On Improving Handover Delay in Software-Defined Mobile Networks

by

Modhawi Salah Alotaibi

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for
the Doctorate in Philosophy degree
in Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

The Software-Defined Networking (SDN) paradigm has become essential in tackling several issues and challenges in conventional networking, especially in mobile wireless networks. In order to realize the benefits brought by SDN to mobility management, we study the effects of SDN in conjunction with the OpenFlow protocol on the handover procedure. In this new setting, the completion of a handover procedure still requires the exchange of signaling messages between the control and data planes through OpenFlow channels, especially in the case of a hard handover, where the "breakage" in an on-going session is correlated to the exchange of management and reconfiguration messages. Consequently, the handover still suffers delay. One of the main causes of delay is the processing delay of those signaling messages. Three of the main factors contributing to the processing delay are the handling procedures, forwarding mechanisms of the network's devices, and the load on controllers. Therefore, in this work, we target these three factors. First, we design an effective handling procedure of the exchange of signaling messages in the OpenFlow channels between the controllers and switches. Second, we provide a forwarding mechanism that is designed to reduce the number of switches involved in configurations of the data plane. Third, we propose a load balancing mechanism to mitigate the over-loading state that may cause longer delay periods. As for our first target, we provide an analytical model, which gave us the ability to model two handover-related OpenFlow messages. Following our analysis, we propose a novel solution to make handover more efficient and less interruptive. Furthermore, we study our solution in the Long Term Evolution (LTE) architecture. As for our second target, we take network heterogeneity into account, and accordingly, we provide an efficient approach using the MobileIP protocol, which shows improvement in decreasing total processing delay. Regarding the third target, we employ "vertical mobility" in our load balancing framework to minimize the maximum response time; thus, the users' mobility-related procedures can be completed with less delay.

Acknowledgements

First and foremost, I would like to thank Allah, the almighty, for providing me with the strength and perseverance to complete this thesis, to overcome obstacles, and for blessing me with a fantastic support system, both in my academic and personal life.

I would like to take this opportunity to express my deepest gratitude and appreciation to Professor Amiya Nayak for his guidance, comments, support, and engagement throughout this learning process.

Also, I would like to express my very great appreciation, gratitude, and love to my husband Fayez, whose sacrifices, support, and encouragement have made the completion of this thesis possible.

I will eternally be grateful for having my two angels Mohammed and Anas, who have been the light, joy, and warmth to my heart.

My words cannot express the gratitude I have for my mother, sisters, and brothers, who have supported me throughout this process. Their prayers, encouragement, and love have always enlightened my journey and pushed me through hardships.

Moreover, I would like to thank my friends, whom I consider sisters (Sahar, Abeer, Hanadi, Noura, Reham, Rouwaida, Ohoud, Shaima, Hessa), for their encouragement, empowerment, and simply being amazing friends.

Last but not least, I sincerely thank the faculty members and staff of the College of Computer Science and Engineering at Taibah University. I am particularly grateful for the help and cooperation given by Dr. Ghadah Alharbi, Dr. Nahla Abed, Dr. Ibrahim Alfadli, and Mr. Nawaf Almuhammadi.

To my father:

Even if he passed away, his dream should come to life.

وإنْ رَحَل عنِ الحياة، لابد لِحُلْمهِ أنْ يَحيا.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AAA	Authentication Authorization and Accounting
AHP	Analytic Hierarchy Process
AP	Access Point
API	Application Programming Interfaces
BER	Bit Error Rate
BS	Base Station
CI	Consistency Index
CIR	Carrier-t-Interference Ratio
CR	Consistency Ratio
EPC	Evolved Packet Core
GPS	Global Positioning System
HSS	Home Subscriber Server
IoT	Internet of Things
IP	Internet Protocol
LTE	Long Term Evolution
MHE	Mobility Handling Entity
MIP	Mobile IP Protocol
MME	Mobility Management Entity
MVA	Mean Value Analysis
OSI	Open Systems Interconnection model
PASTA	Poisson Arrivals See Time Averages

PC	Personal Computing
PMIP	Proxy Mobile IP
PoA	Point of Attachment
P-GW	Packet Data Network Gateway
QoS	Quality of Service
RAN	Radio Access Network
RC	Random Consistency Index
RSS	Received Signal Strength
SDN	Software-Defined Networking
SIR	Signal-t-Interference Ratio
SPoF	Single Point of Failure
SRAM	Static Random Access Memory
S-GW	Serving Gateway
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VNI	Cisco Visual Networking Index
VoIP	Voice over IP
WAN	Wide Area Network
WLAN	Wireless Local Area Network

