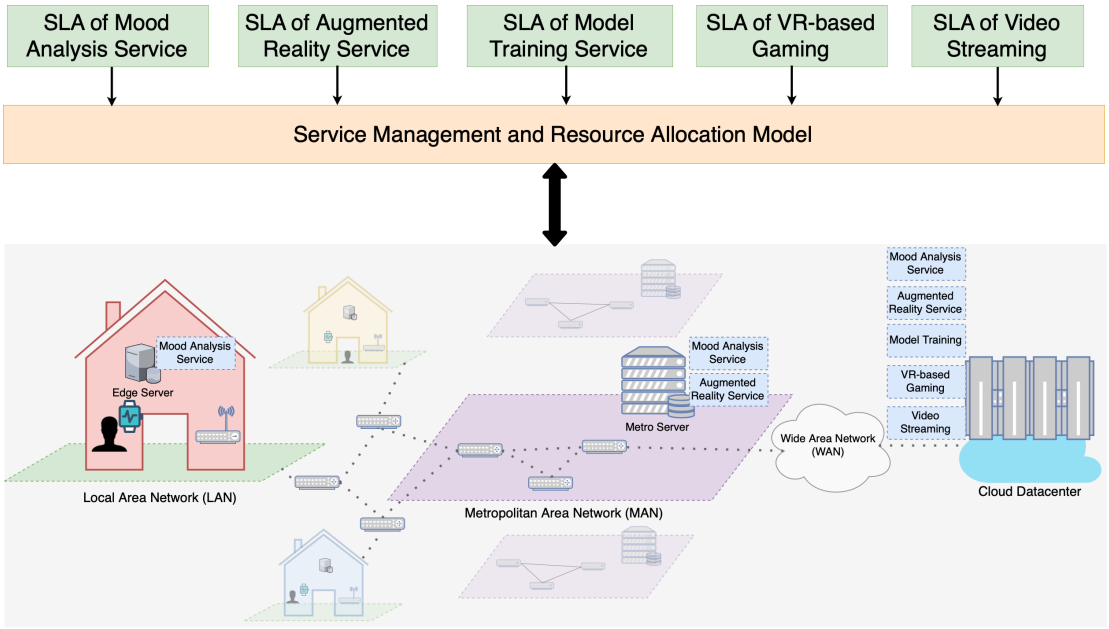

Figure 4.1. Service-oriented environment and SLA definitions of the services.

Correspondingly, this section introduces an SLA-aware optimal resource allocation scheme designed for the service-centric multi-tier computing architecture. Before providing the details of the proposed solution approaches, the main problem components within the multi-tier computing architecture based on service-oriented operations are defined.

4.1. Problem Definition

In a multi-tier service-centric environment as depicted in Figure 4.1, the main objective is to maximize the number of handled requests through an optimal resource allocation scheme while maintaining service quality above a predefined level. The resource allocation decisions have to be determined so that the delay constraints and other requirements of each service instance are met.

This thesis considers two different problem definitions based on the behavior of the services and the context of the SLAs:

- Undifferentiated set of services where each service type has the same priority
- Fairness among the services through a minimum satisfaction ratio imposed by SLAs.

While defining these problems, we assume that the following parameters are specified in advance:

- Locations of users, servers, and service instances
- Network topology
- Capacities of the network and computational resources
- Service attributes and requirements
- Average number of requests by each user for each service type

To model an SLA-aware service-centric environment, the network topology $G = (N, E)$ is assumed to be given in advance, where N and E represent the nodes and the links connecting the nodes, respectively. The capacity of each networking node $n \in N$ is denoted by c_n, and the capacity of each link $\{i, j\} \in E$ is represented by a_{ij}, in terms of Mbps. Similarly, the computation infrastructure capabilities, service attributes, and requirements are considered input parameters for the model. The set S denotes the servers throughout the network, and Q represents the set of service types with different

requirements. The capacity of each server $s \in S$ is denoted by b_s in terms of million instructions per second (MIPS). A binary parameter v_{qs} takes value 1 if service type q is deployed on server s, and 0 otherwise.

Table 4.1. Set and parameter notations.

Definition	**Notation**
Network topology	$G = (N, E)$
Networking nodes	$N = \{n_1, n_2, n_3, ...\}$
Networking links	$E = \{e_{12}, e_{13}, e_{14}, ...\}$
End-users	$U = \{u_1, u_2, u_3, ...\}$
Servers	$S = \{s_1, s_2, s_3, ...\}$
Service types	$Q = \{q_1, q_2, q_3, ...\}$
Server capacity	b_s
Node capacity	c_n
Link capacity	a_{ij}
Service deployment	v_{qs}
Computational load of service type q	m_q
Network load of service type q	l_q^{req}, l_q^{res}
Maximum allowed delay for service type q	α_q
Traffic requirements	r_{uq}

Each service may demand a particular capacity of both networking and computational resources. The expected number of instructions (in millions) to be executed for service type $q \in Q$ is represented by m_q. On the other hand, the service requests and corresponding responses may contribute to the network load in different amounts. A service type q request and its corresponding response generate an amount of load equal to l_q^{req} and l_q^{res} on the average in terms of Mbit, respectively. The user-centric services deployed over the multi-tier system are latency intolerant. Therefore, their SLAs assert a maximum acceptable delay to be satisfied. The upper limit of the overall latency for service type q, maximum acceptable end-to-end latency value is denoted as α_q in seconds which consists of networking and code execution operations.

Finally, the set U refers to the end-users in the system and the average number of offloading operations requested by user u for service type q in a unit time interval is denoted by r_{uq}. All index sets and parameters that are utilized in the following models are summarized in Table 4.1.

4.2. MINLP Model Formulation

4.2.1. Model Formulation for Undifferentiated Services

The first optimization problem consists of various services with different characteristics deployed throughout the multi-tier computing architecture. The services are non-prioritized, and fairness is not of concern. Remarkably, the network and computing infrastructure does not commit privilege to any service in terms of resource allocation.

The primary goal of the problem is to handle the service requests within the specified delay constraints. Therefore, the objective function is to maximize the total number of accomplished service requests, which can be formulated as

$$\max z = \sum_{u} \sum_{q} r_{uq} \theta_{uq} \tag{4.1}$$

$$\theta_{uq} = \begin{cases} 1, & \text{if offloading services of type } q \\ & \text{by user } u \text{ is successfully handled} \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

where θ_{uq} is a binary variable that represents the status of the offloaded task for service type q demanded by user u. If requests of user u are successfully handled by the multi-tier system, $\theta_{uq} = 1$, otherwise, $\theta_{uq} = 0$ indicating that requests are discarded.

Since the service-centric environment aims to allocate the most feasible resources for the requests, it is essential to analyze the task assignments. In order to enhance the resolution, the objective function and the related constraints can be written as

$$\max z = \sum_{u} \sum_{q} \sum_{s} r_{uq} \theta_{uqs} \tag{4.3}$$

$$s.t. \sum_{s} \theta_{uqs} \leq 1 \qquad \forall u, q \tag{4.4}$$

$$\theta_{uqs} \leq v_{qs} \qquad \forall u, q, s \tag{4.5}$$

$$\theta_{uqs} \in \{0, 1\} \qquad \forall u, q, s \tag{4.6}$$

where θ_{uqs} is a binary variable representing whether the request for service type q demanded by user u is handled by server s within the multi-tier system. Therefore, the objective function can be formulated as maximizing the number of satisfied requests considering all users, service types, and servers. In constraints (4.4), it is defined that a request can be handled by at most one server and constraints (4.5) indicate that the assignment operation can be valid if the requested service type is deployed on the server. Constraints (4.6) enforce that θ_{uqs} is a binary decision variable.

Since most user-centric services are delay-sensitive, the end-to-end delay experienced by a task offload operation should be lower than the acceptable delay value specified by the SLA of the corresponding service type. This can be expressed as

$$\tau_{uqs} \leq \alpha_q \qquad \forall u, q \tag{4.7}$$

where τ_{uqs} denotes the expected end-to-end latency of a type q service request by user u when handled by server s. When the life-cycle of a service execution is analyzed, we can observe that the following actions contribute to the overall delay:

(i) Routing the service request from the end-user location to the server-side
(ii) Executing the service code
(iii) Routing the response of the execution as output from the server back to the end-user device

Both networking and computational resources are considered while calculating the overall delay, including the operations mentioned above. Within this context, traditional queuing models represent the behavior of resources in terms of processing. Once the task assignment decision and the corresponding server is determined through the θ_{uqs} variable, the path with the minimum number of hops between the user and the target server is chosen for routing. On this shortest path, the delay contributed by the network nodes is calculated through the $M/M/1$ queuing model. However, the optimization model is flexible enough to incorporate other delay models if necessary. Let $N_{us} \in N$ be the set of nodes on the min-hop route between the location of user u and the location of server s. The expected networking delay of routing the service type q request from the end-user to the destination server can be calculated as

$$\sum_{n \in N_{us}} \frac{l_q^{req}}{c_n - \sum\limits_{q} \sum\limits_{\substack{(u,s): \\ n \in N_{us}}} r_{uq}(l_q^{req} + l_q^{res})\theta_{uqs}} \quad . \tag{4.8}$$

Since the request and the response of an offloading operation are assumed to follow the same route in reverse order, the network delay for forwarding the response back to the end-user is calculated similarly as

$$\sum_{n \in N_{us}} \frac{l_q^{res}}{c_n - \sum\limits_{q} \sum\limits_{\substack{(u,s): \\ n \in N_{us}}} r_{uq}(l_q^{req} + l_q^{res})\theta_{uqs}} \quad . \tag{4.9}$$

Therefore, the overall delay produced by the networking operations as part of task offloading can be expressed as

$$E(T_{node}^q) = \sum_{n \in N_{us}} \frac{l_q^{req} + l_q^{res}}{c_n - \sum\limits_{q} \sum\limits_{\substack{(u,s): \\ n \in N_{us}}} r_{uq}(l_q^{req} + l_q^{res})\theta_{uqs}} \tag{4.10}$$

where the latency is directly affected by the capacity c_n of the forwarding device and the current load on that particular device. The load caused by adjacent operations

within the network is determined by the θ_{uqs} decision variable if the corresponding node is on the same path between different source-destination pairs.

The data transmission delay through the links connecting the nodes on the shortest path depends on the network load requirement of the service and the capacity of the link. The overall delay for both request and response forwarding operations through the links is calculated as

$$E(T^q_{link}) = \sum_{(i,j)\in E_{us}} \frac{(l^{req}_q + l^{res}_q)}{a_{ij}} \tag{4.11}$$

where $E_{us} \in E$ represents the set of links on the shortest path between user u and server s.

In addition to the communication latency, the service execution latency is also considered while meeting the upper limit of the acceptable delay specified for a service. Like networking resources, a traditional queuing model is utilized to calculate the expected code execution delay at the server-side. Thus, the expected service execution time, which is denoted as $E(T_{qs})$, is calculated as

$$E(T_{qs}) = \frac{m_q}{b_s - \sum_u \sum_q r_{uq} m_q \theta_{uqs}} \tag{4.12}$$

where the delay is affected by the service computation load requirements, the capacity of the server, and the adjacent executions on the same server. Hence, the end-to-end delay constraints in (4.7) can be rewritten as

$$\sum_s \left(E(T_{qs}) + E(T^q_{node}) + E(T^q_{link})\right) \theta_{uqs} \leq \alpha_q \qquad \forall u, q\ , \tag{4.13}$$

which states that the total delay of code execution, switching, and link transmission of an offloaded service should not exceed the delay limit specified by the SLA.

For an $M/M/1$ queuing system, as the utilization of a resource increases and reaches a point close to one, the queue length explodes, the delay increases indefinitely, and the system becomes unstable. In order to prevent excessive latency values, the maximum allowed utilization for computational resources is set to ϕ, and for networking resources it is set to φ, where $0 < \phi < 1$ and $0 < \varphi < 1$. In order to ensure that the average utilization of servers and nodes does not exceed this limit, the following constraints are added to the model:

$$\sum_{q} \sum_{\substack{(u,s): \\ n \in N_{us}}} r_{uq}(l_q^{req} + l_q^{res})\theta_{uqs} \leq \varphi c_n \qquad \forall n \qquad (4.14)$$

$$\sum_{u} \sum_{q} r_{uq} m_q \theta_{uqs} \leq \phi b_s \qquad \forall s \ . \qquad (4.15)$$

In constraints (4.14), for all nodes within the network, it is asserted that the mean utilization of a node should be at most ψ. The same limitation is defined for all servers in the multi-tier system in constraints (4.15).

4.2.2. Service Fairness Problem

In the previous problem, even though the services require different loads on computational and network resources and impose specific delay limits, each service assignment is treated equally by the objective function of the problem. In other words, each service execution has the same reward in the overall objective regardless of the service type. However, the previous formulations may result in solutions with unbalanced satisfaction ratios among services to increase the total number of executed tasks.

A more realistic problem definition can be obtained by enforcing additional requirements on resource allocations and task assignments. The SLA definitions may impose not only the delay conditions but also further regulations. The third-party application developers or service providers may specify a minimum ratio of user requests to be satisfied. The objective function and constraints formulated for undifferentiated

services problem do not consider this restriction. The services that are more latency-intolerant or demanding than the others may have secondary importance since it is more challenging to allocate resources optimally.

In order to provide fairness among services, let δ_q denote the minimum required ratio of handled requests to overall requests generated by the users for service type q. For incorporating this modification in the optimization model, we can add the following constraints while the objective function and the rest of the constraints remain the same:

$$\sum_{u} \sum_{s} r_{uq} \theta_{uqs} \geq \delta_q \sum_{u} r_{uq} \qquad \forall q \ . \tag{4.16}$$

These constraints ensure that the number of successful task assignments cannot be lower than the minimum level for each service. The integration of these constraints provides a customized behavior for each service type to achieve a certain level of customer satisfaction.

4.2.3. Linearization of the Non-linear Model

In the previous formulations, θ_{uqs} variables are defined as binary variables, and the delay constraints in (4.13) are nonlinear. Therefore, it is an MINLP model, which combines the challenges of handling nonlinearities with the combinatorial explosion of integer variables.

For providing an alternative approach and solve the problem for which the MINLP model is inadequate to find a good solution, the delay function obtained by the formulations of the $M/M/1$ queuing model can be replaced by a piecewise linear approximation. For computational and networking resources within the network, the utilization-delay curve can be divided into four subsections, as in Figure 4.2. While formulating the delay function, the points where the slope changes are referred to as breakpoints. Let v_1, v_2, v_3, and v_4 denote the four breakpoints along the utilization axis in Figure 4.2,

and let $T(v_1), T(v_2), T(v_3)$, and $T(v_4)$ denote the corresponding latency values that can be calculated beforehand. Utilization levels between v_1 and v_2 are considered as low load traffic. This portion of the delay curve can be approximated by a line, as shown in the figure. Similarly, utilization levels between v_2 and v_3 are considered medium load traffic, and levels between v_3 and v_4, where the exponential increase in the delay function is more noticeable, form high-load traffic. Separate lines also approximate these segments. A utilization beyond v_4 is not allowed for servers and nodes to prevent excessive latency values as in the nonlinear case.

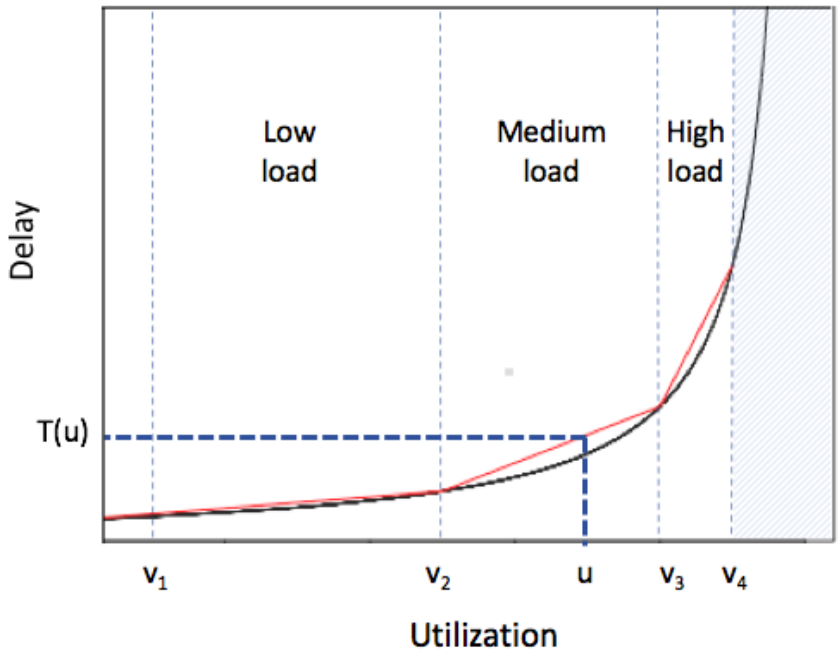

Figure 4.2. Linearization of utilization-delay curve.

It is worth noting that any utilization point u between v_1 and v_4 can be expressed as a weighted sum of breakpoints. Let $\lambda_1, \lambda_2, \lambda_3$, and λ_4 denote four nonnegative weights such that their sum is one. Then, the piecewise linear approximation of the delay function can be written as

$$\lambda_1 T(v_1) + \lambda_2 T(v_2) + \lambda_3 T(v_3) + \lambda_4 T(v_4) = T(u) \tag{4.17}$$

$$\lambda_1 v_1 + \lambda_2 v_2 + \lambda_3 v_3 + \lambda_4 v_4 = u \tag{4.18}$$

$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1 \tag{4.19}$$

with the requirement that at most two adjacent λ's can be nonzero. To ensure this, we defined new binary variables x_1, x_2, x_3 and added the following constraints:

$$\lambda_1 \leq x_1, \tag{4.20}$$

$$\lambda_k \leq x_{k-1} + x_k \quad k = 2, 3, \tag{4.21}$$

$$\lambda_4 \leq x_3, \tag{4.22}$$

$$x_1 + x_2 + x_3 = 1. \tag{4.23}$$

With this transformation, we can eliminate the nonlinearity of the utilization-delay curve, convert the overall problem into an MILP problem. In this method, the number of breakpoints can be increased to obtain more accurate results, which may create a computational overhead.

Since the $M/M/1$ delay calculation is a convex function, the linear approximation may overestimate the real delay value. So, it guarantees that any feasible solution obtained by the linear approximation is also feasible for the original nonlinear model.

To apply this idea in calculating the expected delay contributed by the servers and nodes, the utilization values can be calculated as

$$\Upsilon_s = \frac{\sum_u \sum_q r_{uq} m_q \theta_{uqs}}{b_s} \qquad \forall s \tag{4.24}$$

$$\Upsilon_n = \frac{\sum_q \sum_{(u,s):n \in N_{us}} r_{uq}(l_q^{req} + l_q^{res})\theta_{uqs}}{c_n} \qquad \forall n \tag{4.25}$$

where Υ_s and Υ_n denote utilizations for server s and node n, respectively. Then, the expected code execution delay on server for service type q, represented by $E(T_{qs})$ in equality (4.12), and the expected networking delay on nodes for service type q, denoted by $E(T_{node}^q)$ in equality (4.10) can be replaced by a weighted sum as in equality (4.17) with the additional constraints (4.18-4.23). Hence, the nonlinear end-to-end delay

constraints (4.13) can be replaced by a linear function and the overall problem can be converted into an MILP model. It embraces the same objective function and constraints with the previously presented MINLP model, except the linearized delay formulation. Thereby, the time complexity of the MINLP model can be reduced.

4.3. Nearest-Fit Heuristic Algorithm

In addition to optimization model proposals, a heuristic algorithm is provided to address the time complexity of the undifferentiated services problem. The algorithm can be used to find a feasible solution quickly, which is beneficial for setting a lower bound for the optimization models. It is a greedy approximation algorithm similar to those used to solve the bin packing problem.

The algorithm initially reads the input parameters, such as topology, service characteristics, set of users and servers, and the average number of tasks offloaded by each user. The algorithm starts with the most latency-intolerant service type. For each request targeting that service type, it checks whether there is enough capacity for execution. It attempts to allocate resources for the request on the nearest server by considering the capacity constraints. If the task assignment is not feasible, it continues to check the following servers based on the distance regarding the number of hops.

The task assignment feasibility is maintained if that service is already deployed on the server and the utilization of the target server or any node on the shortest path does not exceed the parameters ϕ and φ after the current task assignment. The overall delay of a request is calculated using formulations of the $M/M/1$ queuing model. If it satisfies the minimum delay requirement, the request can be assigned to that server. However, solutions obtained in this way may violate end-to-end delay constraints of the requests that are already assigned. Therefore, an additional procedure checks whether the latency requirements are still met for the previously assigned requests. If any of them are violated, the algorithm does not make the new assignment and continues to check the subsequent server.

```
foreach q ∈ Q, u ∈ U do
    sortDistance(S, u);
    foreach s ∈ S do
        totalDelay ← 0;
        if v_qs = 0 or serverUtil(s) > φ then
            return false;
        end
        totalDelay+ = delay(q, s);
        foreach n ∈ shortestPath(u, s), (i, j) ∈ shortestPath(u, s) do
            totalDelay+ = delay(q, n);
            totalDelay+ = delay(q, i, j);
            if nodeUtil(n) > φ or totalDelay > α_q then
                return false;
            end
        end
        if totalDelay >α_q then
            return false;
        end
        θ_uqs =← 1;
        foreach assigned request with θ_uqs = 1 do
            newDelay ← calculateNewDelay(u, q, s);
            if newDelay > α_q then
                return false;
            end
        end
    end
end
```

Figure 4.3. Nearest-fit heuristic implementation.

The algorithm iterates over the set of users until the last request for the service type that is the most latency-intolerant. Then, the same process is executed for the

second service type in the set until all requests are inspected. The algorithm then terminates and generates the results of the resource allocations as the output. The details of the heuristic implementation and execution steps are summarized in Algorithm 4.3.

Table 4.2. Topology design and parameters.

Topology Size	**Number of Users**	**Number of Nodes**	**Number of Edge Servers**	**Number of Metro Servers**	**Number of Cloud Servers**
Small Topology (Abilene)	360	11	12	3	7
Medium Topology (AboveNet)	510	18	24	9	9
Large Topology (ATT)	630	25	28	15	14

The main objective of the proposed solution approaches is to create an action plan for the operators in the long run to optimize the policy of orchestration procedures. Therefore, it does not address the requirements of a highly dynamic edge environment, which is not within the scope of the thesis. Legacy solution methodologies may remain infeasible for the problem definition of dynamic task offloading considering concrete use case examples such as vehicular communication or massive IoT deployments in 5G systems. In order to take handover operations into account for capturing the effect of mobility at the edge, intelligent orchestration systems using AI/ML methodologies can be integrated [121].

4.4. Performance Evaluation

The performance of the optimization models and the heuristic algorithm is evaluated on the undifferentiated services problem and the service fairness problem. Before going into the details of the numerical results and discussion of the performances of the solution approaches, the experimental design and use cases are introduced.

4.4.1. Experimental Design

In this study, the proposed methods are evaluated under randomly generated instances with different topologies. Each topology varies concerning the number of nodes, users, and servers. Three different connected real-world topologies from Topology Zoo [122] are utilized for assessing the performance of the optimization models and the heuristic algorithm: small, medium and large-sized topologies. The attributes of each topology and the corresponding parameters are displayed in Table 4.2. Also, the design of the large topology is shown in Figure 4.4 to give an idea about the network environment of the use cases.

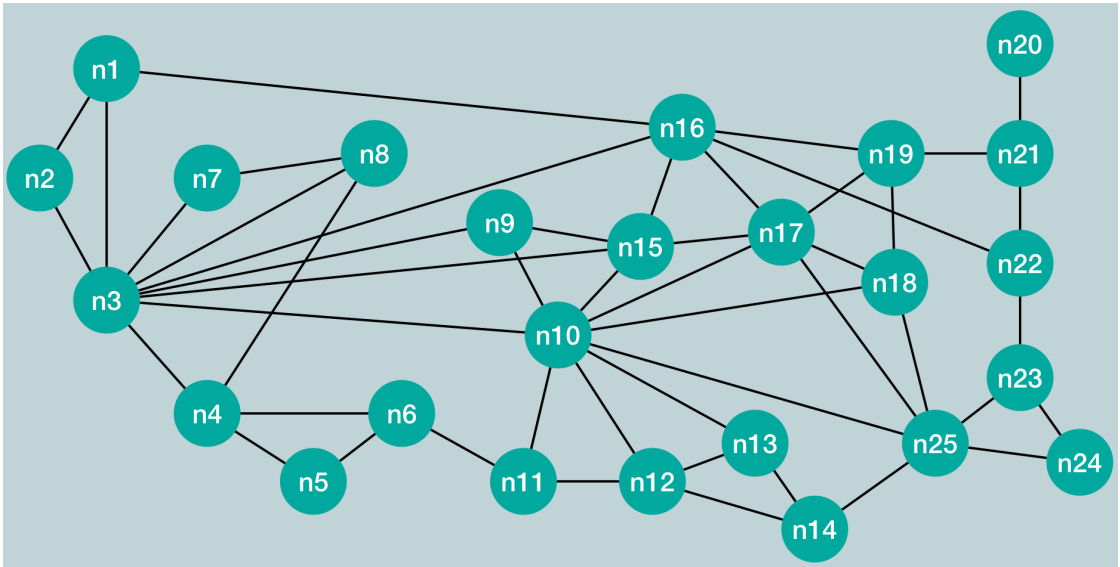

Figure 4.4. The map for large topology (ATT).