List of Symbols

k_i	Number of jobs at node i
λ_{ih}	Arrival rate of *Port-status* messages by i
λ_{ip}	Arrival rate of *Packet-in* messages by i
Γ_{ih}	Total net arrival rate of port changes at i
v_j	*Port-status* configuration flag
Γ_{ch}	Total net arrival rate of port changes at controller
Γ_{cp}	Arrival rate of *Packet-in* to controller
μ_{ip}	Service rate of node i for *Packet-in* messages
μ_{ih}	Service rate of node i for *Port-status* messages
μ_{cr}	Service rate at controller
q_i^{nf}	Probability of *Packet-in* sent to controller from i
p_{ij}	Probability of traffic between any two nodes i, j
u_j^i	Updating flag upon *Packet-in*
T_{ir}	Sojourn time of an event at i
W_i^c	Interaction time between i and controller upon a *Port-status*.
$T_{i(prop)}^c$	Propagating time of a *Port-status* i to controller
$D_{tot}4$	Total delay of handover procedure.
ρ_i	Total utilization of switch i
ρ_c	Total utilization of controller
V_T	Average throughput per user
f_1	Distance function

Hs	Home switch
Fs	Foreign switch
w_1	Coefficient for a distance function
R_A	Region of a set of switches.
M	Number of heterogeneous domains
C_i	Wireless network managing controller
α_i	Maximum number of control requests handled by C_i
β_i	Decay factor
N	Number of switches
X	Switch-controller adjacency matrix
Z_{s^i}	Switch-user adjacency matrix
$l_{s^i_j}$	Number of channels provided by s^i_j
K	Total number of users
Y	Controller-users adjacency matrix
$\xi_{s^i_j}$	Control traffic reported to C_i by switch s^i_j
θ_i	Load on C_i
UL	Under-loaded set
NL	Normally-loaded set
OL	Over-loaded set
R_{C_i}	Average response time of C_i
$\bar{U}$	Set of attached users
MN	Mobile node
CN	Correspondent node
MN^i_e	Edge users
MN^i_{ne}	Non-edge users
U^i_h	Set of edge candidate users
k	Number of users in one domain
H	Number of candidate users

r^i	Remaining capacity of C_i
α_{th_i}	Pre-defined threshold of C_i
$\overset{+}{\alpha_{th}}$	Maximum threshold
$\overset{-}{\alpha_{th}}$	Minimum threshold
Δ	AHP pair-wise matrix
V	Normalized Eigenvector
W	Relative weight matrix
$\overrightarrow{P_h}$	Candidate networks/controller vector
$\overrightarrow{R_h^m}$	Remaining capacity vector of candidate controllers

Chapter 1

Introduction

Nowadays, we witness the wide-spread availability of mobile hand-held devices such as smartphones and tablets. The ease and convenience provided by these devices lead to an extensive and explosive growth of mobile traffic. According to the Cisco Visual Networking Index (VNI) report that was published in February 2019, the global mobile data traffic has grown 18 times over the last five years and will increase by 46% in the next five years. Interestingly, it is expected that smartphone traffic will surpass PC traffic by 2022, where smartphone traffic will be accounted for 44% of the total IP traffic compared to PC traffic which is only 19%. Moreover, the expected traffic from wireless and mobile devices will reach 71% of the total IP traffic by 2022 [11].

As the traffic generated by mobile devices grows, the attention drawn towards designing better solutions for mobile computing increases as well. Mobile computing refers to the communication process with a node while mobility is in effect. This involves keeping track of the mobile node's location as well as maintaining on-going connections from breakage. An interesting way to describe that is the so-called "computing on the go" [91]. In such an environment, the networks are referred to as mobile networks, which can be telecommunication networks such as the third generation (3G), fourth generation (4G), and the anticipated fifth generation (5G), or other wireless networks such as IEEE 802.11 (Wi-Fi).

1.1 Mobility Management

One important aspect of mobile computing is mobility management. Mobility management can be defined as the mechanisms considered to alleviate or at least minimize disruption while communicating nodes change their physical channel, access network or communication protocol [47]. With the explosive use of real-time multimedia applications, mobility management has to be carefully designed to comply with the users' expected Quality of Service (QoS). The "mobility management" term implies several functions including location tracking, paging, managing resources, and more importantly handover.