For each topology, three different load conditions are produced with respect to the average number of service requests in a unit time, as summarized in Table 4.3. Each end-user generates one service request/second on average when the network is lowly loaded. In the high load cases, some of the end-users generate two requests/second on average. Lastly, the medium load cases are configured as an intermediate.

In order to observe the effects of services with different characteristics in detail, four different service types are determined. While creating the service types and regulating the requirements of each type, the use cases envisioned by ETSI for MEC and

5G-PPP for the next-generation cellular networks are adapted. ETSI [123] states that video streaming, augmented reality, and IoT services are example scenarios that are possible to be served by the MEC servers. Besides, 5G-PPP [124] defines a set of services composed of healthcare and entertainment applications. In order to reflect the characteristics of such service types, their requirements are determined accordingly. It is predicted that although service-oriented networks with multi-tier computing systems are not implemented in real environments yet, future deployments will use these services as main use case scenarios. The resource requirements and the latency constraints of these services are shown in Table 4.4.

Table 4.3. Traffic requirements (requests/second).

Topology	**Low Load**	**Medium Load**	**High Load**
Small Topology	360	499	543
Medium Topology	510	664	747
Large Topology	630	854	951

Computational load requirements on servers denoted as m_q are given in terms of million instructions (MI) to be executed, and the networking resource demand of each service request and response is presented as l_q^{req} and l_q^{res}, respectively, in terms of Mbits. Lastly, the maximum acceptable delay values imposed by the SLA definitions denoted as α_q are given in seconds.

Table 4.4. Services and their attributes.

Service Name	**Computation Load (m_q in MI)**	**Network Load for Request (l_q^{req} in Mbits)**	**Network Load for Response (l_q^{res} in Mbits)**	**Max. Delay (α_q in sec)**
Face Recognition (q_1)	500	20	20	0.5
Fall Risk Assessment (q_2)	1000	30	10	1.5
Augmented Reality (q_3)	1500	40	40	1.0
Video Streaming (q_4)	2000	50	50	2.0

Throughout the use cases, three different levels in the multi-tier system are considered. The capacity parameter b_s is configured as 15,000 MIPS for the edge servers, 30,000 MIPS for the metro servers, and 80,000 MIPS for the cloud servers. Each edge server is randomly deployed with two service types, while metro servers provide three, and cloud servers provide all four of the service types.

In addition to the capacities of the computational resources, while the links have an identical capacity of 10 Gbps, the switches close to the end-users provide 10 Gbps processing capacity. The switches providing the connection to the cloud servers have 1 Gbps capacity.

For experimental purposes, the maximum utilization parameters ϕ and φ is set to 0.9 so that at most 90% of the resource capacity can be utilized by servers and switches. Correspondingly, the utilization breakpoints v_1, v_2, v_3, and v_4 defined for linear approximation approach are set as 0, 0.5, 0.8, and 0.9, respectively.

The MINLP and MILP formulations are executed using General Algebraic Modeling System (GAMS 25.1.3) [125] running on a computer with Intel Xeon E5-2690 2.6 GHz CPU and 64GB main memory. For MINLP models, SCIP [126, 127] is used as an optimization solver, whereas Gurobi [58] is utilized for MILP models. Lastly, the heuristic algorithm is implemented in C++ programming language, which also utilizes Gurobi for finding the shortest paths between each source-destination pair. For optimization solvers, three hours of run-time is set as the time limit for each problem size. The following information is reported for each topology and solution approach:

- The total number of service requests in the system,
- The objective value of the best feasible solution obtained by the solution method within the given time limit denoted as Z_{method},
- An upper bound for the optimal objective value obtained by the MINLP model within the given time limit, denoted as Z_{UB},
- The minimum optimality gap obtained at the end of the given time limit for the

MINLP model (calculated as $\frac{Z_{UB}-Z_{MINLP}}{Z_{MINLP}}$),

- The percentage difference between the objective value of the best feasible solution obtained by the solution method and the upper bound for the objective value obtained by the MINLP model (calculated as $\frac{Z_{UB}-Z_{method}}{Z_{method}}$),
- The amount of time spent by each method in seconds,
- The ratio of successfully handled requests to the total number of requests for the overall system and for each service type.

The results of the computational study and the discussion of the performances of the solution approaches are presented in the following subsections for the undifferentiated services and service fairness problems.

Table 4.5. MINLP model for undifferentiated services problem.

Topology Type	**Load Condition**	**Total Requests**	Z_{MINLP}	Z_{UB}	**Optimality Gap**	**Time (sec)**	**Overall Satisfaction Ratio**	**Service Specific Satisfaction Ratio** (q_1-q_2-q_3-q_4)
Small Topology	Low Load	360	360	360	0%	207	100%	100%-100%-100%-100%
	Medium Load	499	472	476	0.85%	10800	95%	100%-100%-99%-80%
	High Load	543	495	501	1.21%	10800	91%	100%-98%-97%-70%
Medium Topology	Low Load	510	510	510	0%	3977	100%	100%-100%-100%-100%
	Medium Load	664	642	657	2.34%	10800	97%	97%-100%-97%-92%
	High Load	747	661	683	3.33%	10800	88%	100%-99%-95%-66%
Large Topology	Low Load	630	630	630	0%	5412	100%	100%-100%-100%-100%
	Medium Load	854	782	806	3.07%	10800	92%	99%-99%-96%-72%
	High Load	951	828	848	2.42%	10800	87%	100%-99%-95%-55%

4.4.2. Results for Undifferentiated Services Problem

For the undifferentiated services problem, the results obtained by the MINLP model are provided in Table 4.5. The MINLP model can find the optimal solution for cases where the network is low-loaded. In these cases, all user requests are satisfied by the system. As the topology size increases, it takes more time to find the optimal solution. Moreover, as the total number of requests increases for each topology size, more tasks can be assigned, but the overall satisfaction ratio decreases. Although the MINLP model can find a good feasible solution with at most 3.33% gap, it cannot guarantee the optimality of the current solution within the given time limit for medium and high load conditions in every topology.

In addition, since each service type is identical regarding contribution to the objective, the satisfaction ratio of the video streaming service, which demands the highest resource among all, is lower than the others, and it decreases even further when the network becomes highly loaded. Since the model aims to maximize the total number of successfully handled offload operations, it greedily accepts the services demanding relatively less resources. Therefore, while requests for the other services are satisfied with at least 95%, the video streaming service users suffer from the deteriorated performance, especially when the network is highly loaded.

Similar results are obtained for the MILP model, as presented in Table 4.6. Since the delay function is linearized, the complexity of the problem is reduced, and so the model can achieve optimality for low load cases in a shorter amount of time. It can also provide slightly better feasible solutions with a higher satisfaction ratio for medium and high load cases. However, the difference is not significant compared to the MINLP model. At the same time, the optimality gap is decreased because of the reduced problem complexity. It is observed that the proposed models can find either optimal or near-optimal solutions for the undifferentiated services problem with at most 3.33% optimality gap. It should be reported that the MILP model can find a good feasible solution in a short while, but it tries to minimize the optimality gap further during the

allowed run time of three hours. Like the MINLP model, video streaming service users suffer from low satisfaction while the other service requests are completely satisfied.

Table 4.6. MILP model for undifferentiated services problem.

Topology Type	**Load Condition**	**Total Requests**	Z_{MILP}	Z_{UB}	**Difference from Z_{UB}**	**Time (sec)**	**Overall Satisfaction Ratio**	**Service Specific Satisfaction Ratio (q_1-q_2-q_3-q_4)**
Small Topology	Low Load	360	360	360	0%	46.1	100%	100%-100%-100%-100%
	Medium Load	499	474	476	0.42%	10800	95%	100%-100%-100%-80%
	High Load	543	499	501	0.40%	10800	92%	100%-100%-100%-68%
Medium Topology	Low Load	510	510	510	0%	119.7	100%	100%-100%-100%-100%
	Medium Load	664	653	657	0.61%	10800	98%	100%-100%-100%-93%
	High Load	747	679	683	0.59%	10800	91%	100%-100%-100%-69%
Large Topology	Low Load	630	630	630	0%	861	100%	100%-100%-100%-100%
	Medium Load	854	796	806	1.26%	10800	93%	100%-100%-100%-72%
	High Load	951	840	848	0.95%	10800	88%	100%-100%-100%-54%

In order to reduce the complexity of the problem, users are considered aggregated demand points. For each topology design, the users are aggregated concerning their locations. The average number of requests generated from a user location is the sum of all individual user requests within the same location as if a single aggregated user generates the requests. In Tables 4.7 and 4.8, the results of both formulations are shown for the aggregate users case. The upper bound values for the objective function given in these tables are taken from the MINLP model where the users are not aggregated, and the difference is calculated accordingly. When we compare the equivalent cases

of separate and aggregate user cases, it can be easily seen that the overall satisfaction ratio decreases, even though the complexity is lowered. Optimal solutions could be found in a much shorter time. By aggregating the users based on their locations, the search space is narrowed to achieve the solutions easily. However, since the requests from the same location are considered a batch, and the assignment variables are binary, handling the requests on different servers becomes impossible. Consequently, there is a trade-off between the solution time and the satisfaction ratios of the service types. The aggregation of end-users and service requests can still be beneficial to find a feasible solution when the network is even larger and the number of users is very high.

Table 4.7. MINLP model for undifferentiated services problem with aggregate users.

Topology Type	**Load Condition**	**Total Requests**	$Z_{MINLP,Agg}$	Z_{UB}	**Difference from Z_{UB}**	**Time (sec)**	**Overall Satisfaction Ratio**	**Service Specific Satisfaction Ratio (q_1-q_2-q_3-q_4)**
Small Topology	Low Load	360	360	360	0%	5	100%	100%-100%-100%-100%
	Medium Load	499	441	476	7.94%	2	88%	100%-100%-100%-54%
	High Load	543	453	501	10.6%	5	83%	100%-100%-94%-41%
Medium Topology	Low Load	510	462	510	10.39%	15	91%	100%-100%-90%-76%
	Medium Load	664	515	657	27.57%	718	78%	100%-100%-65%-45%
	High Load	747	579	683	17.96%	7	78%	100%-100%-79%-41%
Large Topology	Low Load	630	570	630	10.53%	104	90%	100%-100%-88%-74%
	Medium Load	854	696	806	15.8%	612	81%	100%-100%-93%-31%
	High Load	951	693	848	22.37%	273	73%	100%-100%-65%-25%

In order to create a benchmark for the proposed optimization models, the performance of the nearest-fit heuristic algorithm is also evaluated for the undifferentiated

services problem, as shown in Table 4.9. In order to assess the efficiency of the heuristic algorithm more accurately, the difference between the objective values obtained by the heuristic algorithm and the optimization models is presented.

Table 4.8. MILP model for undifferentiated services problem with aggregate users.

Topology Type	**Load Condition**	**Total Requests**	$Z_{MILP,Agg}$	Z_{UB}	**Difference from Z_{UB}**	**Time (sec)**	**Overall Satisfaction Ratio**	**Service Specific Satisfaction Ratio (q_1-q_2-q_3-q_4)**
Small Topology	Low Load	360	360	360	0%	1.2	100%	100%-100%-100%-100%
	Medium Load	499	441	476	7.94%	1.1	88%	100%-100%-100%-54%
	High Load	543	453	501	10.6%	1.6	83%	100%-100%-94%-41%
Medium Topology	Low Load	510	462	510	10.39%	17.3	91%	100%-100%-90%-76%
	Medium Load	664	515	657	27.57%	30.3	78%	100%-100%-66%-44%
	High Load	747	579	683	17.96%	3.6	78%	100%-100%-84%-36%
Large Topology	Low Load	630	570	630	10.53%	33.2	90%	100%-100%-88%-74%
	Medium Load	854	696	806	15.8%	21.8	81%	100%-100%-93%-31%
	High Load	951	693	848	22.37%	16.6	73%	100%-100%-65%-25%

The results indicate that the heuristic approach can find a feasible solution for all instances quickly by making near-optimal task assignment and resource allocation decisions. The maximum difference between objective values of the heuristic algorithm and the optimization models is 7.42%. Since no related study in the literature addresses the exact problem definition with the same objective function and constraints, the nearest-fit heuristic algorithm can be considered a baseline. As a result of its performance in terms of the optimality gap and the run-time, it can be concluded that the nearest-fit heuristic algorithm is a good candidate for assessing the performance of the

proposed optimization models. Besides, it outperforms the user aggregation approach by significantly improving the overall satisfaction ratio within a shorter time. The performance of the proposed heuristic algorithm demonstrates that it can be a promising alternative to find feasible solutions for even more complex network designs.

Table 4.9. Heuristic implementation for undifferentiated services problem.

Topology Type	**Load Condition**	**Total Requests**	$Z_{Heuristic}$	**Difference from MINLP**	**Difference from MILP**	**Time (sec)**	**Overall Satisfaction Ratio**	**Service Specific Satisfaction Ratio (q_1-q_2-q_3-q_4)**
Small Topology	Low Load	360	359	0.28%	0.28%	2.4	100%	100%-100%-100%-99%
	Medium Load	499	446	5.83%	6.28%	3.7	89%	100%-100%-100%-58%
	High Load	543	473	4.65%	5.50%	4.0	87%	100%-100%-100%-49%
Medium Topology	Low Load	510	510	0%	0%	4.9	100%	100%-100%-100%-100%
	Medium Load	664	632	1.58%	3.32%	7.9	95%	100%-100%-100%-79%
	High Load	747	653	1.23%	3.98%	11.5	87%	100%-100%-100%-58%
Large Topology	Low Load	630	625	0.80%	0.80%	18.3	99%	100%-100%-100%-97%
	Medium Load	854	756	3.44%	5.29%	28.9	89%	100%-98%-100%-54%
	High Load	951	782	5.88%	7.42%	33.8	82%	100%-87%-97%-47%

4.4.3. Results for Service Fairness Problem

In order to maintain fairness among different service types and find a solution to the low satisfaction ratio problem for video streaming service, additional constraints forcing a minimum satisfaction ratio for all service types are integrated into the op-

timization models. Since all of the offload operations are successfully handled by the multi-tier architecture when the load within the network is low for each topology design, only medium and high load cases are examined, in which the undifferentiated services problem leads to an unfair solution in terms of service-specific satisfaction ratios. The minimum satisfaction ratio for each service type is set as 90% for medium load cases (except the medium topology, which is set as 95% since, in the solution for the undifferentiated services problem, all service types already satisfy the 90% satisfaction ratio requirement) and 80% for high load cases.

The results obtained by integrating the minimum satisfaction ratio constraints into the MINLP model are presented in Table 4.10. For small topology instances, the solutions satisfy the minimum satisfaction ratio requirement for each service type. The satisfaction ratio of video streaming service increases from 80% to 91%, and from 70% to 81% for medium and high load cases, respectively. However, the upper bound values for the objective function decrease compared to the undifferentiated services problem. It is because new constraints restrict the feasible region, and some solutions become infeasible. Moreover, the objective value of the best feasible solution and the overall satisfaction ratio decreases slightly. The leading cause of this situation can be explained as follows: In order to increase the number of satisfied requests for video streaming service by one, a higher number of requests for the lighter service types should be neglected for possible allocation. Since the problem becomes more complex with the minimum satisfaction ratio constraints, the MINLP model is unable to find feasible solutions for medium and large topologies within the allowed time limit.

The same experiments are also carried out for the MILP model. In Table 4.11, it can be observed that the model can find near-optimal solutions for small and medium topologies with at most 0.85% difference from the upper bound. However, a feasible solution that maintains service fairness could not be extracted for large topology instances. Compared to the MINLP model, it can be concluded that the obtained objective values are very close, and there are minor differences among the service-specific ratios for the small topology instances.

Table 4.10. MINLP model for service fairness problem.

Topology Type	**Load Condition**	**Min. Sat. Ratio**	Z_{MINLP}	Z_{UB}	**Optimality Gap**	**Time (sec)**	**Overall Satisfaction Ratio**	**Service Specific Satisfaction Ratio** (q_1-q_2-q_3-q_4)
Small Topology	Medium Load	90%	466	473	1.5%	10800	93%	99%-94%-90%-91%
	High Load	80%	491	497	1.22%	10800	90%	100%-100%-84%-81%
Medium Topology	Medium Load	95%	-	657	-	10800	-	-
	High Load	80%	-	679	-	10800	-	-
Large Topology	Medium Load	90%	-	784	-	10800	-	-
	High Load	80%	-	811	-	10800	-	-

Integrating the minimum satisfaction ratio for service fairness constraints makes the problem even more complex than the undifferentiated services problem. Hence, the optimization models become inadequate to find feasible solutions for higher-scale network environments. The nearest-fit heuristic algorithm used to find feasible solutions for the undifferentiated services problem cannot remedy the service fairness problem. The main reason is that the newly added constraints narrow the feasible region very much, and the task assignment decisions should be made considering the interactions between different assignments comprehensively. However, the nearest-fit heuristic algorithm performs a pass through the requests once, and when it assigns a request, it never checks and reassigns the request to create a room for the subsequent requests.

A matheuristic approach that combines metaheuristics and mathematical programming techniques can be adopted to attain a feasible solution for larger instances. The operations carried within the scope of a matheuristic consist of two phases. In the first phase, the minimum satisfaction ratio constraints are reversed (i.e., reversing the inequality) so that the optimization models can find a task assignment where each service type is satisfied by the specified maximum ratio. Since the objective is to maximize successfully handled requests, all service-specific satisfaction ratios are close to

the minimum required level in the solution obtained at the end of this phase. This solution can be converted into a feasible solution concerning the service fairness problem by adding a few new task assignments. In the second phase, the task assignment variables of services requiring heavy load are fixed since they are depreciated the most when fairness among services is not enforced. Then, the optimization model is solved for the remaining task assignment variables. Fixing some of the task assignment variables can reduce the problem complexity, so that a feasible solution can be obtained at the end of the second phase. However, the optimality is not guaranteed.

Table 4.11. MILP model for service fairness problem.

Topology Type	**Load Condition**	**Min. Sat. Ratio**	Z_{MILP}	Z_{UB}	**Difference from Z_{UB}**	**Time (sec)**	**Overall Satisfaction Ratio**	**Service Specific Satisfaction Ratio (q_1-q_2-q_3-q_4)**
Small Topology	Medium Load	90%	469	473	0.85%	10800	94%	97%-99%-90%-91%
	High Load	80%	495	497	0.40%	10800	91%	100%-100%-87%-80%
Medium Topology	Medium Load	95%	652	657	0.77%	10800	98%	100%-100%-97%-95%
	High Load	80%	674	679	0.74%	10800	90%	100%-100%-83%-80%
Large Topology	Medium Load	90%	-	784	-	10800	-	-
	High Load	80%	-	811	-	10800	-	-

The efficiency of this approach is tested with both MINLP and MILP models. In these experiments, two approaches are evaluated: i) fixing only the heaviest service (i.e., video streaming service), and ii) fixing two services with heavier load requirements (i.e., augmented reality and video streaming services). The results are presented in Tables 4.12 and 4.13. The upper bound values for the MINLP model presented in Table 4.10 are used to observe the efficiency of the approach. It can be seen that the MINLP model is still incapable of finding a feasible solution for some instances by fixing only one service type, but it can find solutions by fixing two service types.

Table 4.12. Matheuristic approach with MINLP model for service fairness problem.

		Fixing one service type			Fixing two service types		
Topology Type	**Load Condition**	Z_{MINLP}	**Difference from Z_{UB}**	**Service Specific Satisfaction Ratio (q1-q2-q3-q4)**	Z_{MINLP}	**Difference from Z_{UB}**	**Service Specific Satisfaction Ratio (q1-q2-q3-q4)**
Small Topology	Medium Load	463	2.16%	99%-92%-90%-91%	463	2.16%	99%-92%-90%-91%
	High Load	493	0.81%	100%-100%-82%-84%	483	2.90%	99%-84%-89%-84%
Medium Topology	Medium Load	-	-	-	637	3.14%	97%-96%-95%-96%
	High Load	667	1.8%	100%-98%-81%-80%	647	4.95%	100%-83%-86%-80%
Large Topology	Medium Load	-	-	-	772	1.55%	91%-91%-90%-90%
	High Load	-	-	-	782	3.71%	88%-80%-81%-81%

On the other hand, the MILP model can find near-optimal solutions in both approaches utilizing the matheuristic method. MINLP and MILP models may find different solutions for the same instances because they may fix different task assignment variables at the beginning of the second phase. In addition, fixing only one service type yields higher objective values for both MINLP and MILP models since having more fixed task assignment variables may move the model away from the optimality. Nevertheless, the solution quality obtained by fixing two service types is still satisfactory. Therefore, fixing more service types can still be a reasonable option since it reduces the problem complexity and may decrease the solution time.

The matheuristic approach utilizing both MINLP and MILP models finds a near-optimal solution with at most 4.95% optimality gap. Similar to the undifferentiated services problem, there is no study in the literature that addresses the fairness problem among various service types. Therefore, the matheuristic approach results are reported with the optimality gap calculated through the upper bounds set by the optimization models as the primary indicator of the performance. As a result, the matheuristic approach can be utilized to obtain good feasible solutions for the service fairness problem.

The optimal SLA-aware resource allocation problem and solutions introduced in this chapter address the end-user expectations and service requirements. However, con-

sidering the overall business model, another approach could be to address the demands of an operator as the owner of a multi-tier system. In this alternative setting, the operator can aim to optimize the orchestration practices for offloaded tasks to meet the performance requirements and also gain revenue in return. In such a problem definition, the operator can seek optimal placement of servers with different capacity levels and the deployment of the service replications for maximizing the expected revenue gain [128].

Table 4.13. Matheuristic approach with linearized model for service fairness problem.

		Fixing one service type			**Fixing two service types**		
Topology Type	**Load Condition**	Z_{MILP}	**Difference from Z_{UB}**	**Service Specific Satisfaction Ratio (q1-q2-q3-q4)**	Z_{MILP}	**Difference from Z_{UB}**	**Service Specific Satisfaction Ratio (q1-q2-q3-q4)**
Small Topology	Medium Load	468	1.07%	98%-97%-90%-91%	461	2.6%	93%-91%-95%-91%
	High Load	491	1.22%	100%-100%-82%-83%	480	3.54%	97%-90%-85%-83%
Medium Topology	Medium Load	649	1.23%	100%-100%-95%-96%	644	2.02%	100%-96%-97%-95%
	High Load	662	2.57%	100%-92%-80%-83%	647	4.95%	100%-81%-84%-83%
Large Topology	Medium Load	774	1.29%	90%-92%-90%-90%	771	1.69%	90%-90%-90%-90%
	High Load	794	2.14%	94%-81%-80%-81%	792	2.4%	90%-80%-84%-80%

5. OPTIMIZATION OF THE NETWORK SLICING OPERATIONS FOR HIGH-PERFORMANCE SERVICE ORCHESTRATION

Deploying replications of the services at the edge and offloading the tasks to the available edge servers in the vicinity solely is not useful in practice. Assuming that various services coexist in a heterogeneous setting and the offload operations are handled by the same physical infrastructure, the increase in the demand for a service may degrade the performance of the adjacent services. The congestion caused by the services and the introduction of a new application in this environment may affect the performance of all previously deployed services. Therefore, it becomes necessary to provide isolation among the service types and share the physical resources fairly.

Open Networking Foundation (ONF) [103] states that a network link, a networking node, and the computation capacity of a server can be partitioned into virtual resources. By reserving and allocating a specific capacity of a physical resource (i.e., computational and network resources) through virtualization techniques, the isolation between the slice instances is implemented. Hence, the performance of a slice may not be affected by the fluctuations or faults within an adjacent slice.

In this direction, we propose an optimal network slicing methodology for service-oriented edge access to provide isolation among the service types. The proposed Mixed Integer Programming (MIP) model aims to deploy the replications of the services optimally in a distributed manner while virtualizing the networking and computational resources for allocating specific capacities for the slices initiated for each service type. Additionally, a heuristic algorithm NESECS (NEtwork Slicing for Edge Computing Services) is introduced to find high-quality solutions in a short time for cases where the formal optimization tools fail to find feasible solutions.

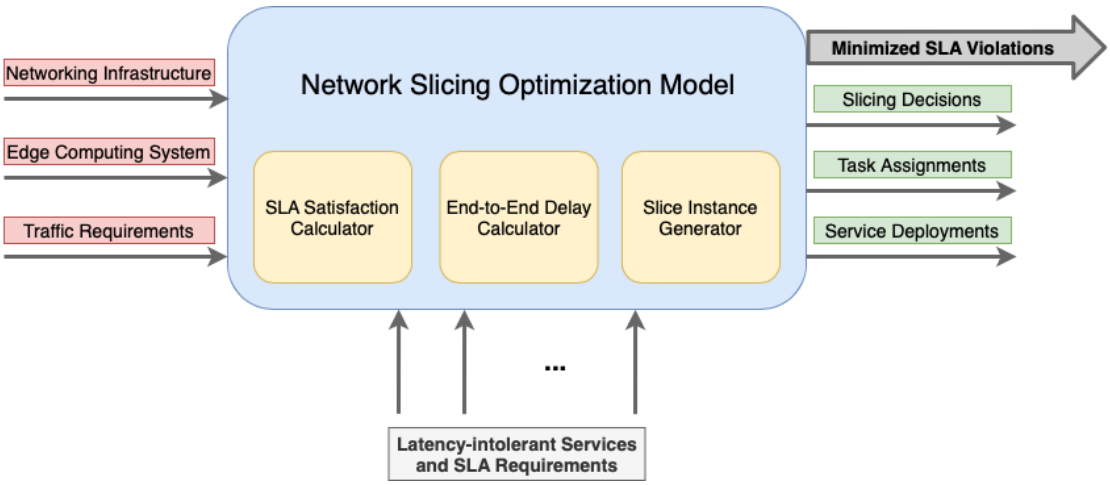

Figure 5.1. Optimization approach for network slicing.