Handover, or handoff, is the procedure of handing a mobile node's traffic from one point of attachment to another, whether changing connection between different cells or different technologies. This implies reserving resources to ensure a sessions continuity. The handover procedure can be divided into two phases: handover decision and handover execution. The handover decision is correlated to the Radio Access Network (RAN), where there are several factors that have to be taken into account. Within the same type of access networks, the mobile node's received signal strength (RSS) plays the main role in making that decision. On the other hand, within different access networks, some of the factors that are responsible for that decision are the cost of service, velocity of the mobile node, and user preferences [52]. After the decision has been made, the handover execution takes place; this phase involves three stages: preparation, execution, and completion [59]. Mainly, those three stages are accomplished by exchanging control messages, or so-called signaling messaging. Note that the focus of this thesis is the second phase, which does not consider the involvement of RAN.

On another note, the handover process can be categorized in term of releasing resources into *soft-handover* and *hard-handover*. The *hard-handover* can be described as the "break before make" type, where there is actual breakage and the resources reserved for a mobile node is released before making a new connection. On the contrary, the *soft-handover* allows

a mobile node to reserve resources from both connections, the old and new ones, forming so-called "make before break" handover [96].

The Diversity of wireless technologies, also known as "network heterogeneity", imposes the challenge of designing effective mechanisms to integrate different access technologies, protocols and service demands [13]. Accordingly, the handover procedure can be further categorized into *Vertical handover* and *Horizontal handover*, where handover is triggered between heterogeneous networks and homogeneous networks, respectively.

1.2 Motivation

There are several key benefits of SDN that can be advantageous to the handover procedure; for instance, automated management, low-cost forwarding devices, network technology heterogeneity support, virtualization and segmentation support, etc. Behind all these key benefits is the fine granularity forwarding imposed by SDN, which is considered one of its design strengths. It provides end-user-specific flow forwarding and in return, can enforce policies per user, device, session, application, etc.; therefore, it can provide a seamless handover [44].

SDN provides the ability to handle mobility more efficiently because it handles the mobility per flow, not only per user [59]. In order to realize the benefits brought by SDN to mobility management, we study the effects of SDN in conjunction with OpenFlow protocol on the handover procedure. However, in this new setting, the handover still suffers from delay due to the exchange of OpenFlow signaling messages.

In light of that, we got the following questions. Considering the rapid and extensive growth of mobile traffic, how do we ensure session continuity for mobile users? How to tolerate moving between cells or technologies without interruption? How do we execute efficient handover? We ask how to prevent delay and high latency while a moving node

roams between cells and technologies? Does load balancing have an impact on handover efficiency? How does the SDN paradigm improve handover and provide better services? Motivated by these intriguing questions, we started researching the effect of SDN on handover.

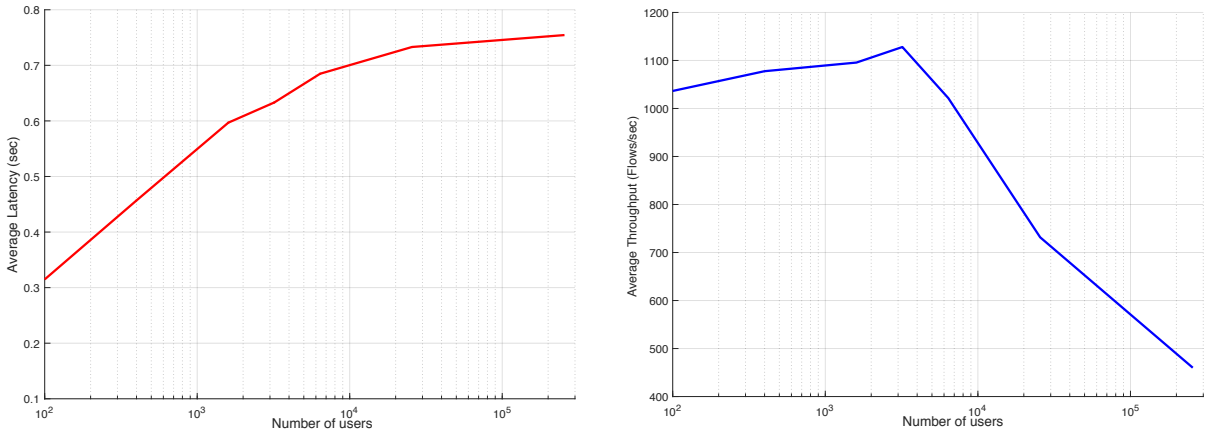
1.3 Problem Statement

Handling handover using the aspects of SDN inherits both the pros and cons of SDN. The SDN paradigm can be very beneficial to mobility management in terms of mitigating the need for tunneling, which burdens the network with signaling overhead, bringing down the high operational costs of proprietary hardware, and efficient resource utilization [59, 79]. On the other hand, handover management still suffers delay that can affect users' perceived QoS. Indeed, the traffic for mobility management is massive, and the cost is enormous due to connection establishment/release, handover, and tracking area update events [80].

Therefore, in our research, we study mobile networks, and we focus on the handover of hosts between different switches. Specifically, our focus is on minimizing handover delay by targeting one of its causes, the delay caused by processing control messages. The completion of a handover procedure requires the exchange of signaling messages between the control and data planes, especially in the case of hard handover, where the interruption in a live session correlates to the exchange of management and reconfiguration messages [100].

Interestingly, it has been showed that an SDN-based solution is more suitable for less-sensitive to latency applications [30]. However, we know for a fact that the handover is latency sensitive, and the primary goal here is to minimize handover latency. This fact motivated us to further analyze the main contributors to delay.

The authors of [70] studied the causes for overall latency in SDN-based mobile networks. They linked latency to two major factors, propagation delay and processing delay.



(a) Average controller latency.

(b) Average controller throughput.

Figure 1.1: The impact of increasing number of users on a controller performance.

Propagation delay is described as the time taken for the data to be transmitted between the controller and the switches; placement algorithms can play an important role here. Processing delay describes the delay within the controller or a switch. Marqueza et al. observed that the number of hops influences the increase in processing latency but that the real cause is buffering handling. They also argued that the load of the requests plays a significant role on processing delay. This remark was backed up by another study by Tootoonchian et al. in [104], where they linked a controller load and its reactivity to requests. Intrigued by their remark, we tested the performance of a single controller against an increasing number of connected users using CBench tool [94]. The measured effect was shown in two metrics, throughput and latency, as in Figure 1.1. We can see that the controller performance worsens as the intensity of connected users increases. Indeed, the timely delivery of a huge number of control messages in SDN is critical. Therefore, we need to dynamically adjust and redistribute the load among controllers to maintain acceptable levels of QoS regardless of sudden unpleasant network conditions, such as congestion and failure [88].

In light of the results from [70, 104], we got motivated to investigate the processing delay factor by studying the following.

- The handling procedures of the exchange of signaling messages in the OpenFlow channels between the controllers and switches.
- The forwarding mechanisms, which reduce the number of switches involved in configurations in the data plane.
- The impact of over-loading controller buffer on processing latency and thus, on handling signaling messages.

We, therefore, highlight that our research focuses on improving SDN-enabled hard handover in terms of reducing the delay caused by processing signaling messaging.

1.4 Contributions

In this thesis, we study the impact of SDN on the handover procedure; specifically, we focus on the hard handover. In our research, we target the delay caused by exchanging signaling messages during the handover execution which lead us to propose solutions to minimize that type of delay. Our contributions can be summarized as follows.

- Studying and then quantifying the delay of handover-related OpenFlow messages in order to identify the performance measures as well as the underlying challenges. For our analysis, we provide an analytical model, this allowed us to model two handover-related OpenFlow messages. To the best of our knowledge, no previous work has modeled OpenFlow messages other than Packet-in messages. Following our analysis, we propose a novel solution to make handover more efficient and less interruptive.

- Applying our solution to a cellular network and compare it to an existing solution [24]. We show that in normal traffic conditions, our solution can decrease the handover delay as much as 20%.

- Considering heterogeneous networks, we propose a hierarchical architecture that provides better scalability over the centralized architecture and less delay compared to the distributed approach. We have modified a solution proposed by a previous study [110] to solve the binding-cache placement problem in MobileIP. We use a heuristic approach. Combing the proposed hierarchical architecture and our version of the binding-cache placement solution, we test and analyze our approach.

- We have proposed a novel approach that mainly aims at vertically handing over some edge users, considering some context information regarding the users and controllers. Consequently, a controller's load drops and thus its response time to any mobility-related procedure decreases.