5.1. Problem Definition

In an edge environment where the end-user devices offload various tasks, the main objective is to satisfy the performance requirements specified in SLA definitions and meet user expectations. Since most edge services impose stringent latency requirements, service deployment and resource allocation operations become complicated. The operator may reserve and allocate a certain amount of resources for each service type by partitioning the already-deployed physical resources. This approach avoids the performance degradation for a service type when the network traffic fluctuates, or another type of resource-hungry service exploits all available resources in the same infrastructure. Thus, we address the problem of service deployment decisions, generating slices for each service type, and reserving resources for these slices. In this thesis, we consider the end-to-end service latency as the main SLA requirement. However, the proposed solutions are flexible enough to accommodate other performance requirements such as energy efficiency and reliability. The general overview of the proposed formal optimization model is depicted in Figure 5.1.

The formulation of the given problem definition considers that the network topology $G = (N, E)$ is known. The set of networking resources N and links connecting them E is functioning. While the capacity of a network node n is denoted as c_n, the capacity of a link between nodes $(i, j) \in E$ is represented by a_{ij}, both in terms of Mbps.

Table 5.1. Set and parameter notations.

Definition	**Notation**
Network topology	$G = (N, E)$
Networking nodes	$N = \{n_1, n_2, n_3, ...\}$
Networking links	$E = \{e_{12}, e_{13}, e_{14}, ...\}$
End-users locations	$U = \{u_1, u_2, u_3, ...\}$
Edge server locations	$S = \{s_1, s_2, s_3, ...\}$
Service types	$Q = \{q_1, q_2, q_3, ...\}$
Max. number of service instances on edge server	y
Node capacity	c_n
Link capacity	a_{ij}
Edge server capacity	b_s
Computational load of service type q	m_q
Network load of service type q	l_q^{req}, l_q^{res}
Max. allowed delay for service type q	α_q
Delay violation penalty for service type q	w_q
Traffic requirements	r_{uq}

The edge servers are assumed to be deployed by the operator to host various service types. The set S denotes the server locations, and the computation capacity of a server $s \in S$ is designated as b_s in terms of MIPS (million of instructions per second). Since the edge servers' resources are restricted regarding the capacity, it is allowed that at most y VMs can be hosted on an edge server. Besides, this constraint minimizes the overhead of the lifecycle management of the VMs, and orchestration in practice.

The service types and their characteristics are known, where the set of services Q is composed of different service types with diverse SLA requirements. The SLA definition of a service type $q \in Q$ specifies that there is a maximum acceptable delay value, which is denoted as α_q in terms of milliseconds. However, in some cases where the network is highly congested, the end-to-end service latency may exceed the specified

maximum delay value. In order to reflect the deterioration in the system performance and user experience, there is a penalty w_q for service type q if the end-to-end delay requirement is violated. Since α_q for each service type q is rendered in milliseconds, w_q represents a penalty value for every millisecond that is violated, similar to the expected reward/penalty for an action taken in reinforcement learning approaches [129]. It is also specified that the task offloading operation for service type q generates m_q load on average as the number of instructions to be executed (in millions) on the edge servers. While a request for the same service introduces l_q^{req} load on average on the network resources, the response generated by an edge server additionally contributes to this load by l_q^{res} on average, both in terms of Mbit. The application provider may enforce that a certain fraction, δ_q, of the overall end-user requests for service type q should be assigned to any server that hosts the corresponding service type.

Lastly, the operator knows the number of users, their locations, and the average number of requests generated by each user for each service type. The set of users is denoted as U, and the average number of requests generated by user u for service type q in a unit time interval is represented as r_{uq}. The parameters and index sets that are utilized in the optimization model are summarized in Table 5.1. Slicing decisions and parameters are quite static in real deployments since they are explicitly defined in a contract between the communication service provider and the consumer. Even though it is possible to modify the properties of the allocated slice, this is rarely performed, and the optimality of the parameters should be evaluated in advance. Therefore, it is assumed that the service provider with the role of instantiating the slice instances has the information above beforehand, and the decisions are made accordingly.

5.2. MIP Model Formulation

According to the problem definition, the operator should configure and initialize the slice instances so that the instances are isolated from each other and the SLA requirements are not violated. Rather than satisfying the SLA requirements precisely under all conditions, there is a penalty for not meeting these requirements. The main

objective of the operator is minimizing the overall SLA violations, as well as the penalties to occur. Therefore, the model can be formulated as

$$\text{minimize} \sum_{u} \sum_{q} \sum_{s} p_{uqs} * w_q * \theta_{uqs} * r_{uq} + (1 - \sum_{s} \theta_{uqs}) * w_q * r_{uq} \quad (5.1)$$

Subject to

$$p_{uqs} = \begin{cases} 0, & \text{if } \tau_{uqs} \leq \alpha_q \\ \tau_{uqs} - \alpha_q & \text{otherwise} \end{cases} \quad (5.2)$$

$$\sum_{s} \theta_{uqs} \leq 1 \qquad \forall u, q \quad (5.3)$$

$$\theta_{uqs} \in \{0, 1\} \qquad \forall u, q, s \quad (5.4)$$

$$\theta_{uqs} \leq Y_{qs} \qquad \forall u, q, s \quad (5.5)$$

$$\sum_{q} Y_{qs} \leq y \qquad \forall s \quad (5.6)$$

$$Y_{qs} \in \{0, 1\} \qquad \forall q, s \quad (5.7)$$

where p_{uqs} represents the penalty due to SLA violation. In Equation (5.2), τ_{uqs} represents the end-to-end delay for a user u requesting service type q that is served by server s. If this value exceeds the maximum acceptable delay value, α_q, it contributes to the total penalty. On the other hand, if τ_{uqs} is less than α_q, it can be considered that SLA is not violated, and it does not contribute to the objective value. Additionally, θ_{uqs} is a decision variable that depicts whether the requests for service type q generated by user u that is successfully assigned to server s. To eliminate the principle of minimizing the penalty by not offloading the task at all, in the second part of Equation (5.1) penalty w_q is added to the objective for the tasks that are not assigned to any edge server. In Constraints (5.3) and (5.4), it is specified that any service request can be executed on at most one server or not assigned to any server respectively. In Constraints (5.5), Y_{qs} represents a binary value that takes value 1 if the service type q is served by server s, and it is enforced that any request for service type q cannot be handled by server s,

if the requested service is not hosted on that server. In Constraints (5.6), it is stated that at most y service instances can be hosted on a physical server. Lastly, Constraints (5.7) state that service deployment decision is represented as a binary variable.

The end-to-end delay is represented as the total latency contributed by the network and the computational resources. The classical congestion-dependent M/M/1 queuing model is applied throughout the network and edge servers. The end-to-end delay for an offloading process is constituted by the same operations discussed in the previous section. In practical scenarios, the arrival rate on average can exceed the service rate, and the system would become unstable. However, in this thesis, the average number of requests and the task assignment operations are used to optimize the slice parameters before initializing the slice. The projection of the minimum penalty in the future depends on the traffic requirements information and a task assignment scheme, and the proposed optimization model will not be used for each offloaded task. The model aims to allocate resources in advance to provide isolation among the slices. It can be achieved by reserving a certain fraction of a physical resource as a virtual resource for each service type q. The total network delay for service type q generated by user u to be executed on server s can be formulated as

$$\sum_{(i,j)\in P(u,s)} \frac{l_q^{req} + l_q^{res}}{\phi_{qij} * a_{ij} - \sum\limits_{u,s} r_{uq} * \theta_{uqs} * (l_q^{req} + l_q^{res})}. \tag{5.8}$$

In Equation (5.8), ϕ_{qij} represents the fraction of the physical resource capacity (i.e., the capacity of the link connecting nodes (i, j)) that is allocated for the slice of service type q. It is assumed that as soon as the corresponding edge server is determined for the task assignment, the minimum hop routing is used to transmit the data in the network. Therefore, $P(u, s)$ represents the set of links that constructs the shortest path between end-user u and edge server s. In this study, the packet processing latency by switching operations within a node is neglected.

After assigning the service type q task to an edge server s, the corresponding VM that hosts the requested service type handles the execution of the service routines. The code execution delay can be calculated as

$$\frac{m_q}{\gamma_{qs} * b_s - \sum_u r_{uq} * \theta_{uqs} * m_q} \tag{5.9}$$

where γ_{qs} represents the capacity allocated for the VM hosting the service type q on edge server s. The ϕ_{qij} and γ_{qs} decision variables are related with the main slicing operations Therefore, additional constraints can be expressed as follows.

$$0 \leq \sum_q \gamma_{qs} \leq 1 \qquad \forall s \tag{5.10}$$

$$0 \leq \sum_q \phi_{qij} \leq 1 \qquad \forall (i,j) \tag{5.11}$$

Constraints (5.10) and (5.11) assert that total capacities of the virtual resources cannot exceed the capacity of physical edge server and network links, respectively.

When assembled, the total end-to-end delay τ_{uqs} of a request for service type q that is generated by user u and handled by server s can be expressed as

$$\begin{aligned} \tau_{uqs} = & \frac{m_q}{\gamma_{qs} * b_s - \sum_u r_{uq} * \theta_{uqs} * m_q} + \\ & \sum_{(i,j) \in P(u,s)} \frac{l_q^{req} + l_q^{res}}{\phi_{qij} * a_{ij} - \sum_{u,s} r_{uq} * \theta_{uqs} * (l_q^{req} + l_q^{res})} \end{aligned} \tag{5.12}$$

The list of decision variables introduced in Equations (5.1)-(5.12), and their definitions are summarized in Table 5.2.

Table 5.2. Decision variables and definitions.

Variable	Definition	Type
p_{uqs}	The value delay limit is exceeded for a request for service type q that is generated by user u and handled by server s	Continuous
θ_{uqs}	Task assignment of a request for service type q that is generated by user u and handled by server s	Binary
α_{qs}	Service deployment for service type q on server s	Binary
$E(T_{qij})$	Networking delay contributed by network link (i,j) for service type q	Continuous
$E(T_{qs})$	Computation delay at server s for service type q	Continuous
ϕ_{qij}	Slicing decision on a network link (i,j) for service type q	Continuous
γ_{qs}	Slicing decision on a server s for service type q	Continuous
τ_{uqs}	Total end-to-end delay of a request for service type q that is generated by user u and handled by server s	Continuous

5.2.1. Addressing the Complexity of Slicing Operations and Delay Model

The original optimization model tends to suffer due to the non-linear objective function and set of constraints. In order to decrease the complexity and integrate the model into an MIP solver, the big-M methodology is applied upon the quadratic constraints, which is a common approach to linearize the constraints including product of binary variables and continuous variables [130], [131]. By tuning the M constant, this approach ensures that the linearized problem has the same optimal value as the original one.

Let us denote the latency contributed by server s for executing the instructions of service type q as $E(T_{qs})$, which can be calculated by Equation (5.9). The user index u is omitted since every user requesting the service type q encounters the same execution delay on an edge server s on average. The execution delay occurs regardless of which user requests the service. When the dividing operation is eliminated, the formulation can be written as

$$E(T_{qs})\gamma_{qs} * b_s - E(T_{qs}) \sum_u r_{uq} * \theta_{uqs} * m_q = m_q. \tag{5.13}$$

In Equation (5.13), both $E(T_{qs})$ and γ_{qs} variables are continuous, and their multiplication results in a non-linear constraint. In order to simplify the linearization efforts, we added an index γ_{qsz}, where $z \in Z$. The set Z can be considered as the identical fragments of a physical resource. Through this approach, γ_{qsz} becomes a binary variable, which equals 1 if the z^{th} fragment of the computational resource of server s is allocated for service type q. In this direction, additional constraints are defined as

$$\sum_{z} \gamma_{qsz} \leq Y_{qs}|Z| \qquad \forall q, s \tag{5.14}$$

$$\sum_{q} \sum_{z} \gamma_{qsz} \leq |Z| \qquad \forall s. \tag{5.15}$$

Constraints (5.14) enforce that a fragment of a computational resource can be allocated for service type q if that service type is hosted on server s. Constraints (5.15) ensure that the total number of fragments allocated for all service types on a server cannot exceed the number of fragments available on the physical resource.

Converting γ to a binary variable results in the multiplication of a binary and continuous variable in Equation (5.13), as $\sum_z E(T_{qs})\gamma_{qsz} * b_s$. Therefore, an additional continuous variable Γ_{qsz} is integrated for transforming the non-linearity into a set of linear constraints by using the big-M methodology where M is a large number

$$\Gamma_{qsz} \geq E(T_{qs})b_s - M(1 - \gamma_{qsz}) \qquad \forall q, s, z \tag{5.16}$$

$$\Gamma_{qsz} \leq \gamma_{qsz} M \qquad \forall q, s, z \tag{5.17}$$

$$\Gamma_{qsz} \leq E(T_{qs})b_s \qquad \forall q, s, z. \tag{5.18}$$

The second operand of Equation (5.13) also consists of the multiplication of a binary and a continuous variable, θ_{uqs} and $E(T_{qs})$, respectively. A similar approach is adopted for this quadratic constraint: proposing Θ_{uqs} variable as a substitution. We integrated the substitute variable and added new constraints as

$$\Theta_{uqs} \geq E(T_{qs}) m_q r_{uq} - M(1 - \theta_{uqs}) \qquad \forall u, q, s \qquad (5.19)$$

$$\Theta_{uqs} \leq \theta_{uqs} M \qquad \forall u, q, s \qquad (5.20)$$

$$\Theta_{uqs} \leq E(T_{qs}) m_q r_{uq} \qquad \forall u, q, s \qquad (5.21)$$

$$m_q = \sum_z \Gamma_{qsz} - \sum_u \Theta_{uqs} \qquad \forall q, s. \qquad (5.22)$$

After the transformation of the non-linear server delay formulation, the same actions are taken for the non-linear networking delay, as presented in Equation (5.8) of the original formulation. Let us denote the delay occurred on link (i, j) for service type q as $E(T_{qij})$. The equation can be rewritten as

$$E(T_{qij})\phi_{qij} * a_{ij} - E(T_{qij}) \sum_{u,s} r_{uq} * \theta_{uqs} * (l_q^{req} + l_q^{res}) = (l_q^{req} + l_q^{res}) \qquad (5.23)$$

Similar to the operations related to computational resources, the networking resources are also partitioned into identical fragments as the allocatable fractions of physical capacity. By using the same notation for a fragment, we have converted the original slicing decision for networking resources ϕ_{qij} to ϕ_{qijz}, where

$$\sum_q \sum_z \phi_{qijz} \leq |Z| \qquad \forall (i, j). \qquad (5.24)$$

Since ϕ_{qijz} is a binary variable, its multiplication with the continuous variable $E(T_{uqij})$ can be transformed into linear constraints by utilizing the big-M methodology as carried out with the computational resources. In order to achieve this objective, a new continuous variable Φ_{qijz} is added to the model. Correspondingly, the following constraints are added with regards to the big-M methodology.

$$\Phi_{qijz} \geq E(T_{qij})a_{ij} - M(1 - \phi_{qijz}) \qquad \forall q, (i,j), z \qquad (5.25)$$

$$\Phi_{qijz} \leq \phi_{qijz}M \qquad \forall q, (i,j), z \qquad (5.26)$$

$$\Phi_{qsz} \leq E(T_{qij})a_{ij} \qquad \forall q, (i,j), z \qquad (5.27)$$

The second operand of Equation (5.23) is also quadratic, due to the multiplication of two decision variables: $E(T_{uqij})$ and θ_{uqs}. Similar to the Θ_{uqs} in Constraints (5.19), (5.20), (5.21) and (5.22), the quadratic multiplication is substituted through a variable, which is denoted as ϑ_{uqs}. Therefore, a new set of constraints is defined as

$$\vartheta_{uqs} \geq E(T_{qij})(l_q^{req} + l_q^{res})r_{uq} - M(1 - \theta_{uqs}) \qquad \forall u, q, i, j \qquad (5.28)$$

$$\vartheta_{uqs} \leq \theta_{uqs}M \qquad \forall u, q, s \qquad (5.29)$$

$$\vartheta_{uqs} \leq E(T_{qij})(l_q^{req} + l_q^{res})r_{uq} \qquad \forall q, i, j, z \qquad (5.30)$$

$$(l_q^{req} + l_q^{res}) = \sum_{z} \Phi_{qijz} - \sum_{us} \vartheta_{uqs} \quad \forall q, i, j. \qquad (5.31)$$

Through these transformations, the set of constraints defined over the objective function in the original model is linearized so that the time and space complexity could be lowered to solve the model through an MIP solver. In order to address the same issue, a greedy heuristic implementation, namely NESECS, is proposed, and its implementation details are presented in the following section.

5.3. NESECS Heuristic Implementation

In addition to the formulated optimization model, a greedy algorithm called NESECS (NEtwork Slicing for Edge Computing Services) is developed. The primary aim is to obtain quick and feasible solutions for larger instances for which the formal

optimization tools cannot find any in a reasonable time. The steps of NESECS are provided in Algorithm 5.2. The general structure of the algorithm consists of three different phases: (i) service deployments, (ii) task assignments, and (iii) slicing decisions.

Before deploying the service instances, the algorithm makes prioritization among the set of servers. Based on the number of users each server can reach with at most x hops, the list of servers S is sorted in descending order. Then, starting with the server that falls within the range of the most crowded area, the algorithm deploys the service instances. Through a calculation made over the traffic requirements matrix r_{uq} for the users in the range of x hops, the algorithm deploys y most requested service types until all servers are processed sequentially.

Then, the solution starts assigning the offloaded tasks feasibly. First, for a user u, the algorithm finds servers deployed with the requested service type q. The algorithm tries to assign the offloaded task to the server, resulting in the minimum service delay, including networking and computation latency. During this phase, the algorithm also checks whether there is enough remaining capacity for handling the m_q of computational load. A similar operation is applied to network resources that are on the shortest path between user u and target server s. For each link on the path, the current load and the load generated by this request as $l_q^{req} + l_q^{res}$ are inspected. After calculating the expected server latency for all possible servers, the algorithm completes the assignment by allocating the server with the minimum service latency. The algorithm iterates by assigning each request until all requests are processed.

After completing the task assignment operations, the algorithm finds slicing decisions for networking and computational resources. The slicing decisions are based on the expected load on both networking and computational resources. When a slice instance is configured for a service type q, it would be a near-optimal decision to base it on the average load generated by that service type on a resource. Since the operator does not know the immediate demands of a service type, allocating more resource capacity than the expected load may lead to under-utilization in general.

Data: Q, U, S

Result: Objective value, slicing decisions

Sort S in descending order of number of users within x hops;

foreach $s \in S$ **do**
  Find and deploy y most requested service types within x hops;
end

foreach $u \in U$, $q \in Q$ **do**
  if $r_{uq} > 0$ **then**
    foreach $s \in S_{uq}$, *link* $ij \in shortestPath_{us}$ **do**
      if $m_q + serverLoad(s) > b_s$ *or* $linkLoad(ij) + (l_q^{req} + l_q^{res}) > a_{ij}$ **then**
        continue;
      end
    end
    Calculate $totalDelay(u, q, s)$;
    assign r_{uq} to s with minimum $totalDelay(u, q, s)$;
  end
  foreach $s \in S$, $(ij) \in E$ **do**
    Calculate total load as $totalLoad_s$;
    Calculate total load as $totalLoad_{ij}$;
    foreach $q \in Q$, **do**
      Calculate load of service type q as $load_{qs}$;
      $gamma_{qs} = load_{qs}/totalLoad_s$;
      Calculate load of service type q as $load_{qij}$;
      $phi_{qij} = load_{qij}/totalLoad_{ij}$;
    end
  end
end

Figure 5.2. NESECS algorithm.

For a computational resource, the slicing decision for service type q is calculated as the load generated by that service type divided by the overall load on that resource. Let

us denote the load generated by service type q on server s as $serverLoad_{qs}$ and the overall load on the same server as $totalServerLoad_s$. The computational slicing decision γ_{qs} for service type q on server s is calculated as $serverLoad_{qs}/totalServerLoad_s$. This approach is applied to all computational resources and service types. A similar slicing approach is applied to network resources. If the overall load on a link (i, j) is denoted as $totalLinkLoad_{ij}$ and the load generated by service type q on that link as $linkLoad_{qij}$, the network slicing decision ϕ_{qij} is calculated as $linkLoad_{qij}/totalLinkLoad_{ij}$. After obtaining the slicing decisions, the algorithm calculates the objective value, as depicted in Equation (5.1), and reports it as the output.

5.4. Performance Evaluation

Performances of the proposed MIP model solved with optimization tools and the heuristic algorithm are evaluated concerning the slicing operations. A detailed and realistic set of experiment cases is designed with randomly generated instances to execute a comprehensive evaluation phase.

5.4.1. Experiment Design

The quality of the proposed solutions is evaluated with different topologies generated using the NetworkX [132] package in Python. Three different topology instances with {50, 100, 200, 300} nodes are used in the tests, where each node's degree is chosen arbitrarily between [2,4] for creating a connected topology. Since evaluating the scalability of the proposed solution approaches is also critical, large-scale randomly generated instances are used instead of real internet topologies such as TopologyZoo [122]. For a particular topology, the number of users and servers is also factorized to observe the effect on the slicing operations. In each case, the locations of the users and servers are determined randomly. Each case is generated with a specific random seed to compare the performance of the proposed approaches accurately. The service types and their characteristics are designed to express the main motivation behind the network slicing and edge computing paradigms. Five different service types are created, as

depicted in Table 5.3, where m_q is denoted in MIPS, l_q^{req} and l_q^{res} in Mbit, and α_q in seconds. Among all, q_1 is shaped as a vehicular emergency service, which is the most sensitive service type to the latency with the highest penalty value.

Table 5.3. Service properties and characteristics.

Property	q_1	q_2	q_3	q_4	q_5
m_q	[500, 900]	[1800, 2000]	[1000, 1500]	[700, 1400]	[1500, 2000]
l_q^{req}	2	20	8	10	15
l_q^{res}	3	25	10	5	10
α_q	0.5	0.6	1.4	1.1	0.8
w_q	1000	10	500	200	300

The capacity of the links a_{ij} $\forall(i,j) \in E$ is set to 10000 Mbps [133] and the capacity of the edge servers b_s, $\forall s \in S$ is fixed as 40000 MIPS to emulate a server that is not very powerful in terms of computation. The capacities of similar sources are chosen identical so that we can interpret the results more accurately. However, capacities can be determined as desired thanks to the flexibility provided by the optimization model and the heuristic algorithm. The maximum number of service instances on a server y is set to 3 throughout the network, and the number of identical fragments on a physical resource z is set to 20. The locations of the edge servers and end-users and the traffic requirements matrix are determined randomly. In order to eliminate the effect of randomness and analyze different states of the environments, each instance is run with eight random seeds, and average results are presented. The same seed is used for corresponding instances to maintain consistency among the optimization model and NESECS algorithm.

The proposed optimization model is implemented in C++, and Gurobi 9.0.1 [58] is used as the MIP solver. Three hours is set as the maximum run time to find a feasible solution for each instance. Additionally, NESECS is implemented using C++, and both approaches are evaluated on a system with Intel Xeon E5-2698 CPU and 250 GB main memory.

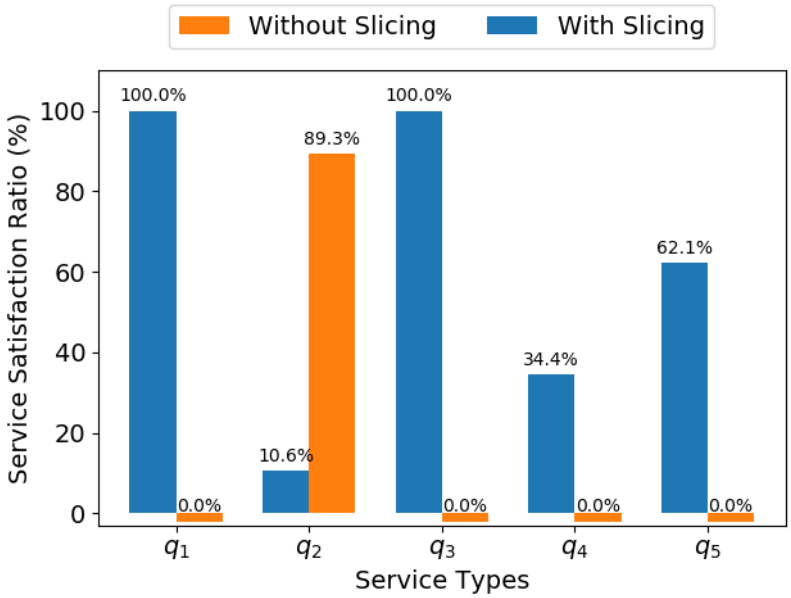

Figure 5.3. Service satisfaction ratios in case of an emergency.

5.4.2. Benefits of Slicing the Network

Before discussing the detailed performance analysis of the proposed solutions, the advantage of the slicing approach over traditional networking operations is demonstrated. The main objective is to preserve the QoS of emergency services by reserving resources through slice instances.

A use case is designed where the topology is composed of 21 nodes. Within this environment, 50 users and 5 edge servers are placed arbitrarily. The proposed optimization model and its equivalent without any slicing decisions are experimented with and run with the optimization solver. The optimization solver determines the corresponding decision variables for both models by satisfying all service types by 100%. After fixing the slicing, service placement, and task assignment decisions, the number of requests for service types q_2, q_4, and q_5, which are the non-emergency services due to low penalty values, is increased. In case of an emergency (i.e., instantaneous increase in service requests), to what extent the service satisfaction ratios are affected is analyzed, and they are presented in Figure 5.3.

When the slicing is enabled, the performance of service types q_1 and q_3, which are the critical services with the highest penalty values among all, does not fluctuate. However, when the resources are not sliced (i.e., without slicing), we can observe that none of the requests are satisfied for any service type, except q_2, because of the resource contention with an increased overall load. The requirements of service type q_2, a non-critical service such as infotainment, appear to be highly satisfied. However, in return, it is seen that emergency services could not be satisfied in any way. The main reason is that q_2 has the highest resource requirements with the lowest penalty, and it predominates the other service types regarding the resource contention. Therefore, the main objective of the network slicing is achieved through the resource reservation provided by the proposed optimization model.

It is also observed that the satisfaction ratio decreases for non-critical services when slicing is enabled. However, since the resources are reserved according to the average number of requests for each service type, resource contention occurs only among requests for the same service type. Therefore, even though the satisfaction ratio for non-critical services has decreased, it is more effective than the non-slicing approach.

5.4.3. Results

An additional approach that randomly assigns the tasks is implemented to assess the quality of the solutions. Instead of utilizing the minimum service latency assignment approach in NESECS, this approach assigns the tasks to a random server deployed with the requested service type. The average of 100 runs is reported as the objective found by the random assignment approach. The performance of the solutions is evaluated with different size instances, and the obtained results are presented in Table 5.4. The columns of this table can be explained as follows:

- *Instance (N, U, S)*: The test environment instance with the format of *(Number of Nodes, Number of Users, Number of Edge Servers)*
- *No. of solved instances*: The number of instances for which a feasible solution

can be found by the corresponding approach (out of 8)

- *Objective (Z_M, Z_R, Z_N)*: The objective value of the solution provided by the MIP model, random assignment and NESECS, respectively
- *Runtime*: Total amount of time in seconds spent by the corresponding approach
- *Improvement over Z_M (%)*: The improvement provided by NESECS over MIP in terms of the objective values, which is calculated as $\frac{Z_M - Z_N}{Z_M}$
- *Improvement over Z_R (%)*: The improvement provided by NESECS over random assignment in terms of the objective values, which is calculated as $\frac{Z_R - Z_N}{Z_R}$

Table 5.4. Main results of the optimization model and NESECS algorithm.

	MIP model			Random Assignment	NESECS				
Instance (N, U, S)	**No. of Solved Instances**	**Objective (Z_M)**	**Runtime (sec)**	**Objective (Z_R)**	**No. of Solved Instances**	**Objective (Z_N)**	**Improvement over Z_M (%)**	**Improvement over Z_R (%)**	**Runtime (sec)**
(50,20,10)	8	0	196.41	2534.89	8	75	-	97.04%	0.01
(50,40,20)	8	0	3215.11	8818.95	8	0	-	100%	0.07
(50,60,30)	8	50	10410.63	17435.40	8	30.17	-	99.82%	0.22
(50,80,40)	5	256	10800	27719.40	8	254.04	-	99.08%	0.48
(100,20,10)	8	0	2420.35	2650.96	8	775	-	97.32%	0.02
(100,60,30)	8	993.92	10800	16774.10	8	101.61	89.77%	99.39%	0.44
(100,100,50)	0	-	-	38024.50	8	414.94	100%*	98.90%	1.82
(100,140,70)	0	-	-	58652.40	8	441.99	100%*	99.24%	4.71
(100,180,90)	0	-	-	81057.65	8	5732.24	100%*	92.92%	9.63
(200,40,20)	8	504.48	10800	5274.82	8	690.47	-	86.91%	0.30
(200,120,60)	0	-	-	35227.96	8	201.46	100%*	99.42%	6.17
(200,200,100)	0	-	-	75919.30	8	1369.59	100%*	98.19%	26.93
(200,280,140)	0	-	-	121594	8	15354.38	100%*	87.37%	71.20
(200,360,180)	0	-	-	165747	8	5817.12	100%*	96.49%	146.28
(300,60,30)	4	13664.79	10800	7959.03	8	79.34	99.72%	99.26%	1.34
(300,180,90)	0	-	-	53177.10	8	1293.39	100%*	97.56%	29.90
(300,300,150)	0	-	-	118127	8	2578.01	100%*	97.81%	133.14
(300,420,210)	0	-	-	186087	8	5031.89	100%*	97.29%	352.42
(300,540,270)	0	-	-	258630	8	2960.64	100%*	98.85%	702.93

* 100% improvement means that NESECS obtains feasible solution for the corresponding instance which cannot be found by the MIP solver.

We can infer that the MIP model can find feasible solutions for small cases. It even finds the optimal solution for the smallest cases with 50-node topology in a

short time. However, the increase in the number of nodes, servers, or users makes the problem difficult due to time and space complexity. With the increase in the number of computational or networking resources in the environment, the number of slicing decisions also increases, making it challenging to find a solution. It can be said that the number of servers has the highest effect on the complexity of the model due to the corresponding number of decision variables. Not only the slicing operations but also the service placement and task assignment processes become complicated as the number of edge servers increases. On the other hand, while the number of users interrelated with the task assignment decisions, the topology size with various networking nodes affects the end-to-end delay and slicing operations. It should be noted that for cases where Gurobi cannot find a solution, the objective value can be considered as the upper bound in the system, in which none of the requests is assigned to an edge server.

Considering $(50, 20, 10)$, $(50, 40, 20)$, $(50, 60, 30)$, $(100, 20, 10)$, $(100, 60, 30)$ and $(200, 40, 20)$ instances, MIP model can produce a feasible solution for all 8 cases. Among these, in relatively smaller instances, it is observed that Gurobi could obtain the optimal solution in which no SLA requirements are violated. However, as the ratio of the overall load within the network to the total resource capacity increases, we see that the degree of SLA violations tends to increase in parallel. On the other hand, for the instance of $(50, 80, 40)$, Gurobi could only produce feasible solutions for 5 cases, and producing a feasible solution for 4 cases of $(300, 60, 30)$ instance. When the instances of $(100, 60, 30)$ and $(300, 60, 30)$ are compared, it can be said that the increase in the number of nodes makes it difficult to find a solution. On the other hand, due to both model complexity and the networking delay, the objective value obtained by Gurobi is much higher in $(300, 60, 30)$ instance. The maximum runtime allowed for the MIP model on Gurobi is set as three hours, and this time is completely used for all cases to find the best solution, except for the small instances.

On the other hand, the NESECS algorithm successfully produces a feasible solution for all instances. When the objective values found by NESECS are compared with the ones obtained by the optimization model, we see that the NESECS outperforms

the MIP model in general, except for the smallest instances with 50-node topology. For the instances where the MIP model produces a better solution in terms of the objective value, the improvement provided by NESECS is not reported. While Gurobi produces better solution than NESECS for small cases by producing the optimal solution in general, the improvements in the objective value brought by NESECS for the cases where Gurobi also produces feasible solutions are 89.15% and 99.72% for $(100, 60, 30)$ and $(300, 60, 30)$ instances, respectively. Since Gurobi could not solve the remaining 11 instances, improvement provided by the NESECS algorithm is reported as 100%.

To validate the quality of the solutions found by NESECS for the cases where Gurobi fails to produce any, the results are compared with the objective value obtained through random assignment. Utilizing the NESECS approach instead of arbitrarily assigning the tasks enhances the quality of the solutions significantly. Among all, the least improvement provided by NESECS over random assignment is 29.46%, and the maximum improvement is 99.78%. Considering all instances, the average improvement as a percentage provided by NESECS over random assignment is 63.32%.

When we examine this table, we can say that the MIP model shows good performance in small environments by producing optimal or near-optimal solutions. However, as the test environment gets larger, the rate of improvement brought by NESECS increases, and it produces a better solution than Gurobi in a very short time.

When we analyze the results of the algorithm in terms of runtime, we observe that it can produce a good solution for small environments instantly and for large environments in a short time. While NESECS spends 0.03 seconds in the smallest topology to produce a feasible solution, it spends 991.82 seconds in the most extensive test environment. We also observe that the number of users and servers is more significant than the number of network nodes in terms of the algorithm's runtime.

An additional comparison of the solution methods is performed on a varied number of users with 100 nodes and 80 servers, as presented in Figure 5.4. According

to the results obtained through Gurobi, the average objective value tends to increase as the number of users exceeds 40. The main reason is that the number of users in the environment as well as the number of task assignment decisions that need to be made increases, and so Gurobi obtains a feasible solution that is far from optimality. This inference can also be made with regards to the variance among the solutions of a corresponding instance. When there are 50 or 60 users in the system, Gurobi fails to assign any offloaded task to an edge server for some of the random instances, which results in high variances. Additionally, this situation applies to all random instances when the number of users exceeds 60. In these scenarios, Gurobi cannot assign any offloaded task, and this is reported as the feasible solution. Therefore, the objective is valuated by penalizing the offloaded tasks that are not assigned to any edge server.

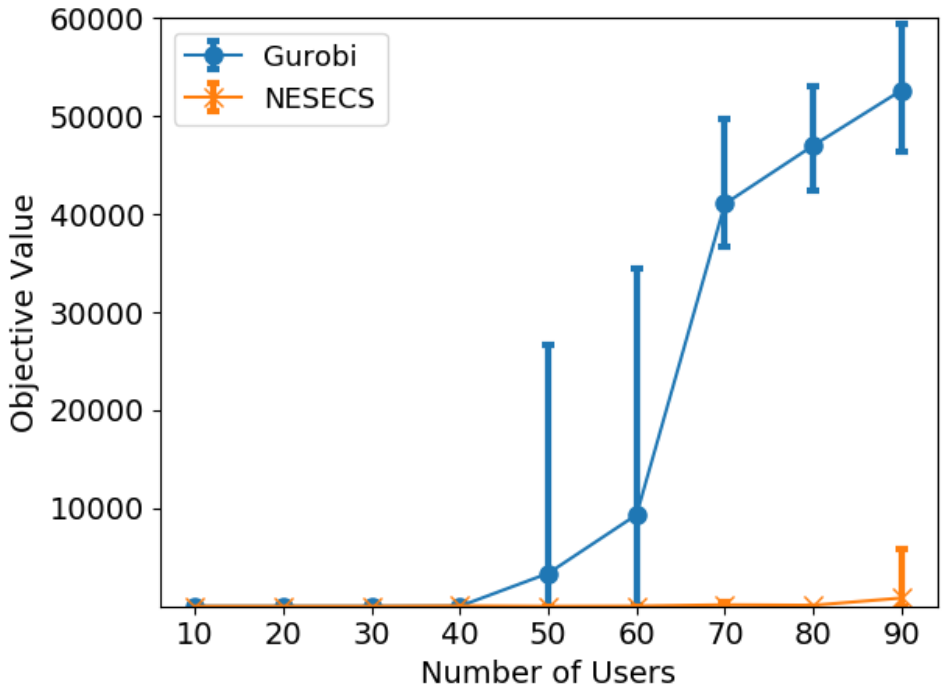

Figure 5.4. Effect of the number of users on the objective value for 100 nodes and 80 servers case.

When we examine the results of NESECS, we can observe that the SLA requirements are satisfied when the number of users is less than 70. Considering that NESECS can find near-optimal solutions to all cases in a short time, with the increase in the number of users, the algorithm can still assign service requests to the servers with low

latency. As the number of users continues to increase, we see that the SLA violations occur within the environment, but the slope of the curve is far less than the curve obtained through Gurobi. On the other hand, with the increase in the number of users, the variance of the objective value among 8 cases in a particular instance increases. However, in these cases, it can be seen that the algorithm can still find the optimal solutions in which the SLA requirements are not violated.

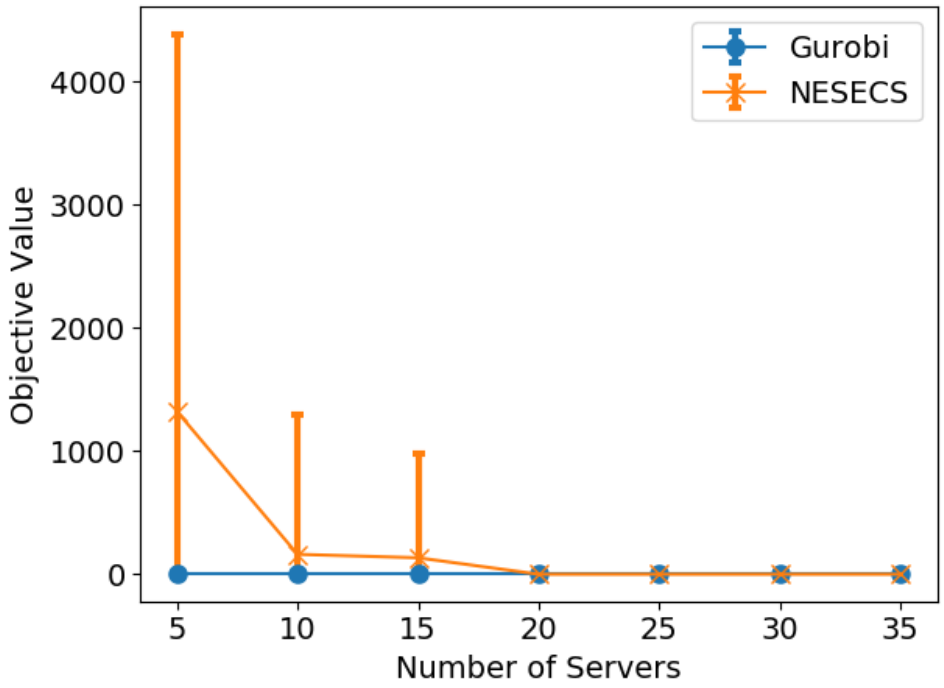

Figure 5.5. Effect of the number of servers on the objective value for 50 nodes and 50 users case.

Similarly, the effect of the edge server capabilities in an environment on the objective value is also evaluated by comparing the proposed solutions. As presented in Figure 5.5, Gurobi could obtain the optimal solution in all instances by not violating any SLA requirement. On the other hand, the objective value could be minimized by NESECS algorithm by increasing the total capacity of the computational resources in the environment. While the deviation among the cases is high for the cases where the overall capacity of the computational resources is restricted, deploying more edge servers in the environment makes it possible to obtain low deviation and an objective value close to the optimality.

As a general outcome, we can say that Gurobi could obtain the optimal value when the available computation capacity is relatively restricted, and the proposed MIP model can be preferred as the primary approach as a solution for the slicing operations. However, as the number of available edge servers in the environment exceeds a certain level, NESECS could produce near-optimal solutions in a very short time. In such cases, NESECS can be efficient in terms of time and space complexity, and more dynamic actions can be taken for the slice management operations at the edge.

6. IMPLEMENTATION OF THE SERVICE-CENTRIC BEHAVIOR WITH PROGRAMMABLE NETWORK PARADIGMS

In the service-centric approach, the services themselves are handled independent of their location, and the focus shifts to "what" instead of "where". Due to the complexities involved, establishing service-centricity at the edge scenarios is not a straightforward task. As a remedy, this thesis proposes a service-centric approach at the edge using orchestration capabilities offered by the programmable network paradigms. Correspondingly, two solution methodologies are developed to achieve this objective: (i) SDN/OpenFlow and (ii) fully-programmable P4 approach. In this section, these proposals are defined and presented along with their performance evaluation. In order to provide a practical application of service-centric implementation, the SDN-based solution is integrated into a pervasive healthcare environment to orchestrate the fall-risk assessment service within the multi-tier structure.

6.1. SDN Implementation

This thesis envisions a sub-service granularity while maintaining the intrinsic features of the service-centric behavior. The set of novel services to be requested by the end-user devices is considered the composition of multiple sub-services. The main reason for decomposing a service instance into sub-procedures is to provide a decent granularity environment. A gadget may execute some of the sub-services locally and offload the remaining part of the task to a remote computation resource [134].

A sample face recognition service that is composed of many sub-services distributed over the end-user gadget and edge servers is depicted in Figure 6.1. A subset of procedures is deployed on the first server. Another subset (common or distinct sub-services) is deployed on the second server, and the last server provides all of the sub-services. In parallel, it should be noted that the end-user device can also execute

some of the service routines locally to optimize the overall execution process. In most cases, it is expected that the sub-services can be served from many locations to increase the system's general performance by balancing the load. This sample illustrates the insight of the sub-service resolution and provides an example case for the sub-service deployment over edge servers.

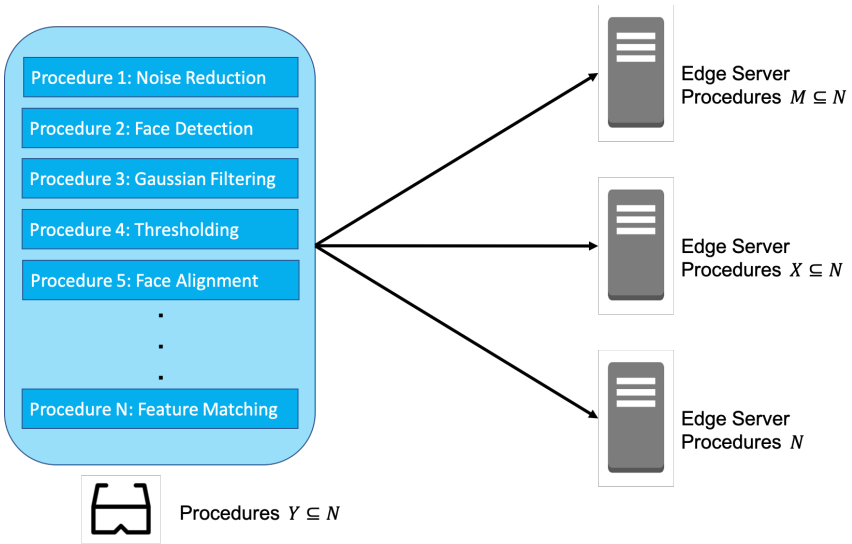

Figure 6.1. Distribution of sub-services over the mobile device and edge servers.

The sub-service resolution allows the system designers to reflect the underlying code-offloading mechanisms into the networking level. Similar commercial attempts exist, such as Amazon Lambda, which allows users to deploy and execute a portion of a code on the Amazon computational resources [135].

6.1.1. System Implementation and Northbound Applications

In order to come up with an efficient scheme that addresses the requirements of service-centric networks and operations with the sub-service resolution, SDN and OpenFlow capabilities are exploited. OpenFlow Extensible Match (OXM) allows using more than 40 different fields as the primary matching mechanism within the flow tables. However, the flexibility provided by the OXM is not available in the legacy TCP/IP stack. The proposed solution should be compatible with the legacy solutions where the network infrastructure is heterogeneous. As a remedy, DSCP (Differentiated services

code point) field of the IP header is used to identify sub-services in the TCP/IP domain before interacting with the SDN domain. In order to identify the service itself, to which the requested sub-service belongs, typical TCP port numbers are used. Historically, 6-bit DSCP and 2-bit ECN (Explicit Congestion Notification) replaced the 8-bit Type of Service (ToS) field to support QoS requirements and mitigate the congestion. ECN has an essential role in the current network structure for congestion control, but DSCP is still not commonly used. The DSCP field is designed by IETF (Internet Engineering Task Force) for differentiated services. This behavior is per-hop and implemented locally in each router. When we consider the OpenFlow-enabled switches, they will not have such implementations soon since they are kept as simple as possible. Therefore, in this thesis, the DSCP field is used with sub-service addressing to achieve the desired service management qualities for edge services.

The main objectives are achieving the task offloading operations using a sub-service resolution instead of the current IP address-based design and satisfying the QoS requirements of the end-users, such as low end-to-end service delay. The SDN as an enabling management layer can realize these objectives through a software-based controller and northbound applications where each one defines a different feature for the whole system.

The general framework of the proposed system has the following components:

- Users and applications that demand different sub-services
- Data plane with OpenFlow-enabled network devices
- Edge server that accessible through a LAN connection
- SDN controller
- Northbound applications for load balancing and service orchestration
- Global service locator

The main operations of the system when a request is generated by an end-user are presented in Figure 6.2. In Step 1, the end-user generates a sub-service request

by providing the destination port number and DSCP value. The OpenFlow-enabled switch buffers this packet and forwards a replication to the controller in Step 2 if it cannot match the packet with any of the flow rules within its flow table. The generated request needs to have a destination IP address to be routed. However, since the user application does not know the service location, it initially inserts a generic IP address defined by the system into the packet header. This information is then modified by the first switch within the network to be forwarded to the destined server.

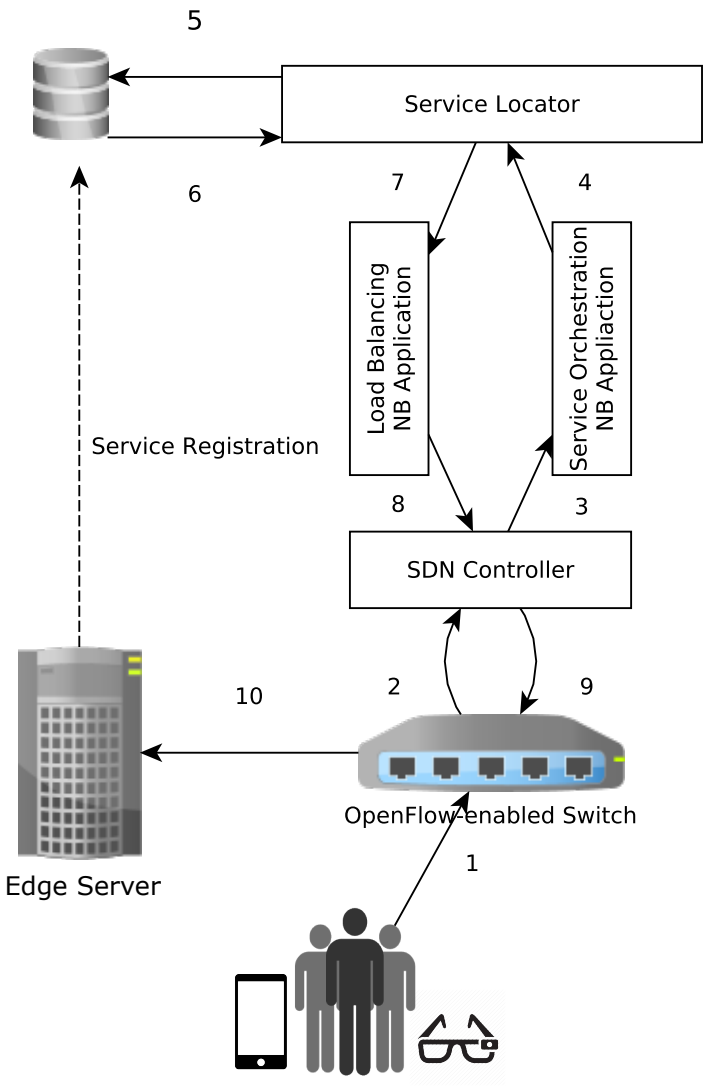

Figure 6.2. Processes for forwarding the service request to the edge server.

The service orchestrator receives this packet in Step 3. It interacts with the service locator in Step 4 to obtain the corresponding location of the edge server that

can provide the demanded sub-service indicated by the DSCP value. The locator queries its database with the specified port number and DSCP value and creates a list of IP addresses available to accomplish the offload operation in Steps 5 and 6.

In general, the orchestrator takes the list of IP addresses and forwards it to the load balancer. These steps are merged as a single step in 7. Load balancer northbound application then internally decides on the destination server with a load balancing algorithm and sends the necessary information about this server, such as its IP address, to the controller in Step 8.

The controller installs a flow rule in Step 9 with a particular timeout value. In OpenFlow, two different timeout values are implemented to enhance the flexibility of the flow table management. First is "idle timeout", which indicates that if no flow arrives that matches a specific rule within that timeout period, the flow rule is removed automatically. The other one is "hard timeout" that causes the removal of the flow rule after the timeout period, independent of the number of the matched flows. These parameters have a significant effect on the performance of the proposed solution.

The newly inserted flow rules indicate that any further request for a sub-service with the corresponding port number and DSCP value should be forwarded to the specified server by replacing the IP header field with the corresponding edge server. After installing the flow rule, it sends an OpenFlow message (OFPT_PACKET_OUT) to the switch to apply the same set of actions on the buffered packet. Lastly, at step 10, the incoming request is forwarded to the edge server that provides the requested sub-service. Besides, any edge server can make a sub-service registration by sending a notification to the controller at any time when it is deployed with that sub-service.

As illustrated in Figure 6.2, the service locator keeps track of the services, sub-services and their locations by introducing a mapping mechanism within the database. The database tuples are recorded through the service registration process initiated by the edge servers. When a sub-service is deployed over an edge server and becomes

available to be executed through a task offloading request, it needs to inform the service locator. Correspondingly, a new record is created by specifying the identity of the edge server and related service information (i.e., corresponding packet header fields). When a task offloading operation arrives, the service locator queries the database with the given TCP port number and DSCP field. As a reply, it generates a list of IP addresses that provide the requested sub-service.

The main contribution of the proposed solution is the customized control layer composed of the SDN controller and two different northbound applications.

6.1.1.1. Service Orchestration. When a sub-service request arrives at the OpenFlow-enabled switch with a specific destination port number and the DSCP field, the switch forwards any unmatched request to the controller. The controller handles this event by delegating it to the service orchestration northbound application. The service orchestration extracts the packet header fields and generates a query for the service locator to find the set of edge servers that provide the requested sub-service. After that, the Service Locator provides the list of corresponding IP addresses to the service orchestrator. Although the load on computational resources plays a vital role in the performance of the framework, the load on the controller also has a significant impact. Therefore, tasks that require a considerable amount of processing are assigned to a northbound application instead of the controller itself. Thus, the orchestrator cooperates to locate the services and reduce the burden on the controller for avoiding performance degradation.

6.1.1.2. Load Balancing. The load balancing application operates in coordination with the service orchestrator. Once the service orchestrator receives the list of servers, it assigns the task of deciding on the most feasible edge server to the load balancing northbound application. After the load balancing application decides on the server, it informs the SDN controller by sending the destined edge server's IP address. In order to apply similar operations on the further requests belonging to the same flow, the controller installs the corresponding flow rules.

Four different load balancing methods are implemented as separate northbound applications. The first algorithm balances the load among the servers considering their CPU usage. By collecting the CPU utilization information from the edge servers frequently, it can determine the least loaded one within an IP address set provided by the service locator.

The other method uses the already existing mechanisms in SDN, where load balancing is done based on port statistics. The load balancing application is able to collect statistics using OpenFlow statistics request message (*OFP_PORT_MULTIPART_REQUEST*) sent to the switches, which reply with port statistics (*OFP_PORT_STATS*), including the number of packets or bytes sent over a port. As a result, it can find the least loaded edge server by keeping the recent statistics. While the first northbound application maintains the balanced state among the servers concerning the computational resources, this algorithm aims to achieve the same objective considering the load on the network resources.

The third load balancing application is implemented by following the traditional Round Robin algorithm at the sub-service level. Lastly, an application that assigns sub-service requests to the edge servers in random order is used in experiments to provide a lower bound of performance.

6.1.2. Performance Evaluation and Load Balancing

This section presents the experimental setup and the performance evaluation of the proposed infrastructure under various load conditions.

The proposed system is implemented over the Mininet [136] evaluation environment, which is configured on a computer with Intel i5-3210M 2.5 GHz CPU and 4GB main memory. Ryu [137] is chosen as the SDN controller because is supports the OpenFlow versions that include the OXM mechanism. The system is implemented in compliance with OpenFlow v1.3 specifications. Within the Mininet environment,

OpenvSwitch (v2.5) instances are used to emulate the network nodes.

In the initial phase of the experiments, the service-centric behavior of the proposal is demonstrated. For the second part, experiments are conducted to evaluate the performance of load balancing northbound applications to minimize the end-to-end service delay. Each experiment is repeated 5 times, and an average of 100000 sub-service requests are generated for each run.

6.1.2.1. Enabling the Service-Centric Approach at the Edge. The first objective of the framework is to enable the end-users to request a sub-service without depending on the network location - i.e., the IP address. Therefore, the initial experiment setup is conducted in order to present this behavior of the implemented system.

Table 6.1. Sub-service deployments for the initial experiment.

Edge Server	Service	Sub-service	IP Address
Edge 1	Port: 50006	DSCP: 4	10.0.0.1
Edge 2	Port: 50006	DSCP: 8	10.0.0.2
Edge 3	Port: 50006	DSCP: 12	10.0.0.3

A single service experiment is designed where the service is identified with port number 50006, and its sub-services are deployed over three different edge servers. The sub-service deployments and the corresponding scenario are summarized in Table 6.1. Accordingly, a single client that has access to the edge servers through LAN/WLAN requests all of these sub-services sequentially to accomplish the service as a whole.

The peaks in Figure 6.3 correspond to the exact times when the sub-service request arrives at the corresponding edge server so that the CPU load increases to execute the routines of that sub-service. The client application generates a request by specifying the TCP destination port number and DSCP value without knowing the destination IP address. Therefore, it initially embeds a generic IP address defined by

the system. This IP address is then modified by the OpenvSwitch on the path and replaced with the edge server's IP address. As observed, the sub-service requests are forwarded to the appropriate edge servers that satisfy the offload operations. In other words, an increase in the CPU load on a server indicates an ongoing execution.

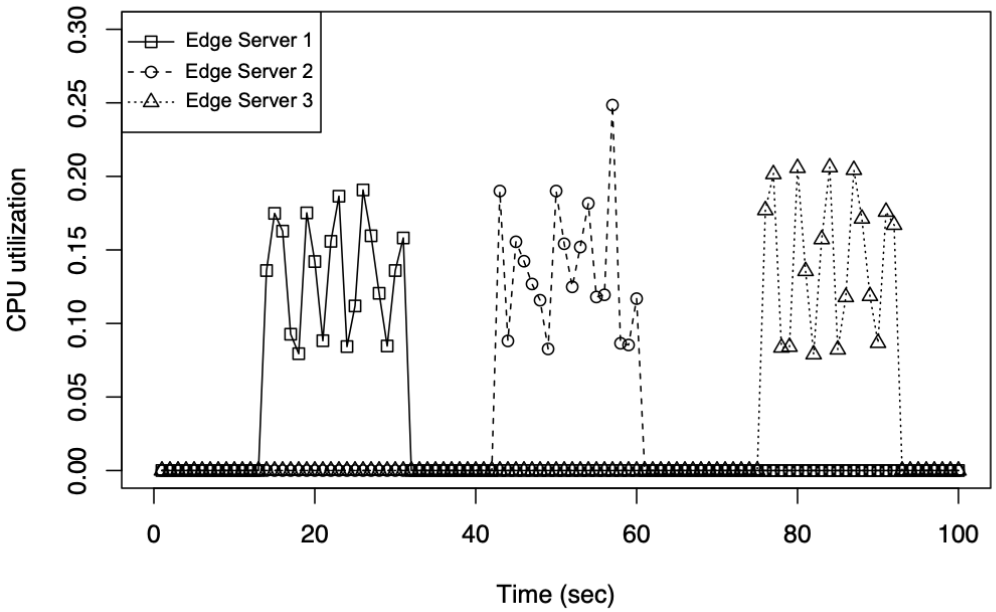

Figure 6.3. Demonstration of the service-centric behavior at the corresponding edge servers.

Initially, the edge servers' CPU loads are close to 0 until the user application generates the initial request. The implementation is expected to execute the first task at Edge Server 1 since it is only served there. Between the 15th and 30th seconds, the first sub-service request (DSCP 4) is forwarded to Edge Server 1 by the SDN controller, and it is executed there. The client application then generates a request for the second sub-service (DSCP 8), which is executed by Edge Server 2 between the 40th and 60th seconds. Lastly, the application generates the latest sub-service (DSCP 12) request, which is provided by Edge Server 3, and the request is forwarded to the corresponding server as observed. As depicted, the system can resolve the locations of the sub-services that are requested by the user application and forward them accordingly. Thus, the service-centric behavior is successfully implemented by the SDN control plane.

6.1.2.2. Minimizing the Service Delay with Load Balancing. It is worth noting that the delay for each sub-service is considered as the "service delay" since a client can request sub-services independent from each other. In the experiment setup, many clients are assumed to request sub-services from five different but identical edge servers in the vicinity.

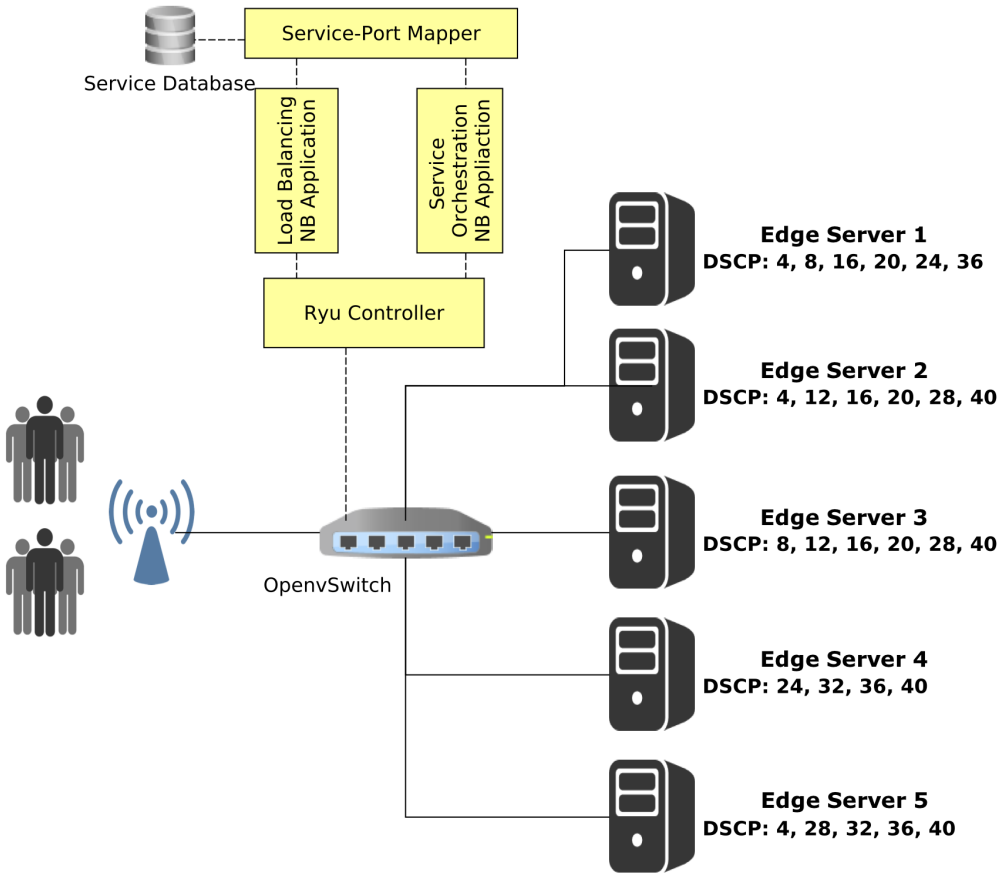

Figure 6.4. Experiment setup for the performance evaluation of the framework.

The service identified with port number 50006 has ten sub-services with DSCP values 4, 8, 12, 16, 20, 24, 28, 32, 36 and 40. The distribution of these sub-services and the experiment setup are shown in Figure 6.4. There are 40 distinct request types defined with different combinations of these ten sub-services, as shown in Equation 7.1 and Equation 7.2, where SS stands for the set of sub-services and each $Request_i$ represents a request type consisting of a subset of SS. For evaluating the performance under different user behavior, each client requests services by randomly selecting one of these subsets.

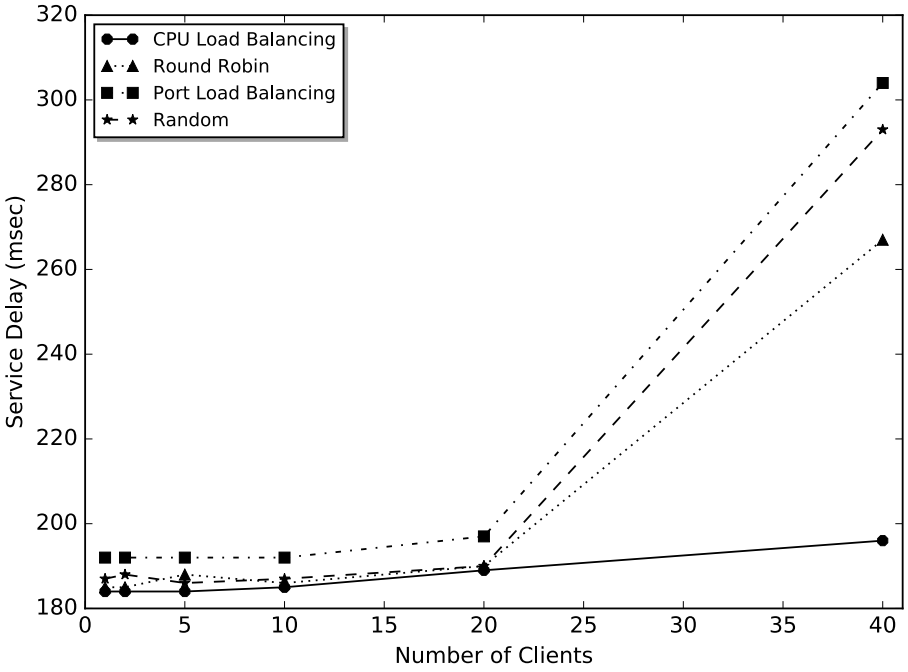

Figure 6.5. Effect of load balancing applications with different number of clients on service delay.

After all sub-services within that subset are completed, the client chooses another available subset of sub-services and sequentially creates requests for each element of this subset. It is essential to mention that each sub-service is unique, and the generated computation load is distinct. The corresponding routines have the characteristics of a CPU-burst application. It does not cover any I/O-related operation so that only the the related computations generate load on the CPU.

$$SS = \{4, 8, 12, 16, 20, 24, 28, 32, 36, 40\} \tag{6.1}$$

$$Request_i \subseteq SS \text{ where } \forall \text{i, } 1 \leq i \leq 40 \tag{6.2}$$

As stated, four load balancing methods are implemented, and these are compared regarding the average end-to-end service latency. Figure 6.5 depicts the effect of user traffic on service delay. In this setting, flow rules have 5 seconds idle timeout, and the northbound application collects CPU/port statistics for every 1 second. For less than

20 clients, all methods provide approximately the same service delay, while CPU load balancing provides the least delay. However, as the number of clients increases, port load balancing, Round-Robin and random assignment result in higher service delays where the CPU load balancing still provides the lowest service delay. It is not affected significantly by the increasing number of clients. Since each sub-service generates a different amount of computation load on the servers, the load balancing approach that considers the load on the computational resource performs the best among all. It is able to track all the changes of the edge server utilization, thus assigning the sub-service to the least-loaded one results in the lowest service delay.

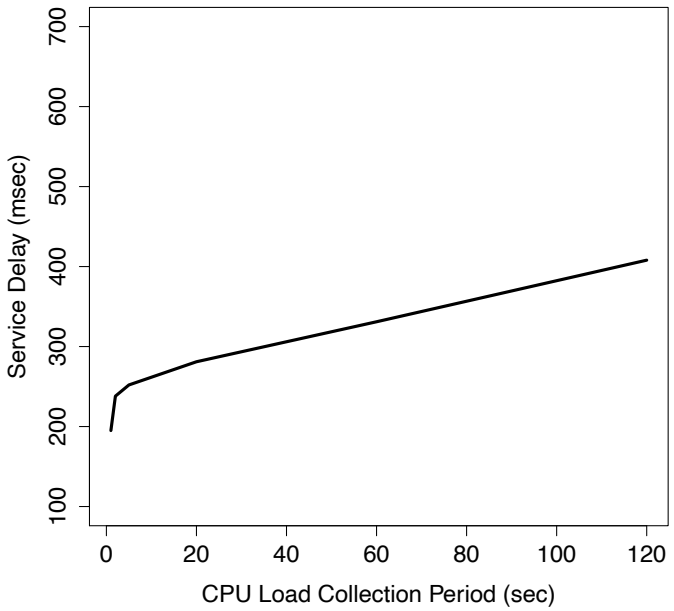

Figure 6.6. Effect of CPU load collection period.

On the other hand, load balancing according to the port load collection results in the worst performance. The main reason is that the number of bytes or the number of packets forwarded by a single port does not differ much among the set of sub-services. Thus, it does not reflect the main load within the network. However, it performs even worse than the random sub-service assignment. Collecting the port statistics through OpenFlow messages generates an extra load on the networking resources, an overhead that does not occur in other cases. For achieving the most accurate and complete results, both network and computation resources can be considered together.

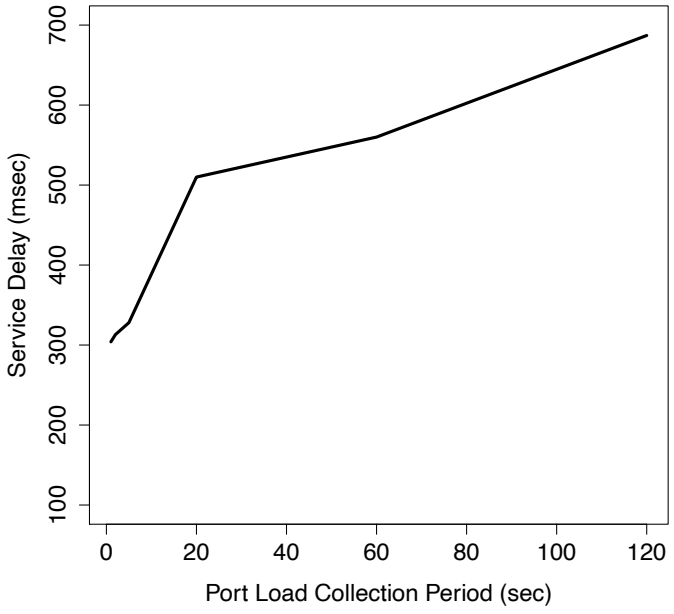

Figure 6.7. Effect of port load collection period.

The frequency of the load statistics collection from the servers and the switches affects the performance of the corresponding methods. To capture this effect, the same environment is utilized with 40 clients where flow rules are installed with 5 seconds of the idle timeout. The effect of the time period between each load collection and balancing the load according to the recent values are shown in Figure 6.6 and 6.7. The minimum period of load collection is 1 second for both methods. Figure 6.6 shows that as the time between two consecutive load collection operations increases, the service delay increases as well. The main reason is that the flow rules are installed according to the latest CPU load information of the edge servers. As the period increases, it does not reflect the most recent state of the resources. The same information can be inferred for the port load collection in Figure 6.7 by sending and receiving OpenFlow messages. As the time between two consecutive OpenFlow statistics request messages increases, the service delay tends to increase in parallel. Although the smaller period results in higher numbers of OpenFlow messages within the network that can cause congestion at specific points or increase the load on the switches, it does not become the bottleneck. Hence, it is extracted that the period of load collection by both servers and switches has an important effect on the performance of the framework.

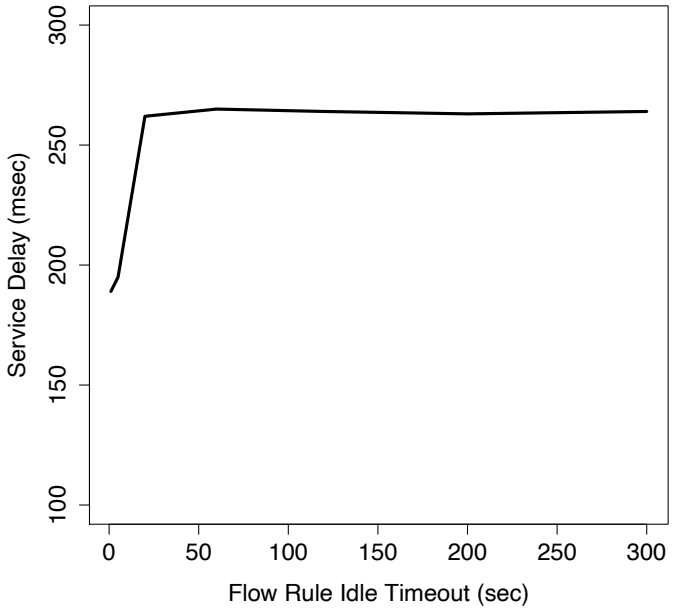

Figure 6.8. Effect of idle timeout on service delay.

Another system parameter is related to the timeout information within the flow rules installed by the controller. Since the CPU load balancing provides the best performance, the idle timeout effect is observed in this method. The number of clients is fixed to 40, and the load collection period is configured as 1 second. The effect of changing the idle timeout for CPU load balancing is presented in Figure 6.8. As the idle timeout increases from 1 (minimum idle timeout) to 30, it increases the service delay, but increasing the idle timeout further does not affect the service delay. The initial increase in the service delay is caused by matching the old flow rules that do not reflect the most recent state concerning the load balancing. Therefore, as the flow rule idle timeout increases, the number of matching packets also increases. The system is highly dynamic, and the load on the edge servers fluctuates continuously with various task offloading operations. After the idle timeout reaches 30 seconds, the service delay does not increase further because the average inter-arrival time between two consecutive requests is not higher than 30 seconds. Thus, the flow rules are never timed out, and the same rules are matched. This behavior heavily depends on traffic characteristics and user behavior. As the average inter-arrival time changes, the point where the service delay becomes stable may change. On the other hand, the minimum

service delay is achieved with the minimum idle timeout in every case. As a result, this parameter is a crucial indicator that affects the performance. Hence, it needs to be optimized according to the user behavior.

6.2. SDN-based Orchestration of Pervasive Healthcare Services

The applications, including physical exercise assistants, sleep trackers, and stress detectors, aim to give users continuous feedback for a healthier lifestyle. While smartwatches and bracelets, with their onboard sensors, provide a convenient platform for ordinary users to track their wellbeing, more specialized healthcare-dedicated sensors grant clinicians the ability to monitor their patients like never before. A set of typical healthcare services and the corresponding devices are depicted in Figure 6.9.

The spectrum of these novel services raises some requirements. The personalization of the services strongly depends on the user context, such as previous health records, and user profile. The wearable sensors generate huge amounts of data, but only a small portion can be analyzed and turned into relevant indicators of wellbeing. This is partly due to the insufficient computation capabilities, limited storage capacity and battery constraints of the devices. Even though the small form factor devices have become more powerful than ever, their performance is still unsatisfactory to execute healthcare service routines, as the complexity of the relevant algorithms also increases.

The complementary integration of edge and cloud servers may concentrate the beneficial aspects of both approaches in a single architecture organized in a layered manner. However, the internal operations of this multi-tier computing architecture and orchestration of the heterogeneous environment are complex. Resolving all these complications requires an external policy entity with an effective monitoring module. The actions could be taken dynamically to enhance the QoE (Quality of Experience) at the edge of the network. The functional capability of the traditional networking infrastructure remains inadequate to implement a reactive orchestration mechanism that can cope with the highly progressive expectations of the services and end-users.

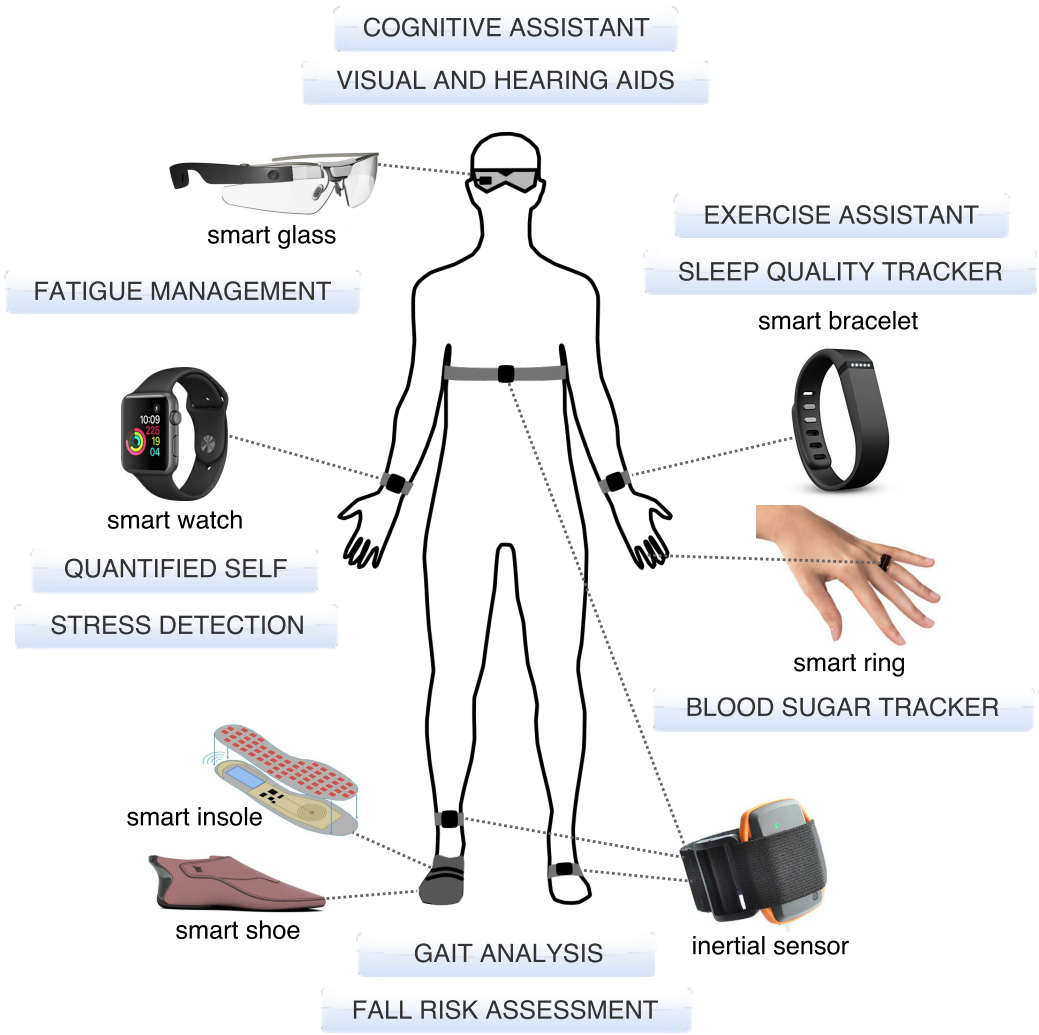

Figure 6.9. Spectrum of smart gadgets and healthcare services.

An SDN-based multi-tier computing and communication architecture is proposed to facilitate personalized healthcare services in a pervasive manner. In order to satisfy various requirements in such a dynamic environment and embrace the heterogeneous characteristics in a federated setting, it is of utmost importance that edge and cloud servers cooperate in harmony. While the cloud servers are utilized for delay-tolerant and resource-hungry tasks such as improving the model and the long-term storage of the personal health records, the edge servers are utilized to execute the healthcare service routines. We are still able to benefit from mobile devices for simple operations such as preprocessing the raw sensory data. By constructing different levels of computation and storage resources, pervasive communication can be achieved with an enhanced management resolution.

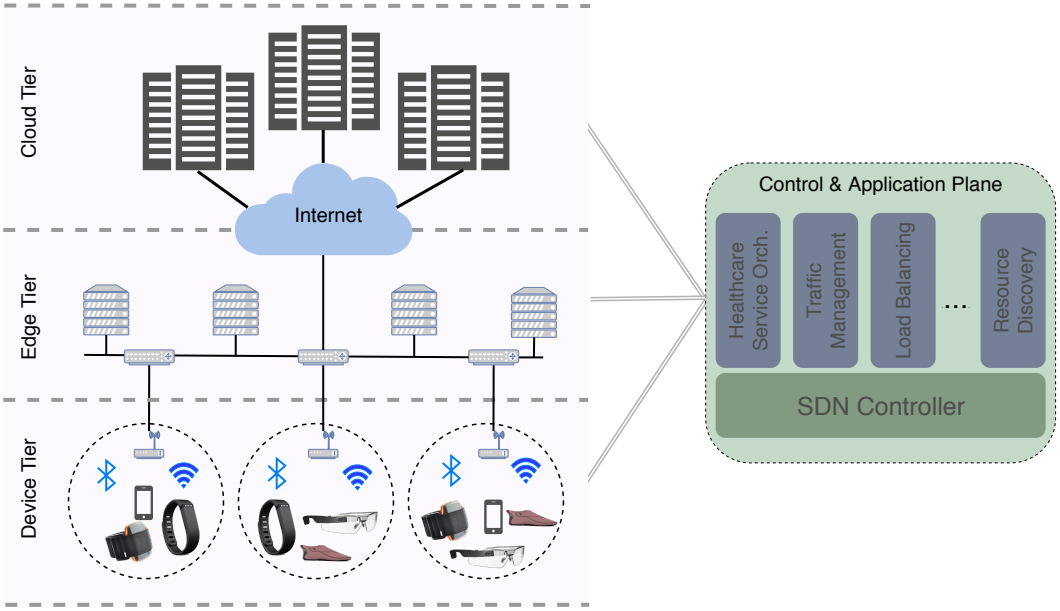

Figure 6.10. SDN-based multi-tier computing and communication architecture.

6.2.1. SDN-based Implementation for Healthcare Services

As depicted in Figure 6.10, the multi-tier computing architecture is composed of a pool of resources organized in a layered manner. The overall infrastructure with the healthcare services deployed throughout the network is controlled and orchestrated with the SDN control plane. We can integrate various northbound applications into the control plane that can be customized according to the requirements of the environment.

As a practical use case to demonstrate the validity of the proposed architecture, we also designed and implemented a fall risk assessment service distributed across multiple tiers by collaborating with the pervasive health group of our laboratory. Continuous fall risk assessment using wearable inertial sensors is a service that can benefit greatly from Edge Computing since it requires near real-time analysis to make timely interceptions possible. Accessibility of the patient records history is also crucial since fall risk is tightly related to changes in gait and motor functions over time. The long-term storage deems a central and reliable entity such as the cloud. At the same time, the short-term data could be stored at the edge to enable time-window analysis, which is essential for typical gait analysis implementations. The service is designed for a hospital or nursing home scenario, where multiple mobile residents are simultaneously

using the service across different floors and rooms. The clinicians or caregivers are alerted to any deviations in a timely manner.

6.2.2. SDN Control Plane Components

The main contribution of this solution proposal is the implementation of a northbound application to leverage the autonomous management of resources in the network for healthcare applications. The northbound application provides functionalities including service discovery, topology discovery, resource allocation and load balancing by using the Python API of the Ryu SDN controller [137].

The load balancing northbound application is presented in Algorithm 6.11. When an unmatched packet arrives at an OpenFlow switch, a replication is sent to the controller for determining the action to be applied with an *OFPT_PACKET_IN* message. The controller then relays the incoming packet to the northbound application, and the header fields are investigated. The northbound application particularly extracts the destination TCP port number information since it is the field that specifies the requested service type as presented in the previous section. Thus, service-oriented access for healthcare applications is enabled by concentrating the service discovery at the control plane instead of the end-user device.

After finding out the requested healthcare service by mapping the TCP port number, the list of servers that are available to provide the requested service is produced. Among this list, the algorithm tries to choose the one that minimizes the end-to-end service latency. For this purpose, it continuously retrieves the CPU load information of the servers and monitors their current state. This thread runs for each second independent of the ongoing operations within control and data planes. The load balancing algorithm assigns a probability to each server inversely proportional with the CPU load of the server. Then, according to these probabilities, the algorithm selects one of the servers. The destination server is determined according to the probability value, which is assigned inversely proportional to the current CPU load.

```
Data: Servers, services
Result: Destination server
while true do
    foreach s ∈ Servers do
        load(s) ← newData;
    end
end
totalLoad ← 0;
foreach s ∈ Servers(tcpPortNumber) do
    totalLoad ← totalLoad + load(s);
end
totalFrac ← 0;
foreach s ∈ Servers(tcpPortNumber) do
    frac(s) ← totalLoad/load(s);
    totalFrac ← totalFrac + frac(s);
end
probOfServers ← [];
foreach s ∈ Servers(tcpPortNumber) do
    prob(s) ← frac(s)/totalFrac;
    for i = 0; i < prob(s); i++ do
        probOfServers.append(s.ipAdress);
    end
end
selectedServer ← probOfServers.random();
return selectedServer;
```

Figure 6.11. Northbound application algorithm for load balancing.

After this operation, the algorithm creates a set of instructions for modifying the destination MAC and IP address fields so that the corresponding packet can be routed to the destination server. As stated, the northbound application is implemented with a topology discovery routine, which frequently communicates with the underlying control plane, and updates the topology accordingly. Thus, the northbound application

is capable of creating routing rules and instructions to enable end-to-end connectivity. In addition to the routing capability, traffic management practices can be applied to the packets to enhance the performance by monitoring the network load by exchanging OpenFlow messages with the data plane entities.

After determining the set of matching fields and list of actions to be applied on the incoming packet, this policy translated into an OpenFlow message *OFPT_FLOW_MOD*. When this message is sent to the switches, a new rule is added to the flow table. This OpenFlow message includes the following information:

- Hard Timeout, set to c seconds
- Matching Fields = [(destination TCP port = identifier of the requested service type), (source IP address = IP address of the requesting end user device)]
- Priority = 50000 (i.e., max. priority)
- Actions = [(modify destination IP address), (modify destination MAC address), (output port of the switch)]

As specified, the flow rule installed in the switches is composed of two matching fields: (i) source IP address and (ii) destination TCP port number. The motivation behind this is that whenever the same end-user device requests the same service type, the list of actions specified by the flow rule entry is applied without creating a load on the control plane. First, the destination MAC address is modified to specify the next hop, then the destination IP address is modified to route the packet to the corresponding server. Lastly, the physical port of the switch is also defined, from which the incoming packet is forwarded.

In order to fully enable the end-to-end connectivity, another *OFPT_FLOW_MOD* message is relayed to the corresponding switch. The purpose of this OpenFlow message is to define the routing rules in the reverse path (i.e., from server to the end-user device). This part of the algorithm is omitted in Algorithm 6.11 in order to keep it simplified. It is important to mention that an *OFPT_FLOW_MOD* message does not have an effect

on the unmatched packet that is buffered by the switch. Thus, for applying the same policy on this packet, an OpenFlow message *OFPT_PACKET_OUT* that has the same content with the *OFPT_FLOW_MOD* message is sent by the controller.

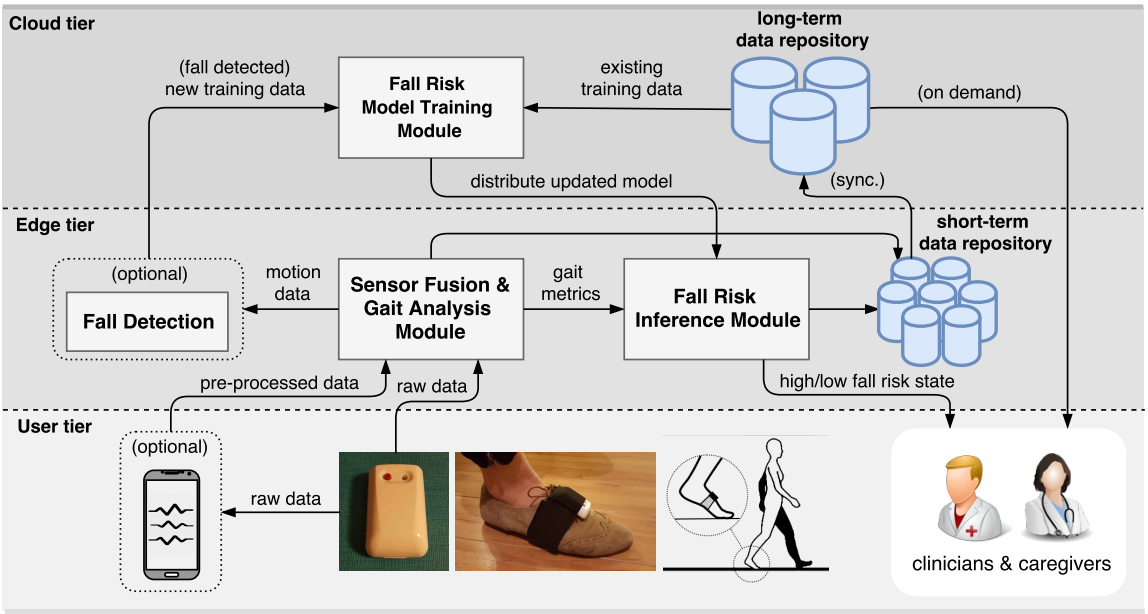

Figure 6.12. Edge-enabled fall risk assessment service architecture [138].

6.2.3. Fall Risk Assessment Service Architecture

In order to implement a fall risk assessment service, a multi-tier computation system is proposed as depicted in Figure 6.12. This multi-tier system incorporates the cloud datacenter, edge servers and end-user devices as separate layers of computational resources. To meet various requirements and provide high-performance service to the end-users, raw data collection, pre-processing, inference and training functionalities are deployed throughout the tiers in a distributed manner.

In a typical scenario, the Bluetooth-enabled sensors may transmit the collected data to the edge servers. Another alternative is to utilize a smartphone as an intermediate hop, where the raw data is pre-processed before forwarding to the edge server. When the edge server retrieves the sensory data, it is stored, the gait parameters are extracted, and the fall risk classification process is executed. If there is a high risk of fall, a notification is sent to the clinicians or the relatives immediately so that the

necessary action can be taken on time. The latency-intolerant behavior of the fall risk assessment service demands edge infrastructure to perform the phase of decision making. The details of the fall-risk assessment mechanism were presented in [138].

6.2.4. Performance Evaluation and Results

This section provides detailed information about the network emulation environment used to evaluate the performance of the proposed architecture. The experiments and the system parameters used in the experimental scenarios are also introduced.

The proposed solution approach is implemented and experimented with in the Mininet [136] setting, which runs on a system with Intel Core i7-6700K CPU, and 16GB DDR4 main memory. We used Ryu SDN controller [137] for the control plane operations and implementation of the northbound application, which follows the OpenFlow v1.3 specification. Within the Mininet emulation environment, Open vSwitch (OVS) [139] instances are used as the networking resources in the data plane.

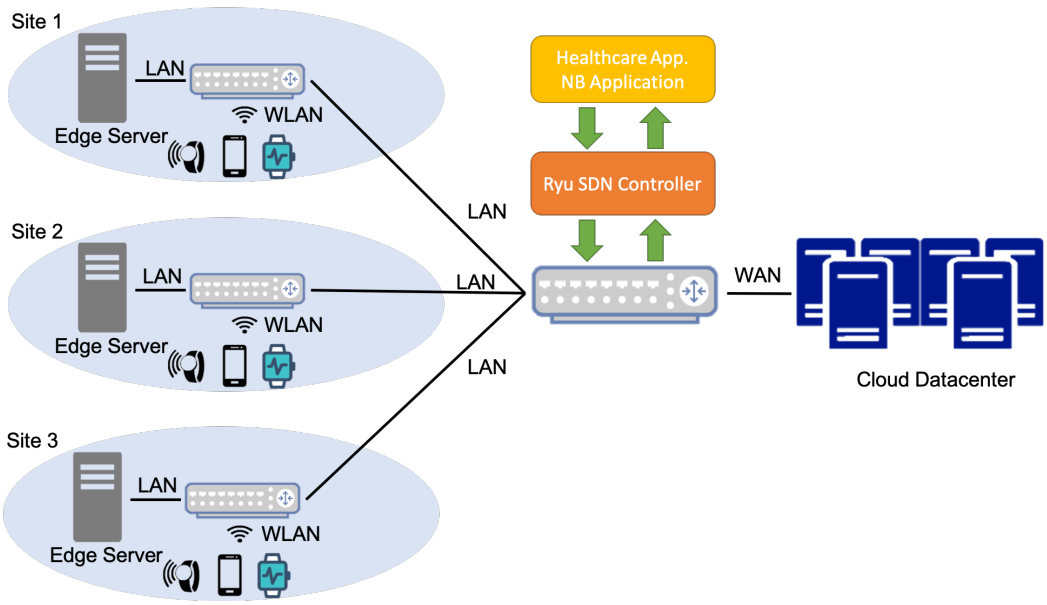

Figure 6.13. Emulation environment of SDN-based multi-tier architecture.

Figure 6.13 shows the network topology we implemented on Mininet for the experiments. As depicted, there are three different sites, each of which covers various end-users demanding the fall risk assessment service. In order to evaluate the scala-

bility of the proposed solution, seven different cases are designed with regards to the number of users at the edge: 1, 5, 10, 20, 30, 40 and 50. The number of users per edge site is determined arbitrarily for each case.

The edge servers are deployed with the identical fall risk assessment procedures and configured on Mininet in a way that an edge server is placed in each site. On top of the edge system, a cloud server is made available to execute the necessary procedures in case of a request. The edge and cloud servers are configured as CPU-limited hosts in Mininet to emulate a realistic scenario, where the edge servers are restricted when compared to the cloud server. In this direction, each edge server is configured to use 20% of the host machine's CPU power (considering all the available cores). In contrast, the cloud server can consume the computation power without any restriction.

The network parameters are also adjusted precisely to emulate the real network deployments. The WLAN links connecting the end-users to the network emulate typical 802.11n connections, and the bandwidth of these links is configured to support 200 Mbps data rate according to average throughput in typical indoor scenarios [140]. Since Mininet does not provide any functionality to create and emulate wireless links, we configured them as wired links by using the corresponding parameters to represent a WLAN connection in the test environments. In addition to this, The LAN links provide connectivity with the edge servers, and they are configured to emulate Gigabit Ethernet (i.e., 1 Gbps). The propagation delays in the LAN are almost negligible. Therefore, we set the delay for the interfaces implementing the LAN links to 5 ms using the Linux traffic control (tc) configurations. Lastly, the link that connects the sites to the cloud datacenter is configured to emulate the WAN access. Since WAN data rate is much lower than the edge of the network due to congestion, the bandwidth is set to 100 Mbps and the propagation delay is set to 250 ms. The emulation parameters for the network links are also presented in Table 6.2.

The sensor data collected from real patients are used in Mininet hosts for the main traffic generation and service request operations. In this direction, a scenario where

the continuous monitoring of the fall risk assessment service is exemplified and each client simultaneously generates 5 minutes of data traffic. The details of an alternative scenario where each user generates traffic for a random duration according to a uniform distribution and performance evaluation results of this scenario are available in [138].

Table 6.2. Network link parameters in Mininet environment.

Link	**Bandwidth**	**Propagation Delay**
WLAN	200 Mbps	5 ms
LAN	1 Gbps	5 ms
WAN	100 Mbps	250 ms

The client and server processes are implemented in Python by using socket programming techniques. The server process executes the fall risk assessment routines after receiving the data sent from the client process runs in the Mininet hosts at the edge sites. Since reliability of the communication is critical for the analysis accuracy, TCP transport layer protocol is used.

For evaluating the the performance of the proposed solution approach, the following scenarios are selected and designed:

- Multi-tier with load balancing
- Multi-tier without load balancing
- Only cloud

In the multi-tier environment with a load balancing approach, SDN controller and northbound application are utilized so that has the objective of minimizing the end-to-end latency can be achieved. In the second approach, load balancing is disabled, and the SDN control plane is configured to redirect the end-user requests to the closest computation resource (i.e., edge server). Lastly, the only cloud approach is evaluated to observe the performance of the system when all tasks are offloaded to the cloud server with higher computation power through a congested network. Each of these scenarios is repeated 10 times, where a single run takes 15 minutes on average.

6.2.5. Performance of Multi-Tier Architecture

The scalability of the system is a vital issue since the dynamicity at the edge of the network, and various request patterns may affect the overall performance. The availability of the service and low end-to-end delay should be preserved even when the popularity increases within a particular region.

First of all, in order to demonstrate that the proposed architecture of the multi-tier system and the northbound application operate as desired, the difference between the load balancing approach and offloading to the closest edge server is shown in Figure 6.14 and Figure 6.15. It can be observed that the maximum CPU utilization throughout the experiment that lasts 10 minutes is lower when the northbound application is active and the load generated by the offloaded tasks is distributed among the computational resources in a fair manner (Figure 6.15). In the load balancing scenario, the CPU utilization of a server reaches ∼20% at most. However, on the other hand, the servers utilize all of their available resources (i.e., 100% CPU utilization) in the second approach, where the tasks are offloaded to the closest edge server.

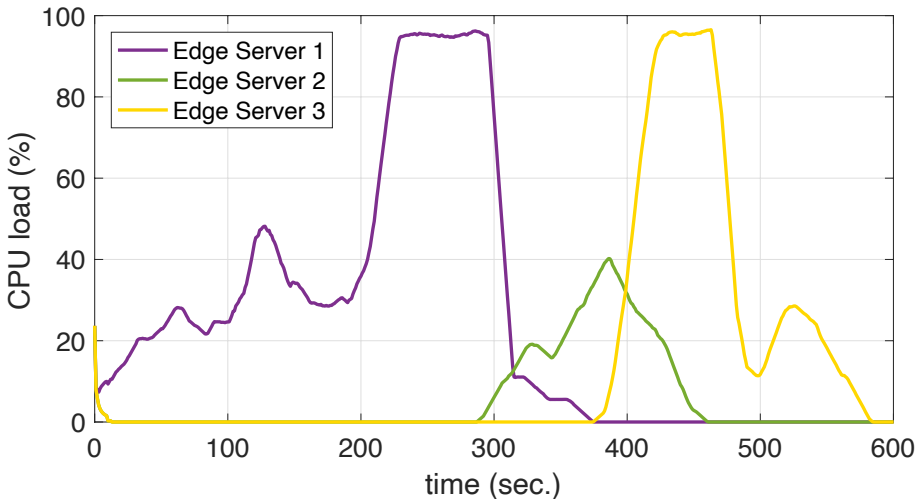

Figure 6.14. Without load balancing.

Figure 6.16 shows the performance of the system where each user generates traffic for the same amount of time (5 minutes). When the network becomes dense with more fall risk service requests, the average service delay for each approach increases. While

the cloud-only approach provides equivalent service quality when the network is less congested, the average service delay increases with the increasing number of end-users. In fact, in this scenario, the cloud tier provides the worst result in terms of service delay among the three approaches. On the other hand, the multi-tier structure with load balancing still provides better performance than the approach where the requests are offloaded to the closest edge server.

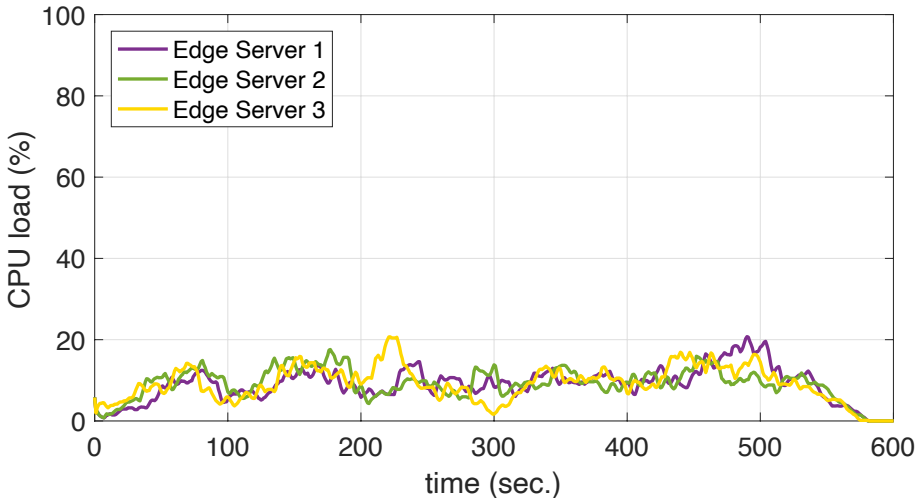

Figure 6.15. With load balancing.

In general, it can be stated that the Edge Computing system is capable of minimizing the end-to-end service delay so that the stringent latency requirements of such services can be met. However, due to the scalability issues, it demands an efficient network management policy with a load balancing functionality provided by the control entity. When we observe the results obtained through extensive tests as depicted in Figure 6.16, the multi-tier system provides the fall risk assessment service with lower service delay than the only cloud approach, since the edge-tier actively participates in the offloading operations. However, once the number of offloaded tasks reaches a certain point, the processing capability of the edge servers remains incapable of meeting the low latency requirement. In other words, the edge servers may suffer from the scalability problem in dense settings, and the overall performance may decrease. The proposed load balancing approach with the SDN control plane is efficient in assigning the offloaded tasks to the computational resources, which increases the scalability, as shown by the results.

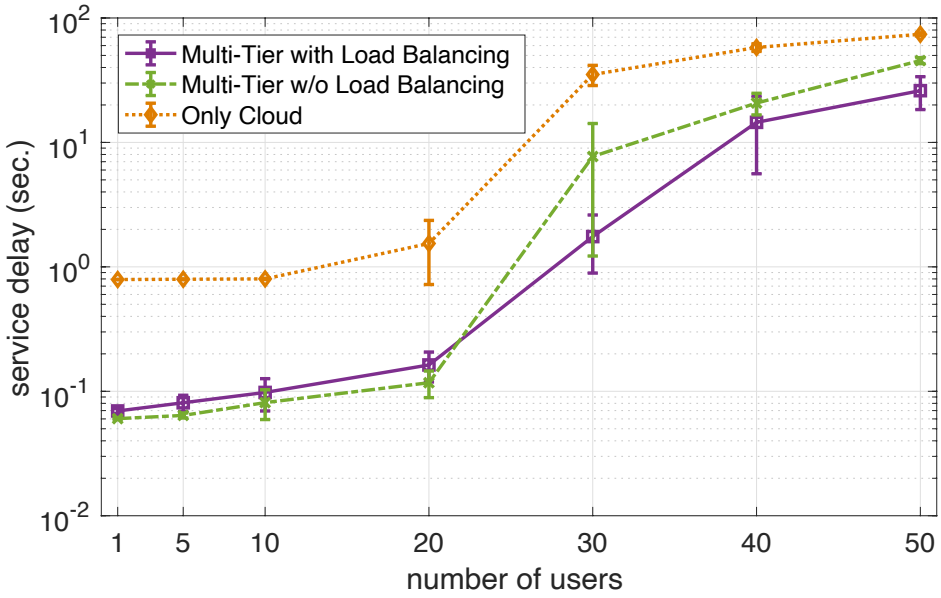

Figure 6.16. Comparison of multi-tier approaches with continuous monitoring.

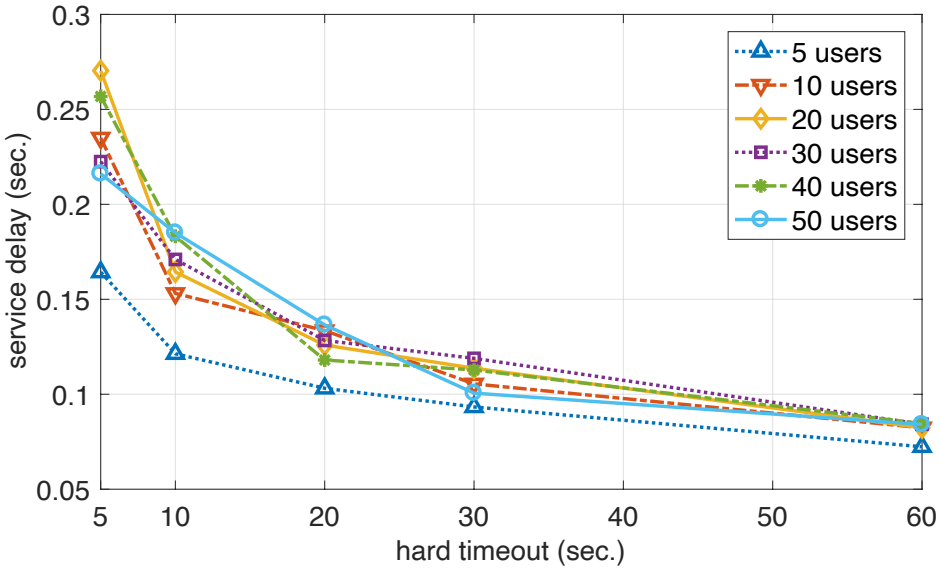

Figure 6.17. Effect of hard timeout on load balancing performance.

Lastly, the effect of hard timeout parameter on the average service delay is analyzed. Hard timeout is important for optimizing the performance of the load balancing algorithm. Figure 6.17 presents the evaluation with varying values of hard timeout. The average service delay tends to decrease up to a certain point as the hard timeout values increase, then becoming stable. Our load balancing approach distributes the load among the servers in advance, and triggering load balancing after the expiration time does not affect the performance greatly. Increasing the hard timeout value in-

dicates a lower burden on the controller. As the hard timeout value increases, the number of packets forwarded to the controller also decreases since the probability of matching with a flow rule becomes higher. The further increase in the hard timeout value exhibits similar results because after the loads are distributed fairly for once. The average service delay until the next trigger is already nearly optimized.

6.3. Fully-programmable Implementation Methodology of Service-centric Networks: P4

Our short-term solution based on the SDN approach [12] embraces the idea of utilizing available header fields that OpenFlow supports in order to define identifications of services and sub-services. The primary motivation of this work is to present a solution that leverages the necessary functionalities of a service-centric model without modifying the TCP/IP protocol stack. Using the header fields supported by OpenFlow, the SDN-based solution approach addresses the service identification with TCP/UDP port numbers and sub-service identification with the 6-bit DSCP field.

Being forced to utilize the supported fields is not the most feasible solution in the long run. Although the solution can name 2^{16} services, the number of sub-services for each service instance is limited to 2^6. In addition to this, using DSCP field and port numbers may be challenging in certain contexts since they are assigned with specific roles within the protocol stack and overriding them may not be a useful approach soon.

In order to leverage a fully functional service-centric approach, the following P4 features are implemented accordingly: (i) protocol fields, (ii) parser, (iii) control flow and (iv) match/action tables. Besides, P4 runtime API is utilized to populate and manage the match/action tables of the nodes.

6.3.1. System Design and Implementation Details

The capability of defining customized protocols, packet header fields, match/action tables and packet processing behavior is the main factor that makes P4 a promising mechanism for leveraging the service-centric approach within the network. Through the flexibility of determining the policies to be adapted within the network, it can be concluded that P4 is able to produce a generally applicable solution for the envisioned network architecture, embracing the necessary functionalities without reshaping the legacy TCP/IP protocol stack.

Although the P4 terminology considers packet header definitions and parser as separate implementations, they are interdependent functionalities in practice. The header definitions implemented through P4 represent the headers that can be available within packets and recognized by P4-loaded switches. A header implementation requires the following information: header name, number of fields, name and width of each field in terms of bits.

The proposed solution focuses on a new protocol design to eliminate the necessity of utilizing available header fields. Therefore, an application layer protocol is formed to create the opportunity of identifying service and sub-services through a separate packet header instead of disrupting the functions of Layer 2/3/4 protocols. The solution based on P4 identifies services with a 32-bit field and sub-services with another 32-bit at the application layer. The routing of network packets is carried out according to these fields. Although it is currently designed as supporting 2^{32} services and 2^{32} sub-services of each service, the flexibility provided by P4 brings in the convenience of widening these fields in case of a need.

Since P4-supported switches are initially not implemented with any functionality or protocol, the proposed solution also includes implementing the TCP/IP protocol stack. Although the primary operations are based on the application layer protocol designed by this study, the host operating systems are designed according to the

TCP/IP protocol stack. In order to provide complete communication between two ends and compatibility with the legacy communication solutions, Ethernet, IP, TCP and UDP packet headers and necessary fields are also implemented with P4.

P4 language defines a parser mechanism, which inspects a user-defined set of packet headers and produces a parsed packet representation. The flow tables then handle the parsed instance to carry out the necessary match and action operations. The proposed solution design embraces a parser implementation, which is summarized in Figure 6.18, that extracts the Ethernet header initially. Then, it parses the corresponding Layer 3 header according to the value of *type* field within this header. If the value is equal to $0x800$, the following header belongs to the IP protocol. Similarly, depending on the *protocol* field of the IP header, it either parses TCP or UDP header. Although the current implementation includes the parsing operations of only TCP/UDP headers, it can be improved to support other *protocol* values such as ICMP. Lastly, it parses the 64-bits application layer header and forwards the parsed representation match/action tables. The control flow specifies the sequence of the tables.

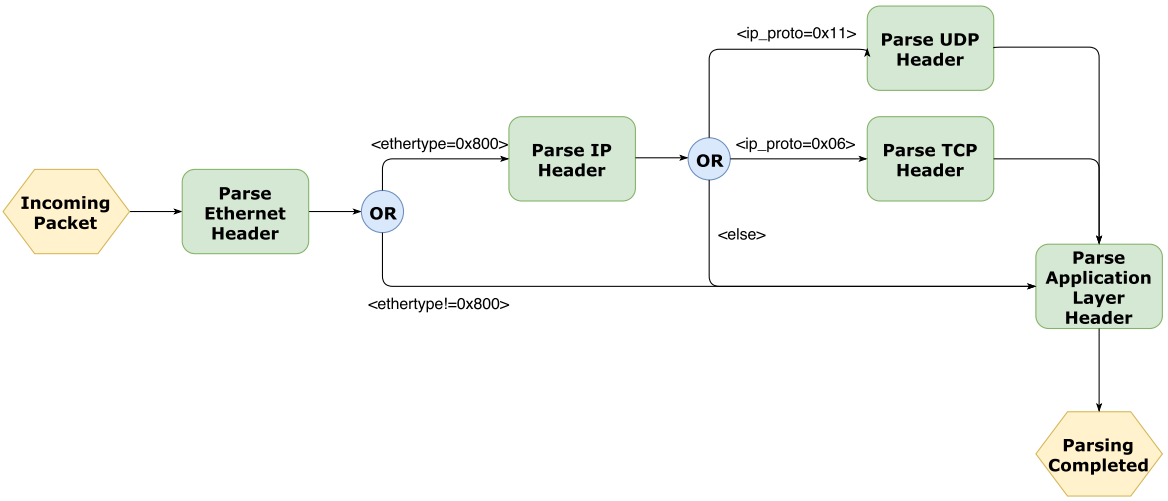

Figure 6.18. Stages of the parser implementation.

In addition to the parsing process, additional functionalities (e.g., checksum calculation) are also implemented within the scope of parser. The input fields and checksum calculation algorithm are specified within the parser, and the checksum field of each header is updated hop-by-hop.

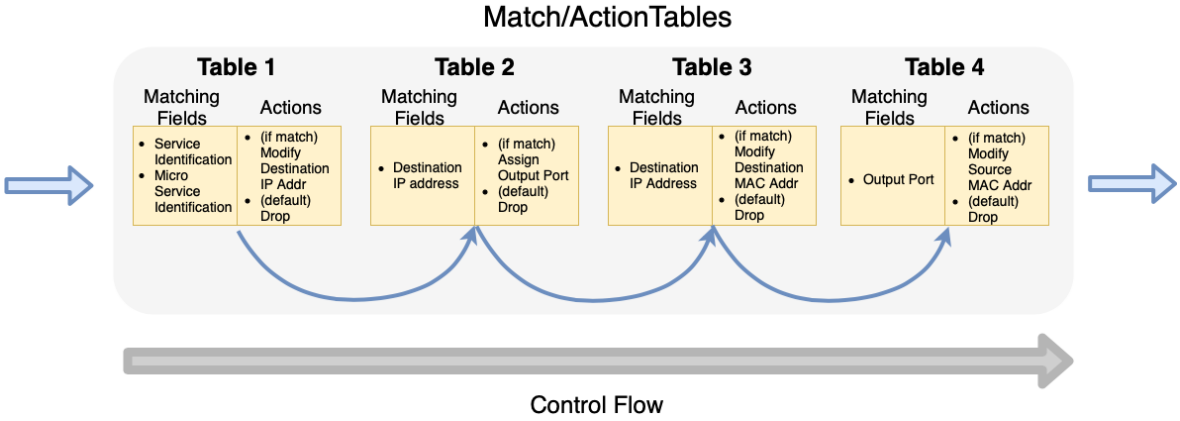

Figure 6.19. Implementation of the control flow and match/action tables.

One of the critical features that the P4 language accommodates is creating custom match/action tables, which is not available in OpenFlow. P4 provides the capability to define the desired number of tables, and each one has separate roles in processing the parsed representation of a network packet.

The proposed solution includes the implementation of four different match/action tables. After the parsed representation of a packet is generated at the ingress stage, the control flow acts as the main orchestrator to define the sequence of match-action tables. The first table retrieves the parsed information and checks the application layer fields, namely service and sub-service identifications. If there is a matched rule, it modifies the destination IP address of the packet and then forwards it to the second table defined by the control flow. The role of this table is to check the destination IP address and determine the necessary output port for routing the packet to the server that provides the desired service and sub-service. The association of the first two tables can be considered as a FIB (Forwarding Information Base) table implementation for the service-centric model. After the output port is determined, the third table modifies the destination MAC address. Lastly, the fourth table modifies the source MAC address and the packet is sent over the specified output port.

The details of the control flow operations, match/action tables and actions of each table are summarized in Figure 6.19. The routing of network packets within the service-

centric model is carried out according to the service and sub-service identifications, but the proposed solution deals with the IP addresses of the destinations. The main reason is that the operating system operations are based on the TCP/IP protocol stack, and if a server receives a packet with an invalid destination IP address, it drops the packet without forwarding it to the upper layers. Therefore, to comply with the available design, the destination IP address is modified by the initial table. An alternative solution may be providing two different implementations types. The core nodes may not deal with the IP addresses, but the switch at the last hop may modify the destination IP address to make it recognizable by the operating system of the server.

The P4 program code is compiled with a P4 compiler, such as *p4c-bmv2*, and it is loaded into the *target* such as a software-based P4 switch called *behavioral model (bmv2)*. *Target* is the term that represents the packet processing machine that can be implemented with P4 language. The last step that makes the proposed solution full-fledged is populating the match/action tables. The P4 Runtime API, which provides additional functionalities to communicate with the switch during runtime, is currently being developed by a specific workgroup within P4 consortium [141]. The Runtime API provides functionalities such as inserting, deleting and modifying table rules, reading counters, retrieving information specific to the tables, and defining registers.

According to the format defined by this API, the arguments required for adding a new entry are the name of the table, the value of the matching fields, name of the action to be applied in case of a match and the arguments required for carrying out specified action. In the first table, a flow rule added to this table needs to indicate the value of service and sub-service fields to be matched and the IP address of the server that is available to execute the demanded service instance. Similar to the first table, necessary entries are added to the following tables through P4 Runtime API.

6.3.2. Demonstration of the P4-enabled Service-centric Model

The P4 environment provides alternatives for experimenting with the implemented programs. The adopted methodology is compiling the implementation and loading the program to the *bmv2*, which can be run within the Mininet emulation environment. Even though Mininet is specifically utilized for OpenFlow-based experiments, it is possible to integrate a P4-loaded switch with a custom Mininet topology.

A service resembling face recognition is used to demonstrate service-centric operations of the proposed P4-based solution. The identification of this service is determined as *fcrg*, and the service routines are deployed over a server operating in the Mininet environment. On the other hand, in order to proliferate sub-service resolution, the feature extraction step of this service is separately taken into consideration and identified as *extr*. The P4 program is tested in Mininet with the topology in Figure 6.20. An end-user requests for the execution of the feature extraction sub-service of the face recognition algorithm. Instead of generic traffic generators, socket programming adds the necessary information to the application layer fields. The end-user application generates a request message through embedding *fcrg* and *extr* identifications into the packets. For routing the request to the corresponding server where the feature extraction subroutine is available, the necessary flow rules for each table described in the previous section are installed as follows:

- add_entry table1 0x66637267 0x65787472 service_dest_ip 10.0.1.10
- add_entry table2 10.0.1.10 set_nhop 2
- add_entry table3 10.0.1.10 set_dmac 00:04:00:00:00:01
- add_entry table4 2 rewrite_mac 00:aa:bb:00:00:01

The first flow rule is added to *table1* for matching with the service and sub-service identification and modifying the destination IP for specifying the IP address of the corresponding server. The service and sub-service identifications are converted to the hexadecimal information where 0x66637267 represents *fcrg* and 0x65787472 represents

extr. After the destination IP is modified as *10.0.1.10*, the output port is determined as 2 by *table2*, which is the switch port that is connected to the destination server. Then, *table3* modifies the destination MAC address and *table4* modifies the source MAC addresses according to the matching fields that are specified. After the server receives the request message, it executes the necessary instructions and generates a response to the client. In order to route the response packets to the client, a reverse path is constructed through similar flow rules.

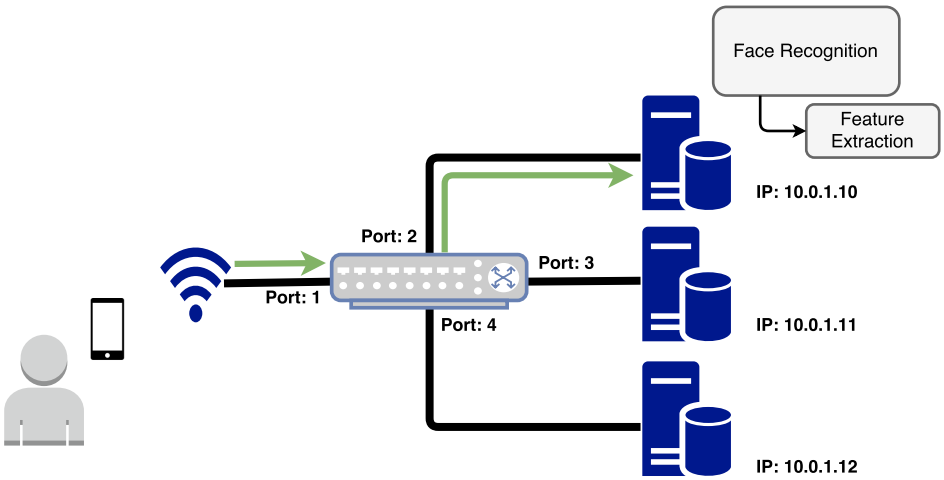

Figure 6.20. Topology for testing the P4 program.

Embedding 64-bits of identification into packets can introduce an overhead within the network. However, considering that video streams generates most of the overall IP traffic, the load generated by the service request packets with extra 8 bytes may be neglected. This overhead can also be quantitatively analyzed. For instance, an end-user may offload a face recognition task with an image file or a fall-risk assessment service with a set of sensory data. By taking into account that MTU (Maximum Transmission Unit) for Ethernet is 1500 bytes, integration of 8 bytes of information to the network packets results in approximately 0.06% of overhead. If an end-user requests a video streaming service, where the size and number of packets are small, the overhead rate may be higher. However, in this case, the response is composed of a video stream, which means that the overhead caused by request packets can be easily neglected compared with the response load.

7. CONCLUSION

The progress of making the typical end-user devices in daily life even smarter and improvements in the computational technologies render a novel set of services more feasible than ever. With the processing of data produced by multi-modal sensors embedded in these smart devices by machine learning-based applications, the requirements of the services and the expectations of the users have become more stringent.

Even though the cooperation of edge and cloud servers in a multi-tier computing system enables real-time interaction and satisfy the low latency requirement of novel services, such as fall risk assessment, the current operations of network infrastructure and protocol stack remain infeasible to provide flexibility for dynamic management and orchestration. Therefore, a service-oriented behavior becomes necessary instead of host-centric operations. However, it should be noted that the envisioned multi-tier structure with service-oriented functions should be compatible with the TCP/IP protocol stack. Besides its implementation, the requirements in such an environment should be analyzed, and the optimization problems should be studied in depth.

In this thesis, an SLA-aware task assignment problem is initially studied. In order to maximize the number of successfully handled task offload operations, two different optimization models are proposed: (i) an MINLP model and (ii) an MILP model with a linear approximation approach. Besides these solution proposals, a nearest-fit heuristic implementation is also provided to find near-optimal solutions for the problems where the optimization models suffer from scalability issues due to time and space complexity. In this study, two variant of the problem is studied. While the first one consists of undifferentiated services with various performance requirements, the second one aims to provide fairness among the service types. According to the results obtained through an extensive experiments, the proposed solutions can provide near-optimal solutions. While the MILP model enhances the quality of the solutions compared to the MINLP model, the nearest-fit heuristic is capable of finding good quality solutions in a very

short runtime. Additionally, an MILP model, a Lagrangian relaxation-based heuristic, and a greedy heuristic solution are proposed to address the problem of multi-tier edge computing structure design.

Even though these solutions are promising for addressing the low latency requirement of diversified services and expectations of both end-users and operators, the highly dynamic environment may fluctuate the performance experienced by the end-users. In order to ensure that all these requirements are satisfied, the network slicing concept should be integrated to assign dedicated virtual resources to each of the service types. Therefore, the second part of this thesis is composed of providing an optimal slicing approach with the optimal capacity reservation of both networking and computational resources. An MILP model is proposed with a discussion on the linearization approach to find the optimal slicing decisions. Similar to the previous studies, a heuristic solution is also proposed to provide a lower bound for the solutions and find reasonable quality solutions for large instances. With an extensive set of experiments, it is depicted that enabling the network slicing operations in an edge environment is critical for ensuring a certain level of performance by providing isolation among the slice instances. Additionally, the heuristic algorithm is shown to find near-optimal solutions for all cases in a short time.

Lastly, in addition to the optimization efforts, this thesis focuses on the implementation of service-centric behavior. Accordingly, two different solution approaches are proposed by using the programmable network paradigms. The first is based on SDN and OpenFlow concepts. A short-term solution is provided by utilizing the current TCP/IP protocol stack to implement the service-centric operations. On the other hand, the second solution proposal is long-term, and it implements the service-centric behavior using P4 capabilities. Since the P4-based solution handles the operations at the application layer with a customized protocol definition, it is compatible with the current protocol stack and provides a more realistic solution. In addition to the experiments with these solution proposals, the SDN-based approach is further evaluated with a fall-risk assessment service. Since this healthcare service demands low latency,

it is a good candidate for hosting in a multi-tier environment and orchestrating the task assignments with the SDN approach.

Although an extensive literature review has been carried out in terms of optimization and implementation of service-centric behavior in a multi-tier computing environment, another contribution of this thesis is the determination of various future works in this field. Firstly, since the optimization models provide an offline solution for the problem definitions of task assignment, system design and network slicing, future work is necessary for delivering online solutions. Secondly, the implementation work can be improved for covering multiple domains with a distributed control plane. Lastly, the network slicing problem can be extended to cover the RAN, especially targeting the 5G deployments to provide service assurance on a large scale.

REFERENCES

1. Weiser, M., "The Computer for the 21 st Century", *Scientific American*, Vol. 265, No. 3, pp. 94–105, 1991.

2. IEEE Standards Association, *IEEE Standards Activities in the Internet of Things (IoT)*, 2021.

3. Masek, P., R. Fujdiak, K. Zeman, J. Hosek and A. Muthanna, "Remote Networking Technology for IoT: Cloud-based Access for AllJoyn-enabled Devices", *2016 18th Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)*, pp. 200–205, IEEE, 2016.

4. Thread Group, *Thread*, 2021, `http://www.threadgroup.org/`, accessed in August 2021.

5. Open Interconnect Consortium, *Open Interconnect*, 2021, `http://openinterconnect.org/`, accessed in August 2021.

6. Stackowiak, R., A. Licht, V. Mantha and L. Nagode, "Internet of Things Standards", *Big Data and the Internet of Things*, pp. 185–190, Apress, Berkeley, CA, 2015.

7. Chen, X., L. Jiao, W. Li and X. Fu, "Efficient Multi-user Computation Offloading for Mobile-edge Cloud Computing", *IEEE/ACM Transactions on Networking*, Vol. 24, No. 5, pp. 2795–2808, 2016.

8. Gao, S., Y. Zeng, H. Luo and H. Zhang, "Scalable Control Plane for Intra-domain Communication in Software Defined Information Centric Networking", *Future Generation Computer Systems*, Vol. 56, pp. 110–120, 2016.

9. Braun, T., A. Mauthe and V. Siris, "Service-centric Networking Extensions", *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 583–590, 2013.

10. Sathiaseelan, A., L. Wang, A. Aucinas, G. Tyson and J. Crowcroft, "Scandex: Service Centric Networking for Challenged Decentralised Networks", *Proceedings of the 2015 Workshop on Do-it-yourself Networking: an Interdisciplinary Approach*, pp. 15–20, ACM, 2015.

11. Baktır, A. C., B. Ahat, N. Aras, A. Özgövde and C. Ersoy, "SLA-aware Optimal Resource Allocation for Service-oriented Networks", *Future Generation Computer Systems*, Vol. 101, pp. 959–974, 2019.

12. Baktir, A. C., A. Ozgovde and C. Ersoy, "Enabling Service-centric Networks for Cloudlets Using SDN", *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 344–352, 2017.

13. Baktir, A. C., A. Ozgovde and C. Ersoy, "Implementing Service-centric Model with P4: A Fully-programmable Approach", *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, 2018.

14. El Mougy, A., "On the Integration of Software-defined and Information-centric Networking Paradigms", *2015 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pp. 105–110, 2015.

15. Charpinel, S., C. A. S. Santos, A. B. Vieira, R. Villaca and M. Martinello, "SD-CCN: A Novel Software Defined Content-Centric Networking Approach", *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 87–94, 2016.

16. Salsano, S., N. Blefari-Melazzi, A. Detti, G. Morabito and L. Veltri, "Information Centric networking over SDN and OpenFlow: Architectural Aspects and

Experiments on the OFELIA Testbed", *Computer Networks*, Vol. 57, No. 16, pp. 3207–3221, 2013.

17. Zhang, L., A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named Data Networking", *ACM SIGCOMM Computer Communication Review*, Vol. 44, No. 3, pp. 66–73, 2014.

18. van Adrichem, N. L. and F. A. Kuipers, "NDNFlow: Software-defined Named Data Networking", *2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5, 2015.

19. Melazzi, N. B., A. Detti, G. Mazza, G. Morabito, S. Salsano and L. Veltri, "An Openflow-based Testbed for Information Centric Networking", *Future Network & Mobile Summit (FutureNetw), 2012*, pp. 1–9, 2012.

20. Nordström, E., D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Ko, J. Rexford and M. J. Freedman, "Serval: An End-host Stack for Service-centric Networking", *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pp. 85–98, 2012.

21. Signorello, S., R. State, J. François and O. Festor, "NDN.p4: Programming Information-centric Data-planes", *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*, pp. 384–389, 2016.

22. Wang, F., J. Xu, X. Wang and S. Cui, "Joint Offloading and Computing Optimization in Wireless Powered Mobile-edge Computing Systems", *IEEE Transactions on Wireless Communications*, Vol. 17, No. 3, pp. 1784–1797, 2018.

23. Zhang, J., W. Xia, F. Yan and L. Shen, "Joint Computation Offloading and Resource Allocation Optimization in Heterogeneous Networks With Mobile Edge Computing", *IEEE Access*, Vol. 6, pp. 19324–19337, 2018.

24. Zhang, K., Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan

and Y. Zhang, "Energy-efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks", *IEEE access*, Vol. 4, pp. 5896–5907, 2016.

25. Ceselli, A., M. Fiore, M. Premoli and S. Secci, "Optimized Assignment Patterns in Mobile Edge Cloud Networks", *Computers & Operations Research*, Vol. 106, pp. 246–259, 2019.

26. Liu, L., Z. Chang, X. Guo, S. Mao and T. Ristaniemi, "Multiobjective Optimization for Computation Offloading in Fog Computing", *IEEE Internet of Things Journal*, Vol. 5, No. 1, pp. 283–294, 2018.

27. Ren, J., G. Yu, Y. Cai and Y. He, "Latency Optimization for Resource Allocation in Mobile-edge Computation Offloading", *IEEE Transactions on Wireless Communications*, Vol. 17, No. 8, pp. 5506–5519, 2018.

28. Chen, M. and Y. Hao, "Task Offloading for Mobile Edge Computing in Software Defined Ultra-dense Network", *IEEE Journal on Selected Areas in Communications*, Vol. 36, No. 3, pp. 587–597, 2018.

29. Tong, L., Y. Li and W. Gao, "A Hierarchical Edge Cloud Architecture for Mobile Computing", *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pp. 1–9, 2016.

30. Malekloo, M.-H., N. Kara and M. El Barachi, "An Energy Efficient and SLA Compliant Approach for Resource Allocation and Consolidation in Cloud Computing Environments", *Sustainable Computing: Informatics and Systems*, Vol. 17, pp. 9–24, 2018.

31. Fan, Q., H. Yin, L. Jiao, Y. Lyu, H. Huang and X. Zhang, "Towards Optimal Request Mapping and Response Routing for Content Delivery Networks", *IEEE Transactions on Services Computing*, Vol. 14, No. 2, pp. 606–613, 2018.

32. Tiwary, M., D. Puthal, K. S. Sahoo, B. Sahoo and L. T. Yang, "Response Time

Optimization for Cloudlets in Mobile Edge Computing", *Journal of Parallel and Distributed Computing*, Vol. 119, pp. 81–91, 2018.

33. Smet, P., P. Simoens and B. Dhoedt, "QuLa: Queue and Latency-aware Service Selection and Routing in Service-centric Networking", *Journal of Communications and Networks*, Vol. 17, No. 3, pp. 306–320, 2015.

34. Agrawal, A., U. Vyas, V. Bhatia and S. Prakash, "SLA-aware Differentiated QoS in Elastic Optical Networks", *Optical Fiber Technology*, Vol. 36, pp. 41–50, 2017.

35. Abdenacer, N., H. Wu, N. N. Abdelkader, S. Dhelim and H. Ning, "A novel Framework for Mobile Edge Computing by Optimizing Task Offloading", *IEEE Internet of Things Journal*, Vol. 8, No. 16, pp. 13065–13076, 2021.

36. Li, G., J. Wu, J. Li, Z. Zhou and L. Guo, "SLA-aware Fine-grained QoS Provisioning for Multi-tenant Software-defined Networks", *IEEE Access*, Vol. 6, pp. 159–170, 2017.

37. Sun, J., Q. Gu, T. Zheng, P. Dong, A. Valera and Y. Qin, "Joint Optimization of Computation Offloading and Task Scheduling in Vehicular Edge Computing Networks", *IEEE Access*, Vol. 8, pp. 10466–10477, 2020.

38. Lee, K.-W., B.-J. Ko and S. Calo, "Adaptive Server Selection for Large Scale Interactive Online Games", *Computer Networks*, Vol. 49, No. 1, pp. 84–102, 2005.

39. Zhan, W., C. Luo, G. Min, C. Wang, Q. Zhu and H. Duan, "Mobility-aware Multi-user Offloading Optimization for Mobile Edge Computing", *IEEE Transactions on Vehicular Technology*, Vol. 69, No. 3, pp. 3341–3356, 2020.

40. Kuang, Z., Z. Ma, Z. Li and X. Deng, "Cooperative Computation Offloading and Resource Allocation for Delay Minimization in Mobile Edge Computing", *Journal of Systems Architecture*, Vol. 118, p. 102167, 2021.

41. Kao, Y.-H., B. Krishnamachari, M.-R. Ra and F. Bai, "Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing", *IEEE Transactions on Mobile Computing*, Vol. 16, No. 11, pp. 3056–3069, 2017.

42. Gu, L., D. Zeng, S. Tao, S. Guo, H. Jin, A. Y. Zomaya and W. Zhuang, "Fairness-Aware Dynamic Rate Control and Flow Scheduling for Network Utility Maximization in Network Service Chain", *IEEE Journal on Selected Areas in Communications*, Vol. 37, No. 5, pp. 1059–1071, 2019.

43. García-Valls, M., C. Calva-Urrego and A. García-Fornes, "Accelerating smart eHealth services execution at the fog computing infrastructure", *Future generation computer systems*, Vol. 108, pp. 882–893, 2020.

44. Wang, W., B. Liang and B. Li, "Multi-resource Fair Allocation in Heterogeneous Cloud Computing Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 26, No. 10, pp. 2822–2835, 2014.

45. Zhou, F., H. Sun, Z. Chu and R. Q. Hu, "Computation Efficiency Maximization for Wireless-Powered Mobile Edge Computing", *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2018.

46. Garcia-Saavedra, A., A. Banchs, P. Serrano and J. Widmer, "Distributed Opportunistic Scheduling: A Control Theoretic Approach", *2012 Proceedings IEEE INFOCOM*, pp. 540–548, 2012.

47. Phan, T. K., D. Griffin, E. Maini and M. Rio, "Utility-centric Networking: Balancing Transit Costs with Quality of Experience", *IEEE/ACM Transactions on Networking (TON)*, Vol. 26, No. 1, pp. 245–258, 2018.

48. Huang, J. and J. Bi, "A Proportional Fairness Scheduling for Wireless Sensor Networks", *Personal and Ubiquitous Computing*, Vol. 20, No. 5, pp. 695–703, 2016.

49. Kumar, A., S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila *et al.*, "BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing", *ACM SIGCOMM Computer Communication Review*, Vol. 45, No. 4, pp. 1–14, 2015.

50. Zheng, M., W. Liang and H. Yu, "Utility-based resource allocation in wireless-powered communication networks", *IEEE Systems Journal*, Vol. 12, No. 4, pp. 3881–3884, 2018.

51. Kolomvatsos, K. and C. Anagnostopoulos, "Multi-criteria Optimal Task Allocation at the Edge", *Future Generation Computer Systems*, Vol. 93, pp. 358–372, 2019.

52. Bahnasse, A., F. E. Louhab, H. A. Oulahyane, M. Talea and A. Bakali, "Novel SDN Architecture for Smart MPLS Traffic Engineering-DiffServ Aware Management", *Future Generation Computer Systems*, Vol. 87, pp. 115–126, 2018.

53. Elhoseny, M., A. Abdelaziz, A. S. Salama, A. M. Riad, K. Muhammad and A. K. Sangaiah, "A Hybrid Model of Internet of Things and Cloud Computing to Manage Big Data in Health Services Applications", *Future generation computer systems*, Vol. 86, pp. 1383–1394, 2018.

54. Chen, M., W. Li, Y. Hao, Y. Qian and I. Humar, "Edge Cognitive Computing Based Smart Healthcare System", *Future Generation Computer Systems*, Vol. 86, pp. 403–411, 2018.

55. Richart, M., J. Baliosian, J. Serrat and J.-L. Gorricho, "Resource Slicing in Virtual Wireless Networks: A Survey", *IEEE Transactions on Network and Service Management*, Vol. 13, No. 3, pp. 462–476, 2016.

56. Taleb, T., B. Mada, M.-I. Corici, A. Nakao and H. Flinck, "PERMIT: Network Slicing for Personalized 5G Mobile Telecommunications", *IEEE Communications*

Magazine, Vol. 55, No. 5, pp. 88–93, 2017.

57. Addad, R. A., M. Bagaa, T. Taleb, D. L. C. Dutra and H. Flinck, "Optimization Model for Cross-domain Network Slices in 5G Networks", *IEEE Transactions on Mobile Computing*, Vol. 19, No. 5, pp. 1156–1169, 2019.

58. Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2021, `http://www.gurobi.com`, accessed in August 2021.

59. Guan, W., X. Wen, L. Wang, Z. Lu and Y. Shen, "A Service-oriented Deployment Policy of End-to-end Network Slicing Based on Complex Network Theory", *IEEE Access*, Vol. 6, pp. 19691–19701, 2018.

60. Kasgari, A. T. Z. and W. Saad, "Stochastic Optimization and Control Framework for 5G Network Slicing with Effective Isolation", *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, 2018.

61. Han, B., J. Lianghai and H. D. Schotten, "Slice as an Evolutionary Service: Genetic Optimization for Inter-slice Resource Management in 5G Networks", *IEEE Access*, Vol. 6, pp. 33137–33147, 2018.

62. Agarwal, S., F. Malandrino, C.-F. Chiasserini and S. De, "Joint VNF placement and CPU Allocation in 5G", *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 1943–1951, 2018.

63. D'Oro, S., L. Bonati, F. Restuccia, M. Polese, M. Zorzi and T. Melodia, "Sl-Edge: Network Slicing at the Edge", *Proceedings of the Twenty-First International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, p. 1–10, Association for Computing Machinery, New York, NY, USA, 2020.

64. Hossain, M. A. and N. Ansari, "Energy Aware Latency Minimization for Network Slicing Enabled Edge Computing", *IEEE Transactions on Green Communications*

and Networking, pp. 1–1, 2021.

65. D'Oro, S., L. Bonati, F. Restuccia and T. Melodia, "Coordinated 5G Network Slicing: How Constructive Interference Can Boost Network Throughput", *IEEE/ACM Transactions on Networking*, Vol. 29, No. 4, pp. 1881–1894, 2021.

66. Leconte, M., G. S. Paschos, P. Mertikopoulos and U. C. Kozat, "A Resource Allocation Framework for Network Slicing", *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 2177–2185, 2018.

67. Wang, G., G. Feng, W. Tan, S. Qin, R. Wen and S. Sun, "Resource Allocation for Network Slices in 5G with Network Resource Pricing", *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pp. 1–6, 2017.

68. Sattar, D. and A. Matrawy, "Optimal Slice Allocation in 5G Core Networks", *IEEE Networking Letters*, Vol. 1, No. 2, pp. 48–51, 2019.

69. Halabian, H., "Distributed Resource Allocation Optimization in 5G Virtualized Networks", *IEEE Journal on Selected Areas in Communications*, Vol. 37, No. 3, pp. 627–642, 2019.

70. Wang, G., G. Feng, T. Q. Quek, S. Qin, R. Wen and W. Tan, "Reconfiguration in Network Slicing—Optimizing the Profit and Performance", *IEEE Transactions on Network and Service Management*, Vol. 16, No. 2, pp. 591–605, 2019.

71. Fantacci, R. and B. Picano, "When Network Slicing Meets Prospect Theory: A Service Provider Revenue Maximization Framework", *IEEE Transactions on Vehicular Technology*, Vol. 69, No. 3, pp. 3179–3189, 2020.

72. Zhang, Q., F. Liu and C. Zeng, "Adaptive Interference-aware VNF Placement for Service-customized 5G Network Slices", *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2449–2457, 2019.

73. Afolabi, I., J. Prados-Garzon, M. Bagaa, T. Taleb and P. Ameigeiras, "Dynamic Resource Provisioning of a Scalable E2E Network Slicing Orchestration System", *IEEE Transactions on Mobile Computing*, Vol. 19, No. 11, pp. 2594–2608, 2019.

74. Xiong, K., S. Leng, J. Hu, X. Chen and K. Yang, "Smart Network Slicing for Vehicular Fog-RANs", *IEEE Transactions on Vehicular Technology*, Vol. 68, No. 4, pp. 3075–3085, 2019.

75. Jin, H., H. Lu, Y. Jin and C. Zhao, "IVCN: Information-centric Network Slicing Optimization Based on NFV in Fog-enabled RAN", *IEEE Access*, Vol. 7, pp. 69667–69686, 2019.

76. Abidi, M. H., H. Alkhalefah, K. Moiduddin, M. Alazab, M. K. Mohammed, W. Ameen and T. R. Gadekallu, "Optimal 5G Network Slicing Using Machine Learning and Deep Learning Concepts", *Computer Standards & Interfaces*, Vol. 76, p. 103518, 2021.

77. Hua, Y., R. Li, Z. Zhao, X. Chen and H. Zhang, "GAN-powered Deep Distributional Reinforcement Learning for Resource Management in Network Slicing", *IEEE Journal on Selected Areas in Communications*, Vol. 38, No. 2, pp. 334–349, 2019.

78. Chergui, H. and C. Verikoukis, "Offline SLA-constrained Deep Learning for 5G Networks Reliable and Dynamic End-to-end Slicing", *IEEE Journal on Selected Areas in Communications*, Vol. 38, No. 2, pp. 350–360, 2019.

79. Shanbhag, S., N. Schwan, I. Rimac and M. Varvello, "SoCCeR: Services over Content-centric Routing", *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, pp. 62–67, 2011.

80. Fang, C., F. R. Yu, T. Huang, J. Liu and Y. Liu, "A Survey of Green Information-centric Networking: Research Issues and Challenges", *IEEE Communications*

Surveys & Tutorials, Vol. 17, No. 3, pp. 1455–1472, 2015.

81. Amadeo, M., C. Campolo, J. Quevedo, D. Corujo, A. Molinaro, A. Iera, R. L. Aguiar and A. V. Vasilakos, "Information-centric Networking for the Internet of Things: Challenges and Opportunities", *IEEE Network*, Vol. 30, No. 2, pp. 92–100, 2016.

82. Ahlgren, B., C. Dannewitz, C. Imbrenda, D. Kutscher and B. Ohlman, "A Survey of Information-centric Networking", *IEEE Communications Magazine*, Vol. 50, No. 7, pp. 26–36, 2012.

83. Latre, S., J. Famaey, F. De Turck and P. Demeester, "The Fluid Internet: Service-centric Management of a Virtualized Future Internet", *IEEE Communications Magazine*, Vol. 52, No. 1, pp. 140–148, 2014.

84. Verbelen, T., P. Simoens, F. De Turck and B. Dhoedt, "Adaptive Deployment and Configuration for Mobile Augmented Reality in the Cloudlet", *Journal of Network and Computer Applications*, Vol. 41, pp. 206–216, 2014.

85. Dinh, H. T., C. Lee, D. Niyato and P. Wang, "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches", *Wireless communications and mobile computing*, Vol. 13, No. 18, pp. 1587–1611, 2013.

86. Satyanarayanan, M., P. Bahl, R. Caceres and N. Davies, "The Case for VM-based Cloudlets in Mobile Computing", *Pervasive Computing, IEEE*, Vol. 8, No. 4, pp. 14–23, 2009.

87. Bonomi, F., R. Milito, J. Zhu and S. Addepalli, "Fog Computing and Its role in the Internet of Things", *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, 2012.

88. Pang, H. and K.-L. Tan, "Authenticating Query Results in Edge Computing", *20th International Conference on Data Engineering Proceedings*, pp. 560–571,

IEEE, 2004.

89. European Telecommunications Standard Institute, *Multi-access Edge Computing (MEC); Terminology*, 2016.

90. Firdhous, M., O. Ghazali and S. Hassan, "Fog Computing: Will it be the Future of Cloud Computing?", *Third International Conference on Informatics & Applications, Kuala Terengganu, Malaysia*, pp. 8–15, 2014.

91. Bahtovski, A. and M. Gusev, "Cloudlet Challenges", *Procedia Engineering*, Vol. 69, pp. 704–711, 2014.

92. Ahmadi, M., N. Khanezaei, S. Manavi, F. Fatemi Moghaddam and T. Khodadadi, "A Comparative Study of Time Management and Energy Consumption in Mobile Cloud Computing", *Control and System Graduate Research Colloquium (ICSGRC), 2014 IEEE 5th*, pp. 199–203, 2014.

93. Zhou, B., A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama and R. Buyya, "A Context Sensitive Offloading Scheme for Mobile Cloud Computing Service", *2015 IEEE 8th International Conference on Cloud Computing*, pp. 869–876, 2015.

94. Nir, M., A. Matrawy and M. St-Hilaire, "An Energy Optimizing Scheduler for Mobile Cloud Computing Environments", *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 404–409, 2014.

95. Li, Y., L. Sun and W. Wang, "Exploring Device-to-device Communication for Mobile Cloud Computing", *2014 IEEE International Conference on Communications (ICC)*, pp. 2239–2244, 2014.

96. Ha, K., P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies and M. Satyanarayanan, "The Impact of Mobile Multimedia Applications on Data Center Consolidation", *2013 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 166–176, 2013.

97. Cisco, *Cisco Annual Internet Report (2018–2023) White Paper*, 2020.

98. Beck, M. T., M. Werner, S. Feld and S. Schimper, "Mobile Edge Computing: A Taxonomy", *Proc. of the Sixth International Conference on Advances in Future Internet*, pp. 48–55, Citeseer, 2014.

99. Bonomi, F., R. Milito, P. Natarajan and J. Zhu, "Fog Computing: A Platform for Internet of Things and Analytics", *Big Data and Internet of Things: A Roadmap for Smart Environments*, pp. 169–186, Springer, 2014.

100. Thomas, D., "Cloud Computing-Benefits and Challenges!", *Journal of Object Technology*, Vol. 8, No. 3, pp. 37–41, 2009.

101. Dastjerdi, A. V., H. Gupta, R. N. Calheiros, S. K. Ghosh and R. Buyya, "Fog Computing: Principles, Architectures, and Applications", *Internet of things*, pp. 61–75, Elsevier, 2016.

102. NGMN Alliance, *Description of Network Slicing Concept*, 2016.

103. Open Networking Foundation, *Applying SDN Architecture to 5G Slicing*, 2016.

104. Marquez, C., M. Gramaglia, M. Fiore, A. Banchs and X. Costa-Pérez, "Resource Sharing Efficiency in Network Slicing", *IEEE Transactions on Network and Service Management*, Vol. 16, No. 3, pp. 909–923, 2019.

105. Elbayoumi, M., M. Kamel, W. Hamouda and A. Youssef, "NOMA-assisted Machine-type Communications in UDN: State-of-the-Art and Challenges", *IEEE Communications Surveys & Tutorials*, Vol. 22, No. 2, pp. 1276–1304, 2020.

106. Foukas, X., G. Patounas, A. Elmokashfi and M. K. Marina, "Network Slicing in 5G: Survey and Challenges", *IEEE Communications Magazine*, Vol. 55, No. 5, pp. 94–100, 2017.

107. Khan, H., P. Luoto, S. Samarakoon, M. Bennis and M. Latva-Aho, "Network Slicing for Vehicular Communication", *Transactions on Emerging Telecommunications Technologies*, Vol. 32, No. 1, p. e3652, 2021.

108. Mei, J., X. Wang and K. Zheng, "Intelligent Network Slicing for V2X Services Toward 5G", *IEEE Network*, Vol. 33, No. 6, pp. 196–204, 2019.

109. Kukkalli, H., S. Maheshwari, I. Seskar and M. Skorupski, "Evaluation of Multi-operator dynamic 5G Network Slicing for Vehicular Emergency Scenarios", *2020 IFIP Networking Conference (Networking)*, pp. 761–766, IEEE, 2020.

110. Farhady, H., H. Lee and A. Nakao, "Software-Defined Networking: A survey", *Computer Networks*, Vol. 81, pp. 79–95, 2015.

111. Teixeira, J., G. Antichi, D. Adami, A. Del Chiaro, S. Giordano and A. Santos, "Datacenter in a Box: Test your SDN Cloud-datacenter Controller at Home", *2013 Second European Workshop on Software Defined Networks (EWSDN)*, pp. 99–104, 2013.

112. Banikazemi, M., D. Olshefski, A. Shaikh, J. Tracey and G. Wang, "Meridian: An SDN Platform for Cloud Network Services", *Communications Magazine, IEEE*, Vol. 51, No. 2, pp. 120–127, 2013.

113. Jain, R. and S. Paul, "Network Virtualization and Software Defined Networking for Cloud Computing: A Survey", *Communications Magazine, IEEE*, Vol. 51, No. 11, pp. 24–31, 2013.

114. Tomovic, S., M. Pejanovic-Djurisic and I. Radusinovic, "SDN Based Mobile Networks: Concepts and Benefits", *Wireless Personal Communications*, Vol. 78, No. 3, pp. 1629–1644, 2014.

115. Malik, M. S., M. Montanari, J. H. Huh, R. B. Bobba and R. H. Campbell, "Towards SDN Enabled Network Control Delegation in Clouds", *2013 43rd An-*

nual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1–6, 2013.

116. McKeown, N., 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.

117. Hu, F., Q. Hao and K. Bao, "A survey on Software-defined Network and Openflow: From Concept to Implementation", *Communications Surveys & Tutorials, IEEE*, Vol. 16, No. 4, pp. 2181–2206, 2014.

118. Nguyen, X.-N., D. Saucez, C. Barakat and T. Turletti, "Rules Placement Problem in Openflow Networks: A Survey", *IEEE Communications Surveys & Tutorials*, Vol. 18, No. 2, pp. 1273–1286, 2016.

119. Open Networking Foundation, *P4 Language and Related Specifications*, 2021, `https://p4.org/specs/`, accessed in August 2021.

120. Open Networking Foundation, *OpenFlow Switch Specification Version 1.5.1*, 2014.

121. Sonmez, C., C. Tunca, A. Ozgovde and C. Ersoy, "Machine learning-based Workload Orchestrator for Vehicular Edge Computing", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 22, No. 4, pp. 2239–2251, 2020.

122. Knight, S., H. X. Nguyen, N. Falkner, R. Bowden and M. Roughan, "The Internet Topology Zoo", *IEEE Journal on Selected Areas in Communications*, Vol. 29, No. 9, pp. 1765–1775, 2011.

123. European Telecommunications Standard Institute, *ETSI GS MEC-IEG 004 v1.1.1 - Mobile-Edge Computing Service Scenarios*, 2015.

124. 5G PPP, *5G-PPP Use Cases and Performance Evaluation Models*, 2016, `https://5g-ppp.eu/white-papers/`, accessed in August 2021.

125. GAMS Development Corporation, *General Algebraic Modeling System (GAMS) Release 25.1.3. Washington, DC, USA*, 2013, `https://www.gams.com/products/introduction/`, accessed in August 2021.

126. Gleixner, A., L. Eifler, T. Gally, G. Gamrath, P. Gemander, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, F. Serrano, Y. Shinano, J. M. Viernickel, S. Vigerske, D. Weninger, J. T. Witt and J. Witzig, *The SCIP Optimization Suite 5.0*, Technical report, Optimization Online, December 2017.

127. Gleixner, A., L. Eifler, T. Gally, G. Gamrath, P. Gemander, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, F. Serrano, Y. Shinano, J. M. Viernickel, S. Vigerske, D. Weninger, J. T. Witt and J. Witzig, *The SCIP Optimization Suite 5.0*, ZIB-Report 17-61, Zuse Institute Berlin, December 2017.

128. Ahat, B., A. C. Baktır, N. Aras, İ. K. Altınel, A. Özgövde and C. Ersoy, "Optimal Server and Service Deployment for Multi-tier Edge Cloud Computing", *Computer Networks*, Vol. 199, p. 108393, 2021.

129. Lin, Z., K. Li, Y. Yang, F. Sun, L. Wu, P. Shi, S. Ci and Y. Zuo, "DRESIA: Deep Reinforcement Learning-enabled Gray Box Approach for Large-Scale Dynamic Cyber-Twin System Simulation", *IEEE Open Journal of the Computer Society*, 2021.

130. Carbonneau, R. A., G. Caporossi and P. Hansen, "Globally Optimal Cluster-wise Regression by Mixed Logical-quadratic Programming", *European Journal of Operational Research*, Vol. 212, No. 1, pp. 213–222, 2011.

131. Cococcioni, M. and L. Fiaschi, "The Big-M Method with the Numerical Infinite M", *Optimization Letters*, pp. 1–14, 2020.

132. NetworkX, *NetworkX*, 2020, `https://networkx.github.io/`, accessed in August 2021.

133. Confais, B., A. Lebre and B. Parrein, "An Object Store Service for a Fog/edge Computing Infrastructure Based on IPFS and a Scale-out NAS", *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pp. 41–50, 2017.

134. Cuervo, E., A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra and P. Bahl, "MAUI: Making Smartphones Last longer with Code Offload", *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62, ACM, 2010.

135. Amazon, *AWS Lambda*, 2021, `https://aws.amazon.com/lambda/details/`, accessed in August 2021.

136. Open Networking Foundation, *Mininet*, 2021, `https://opennetworking.org/mininet/`, accessed in August 2021.

137. Ryu SDN Framework Community, *Ryu*, 2021, `https://ryu-sdn.org`, accessed in August 2021.

138. Baktir, A. C., C. Tunca, A. Ozgovde, G. Salur and C. Ersoy, "SDN-Based Multi-Tier Computing and Communication Architecture for Pervasive Healthcare", *IEEE Access*, Vol. 6, pp. 56765–56781, 2018.

139. Pfaff, B., J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The Design and Implementation of Open vSwitch.", *NSDI*, pp. 117–130, 2015.

140. Zeng, Y., P. H. Pathak and P. Mohapatra, "Throughput, Energy Efficiency and

Interference Characterisation of 802.11ac", *Transactions on Emerging Telecommunications Technologies*, Vol. 28, No. 2, p. e2946, 2017.

141. The P4.org API Working Group, *P4Runtime Specification*, 2021, `https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html`, accessed in August 2021.