1.5 Scholarly Achievements

In the process of completing this work, the following papers have been submitted or published.

Conference Papers:

- M. Alotaibi, and A. Nayak. A distributed approach to improving EPC controller performance. In *Proceedings of the 86th IEEE Vehicular Technology Conference*, 2017.

- M. Alotaibi, H. Lu, and A. Nayak. A hierarchical approach to handle inter-domain mobility in SDN-based networks using MobileIP. In *Proceedings of the 25th IEEE International Conference on Telecommunications*, 2018.

- M. Alotaibi, D. Liu, and A. Nayak. Improving the response time of software-defined networking controllers based on vertical mobility, *IEEE Computer Society Signature Conference on Computers, Software and Applications*, 2019. (*Submitted*)

Journal Papers:

- M. Alotaibi, A. Helmy, and A. Nayak. Modeling handover signaling messages in OpenFlow-based mobile software-defined networks. *Journal of Computer Networks and Communications*,vol. 2018, 14 pages, 2018.

- M. Alotaibi, and A. Nayak. Linking handover delay to load balancing in SDN-based heterogeneous networks. *Computer Communications*, 2020. (*Submitted*)

1.6 Thesis Outline

The remainder of the thesis is organized as the following.

- In Chapter 2, we present background information on SDN and mobility management; then, we review the literature on topics related to our research problem. We begin by reviewing some papers that address different aspects of handover and SDN. Then, we go through some papers that study the OpenFlow messages modeling using queueing theory. We also survey some papers that consider the mobility management solutions for the LTE architecture taking into consideration the handover metrics. We then survey papers that study the merge of MobileIP protocol and SDN. Lastly, we review some of the existing scholars that proposed load balancing methods for SDN-based networks.

- In Chapter 3, we tackle one of the processing delay contributors mentioned earlier, the handling procedure of signaling messages. In this chapter, we propose a novel

idea of dedicating a specific type of OpenFlow messages to the handover procedure. Then, we provide an analytical model based on the queueing theory to realize our idea.

- In Chapter 4, we apply our solution to an LTE architecture to handle handover procedures. Moreover, we carry out a comparative analysis of an existing architecture [24] and our proposed architecture.

- In Chapter 5, we address our problem by targeting another contributing factor to the processing delay of signaling messaging, which is the number of to-be-configured switches upon mobility, or what so referred to as the binding-cache placement problem. We describe in detail this factor considering heterogeneous networks and propose a novel idea to minimize its effect.

- In Chapter 6, we study the relevancy between a controller's load and handover delay. We show that an over-loading state can prolong delay, so as a countermeasure, we avoid reaching that state by applying a load balancing mechanism. We propose a management framework that includes three main aspects. Firstly, we identify candidate users based on their context information. Secondly, we propose a novel mechanism that reduces the frequency of load informing between multiple controllers, and hence, reducing processing and communication overhead. Finally, after the candidate users are determined, we optimize the decision problem on the selection among candidate heterogeneous networks.

- In Chapter 7, we conclude our thesis by summarizing all the results, presenting our future work and discussing some open problems related to our topic.

Chapter 2

Background and Related Work

2.1 Background

In this section, we present background information regarding the current networking architecture, as opposed to the SDN architecture.

2.1.1 Conventional Networking

Current computer networks are built out of an enormous number of forwarding devices and middle-boxes scattered on large-scales. Each device has some logic and local intelligence to perform certain roles. The most well-known and standard conceptual model that describes how conventional network's devices work is the Open Systems Interconnection (OSI) model. In spite of that, this network's architecture has reportedly been suffering from multiple issues that get exaggerated with the increase in traffic demands and with new applications and protocols requirements. For instance, conventional networking devices are built with proprietary hardware and software from a vendor, leading to what we refer to as "vendor lock-in"; hence, multi-vendor solutions are difficult to realize[112]. Additionally, the high cost due to the hardware devices is unavoidable [54, 71]. Another

issue is the inflexible adaptability to network changing conditions, which is a significant challenge that burdens network managers. Moreover, scaling and managing due to the need to configure a network's devices in a low-level manner are two challenging tasks. On a global scale, another challenge is "Internet ossification," which means the difficulty of Internet evolution in terms of physical infrastructure and protocol installation [81].

Regarding the mobile and wireless networks, the issues are amplified due to the inherent problems in these networks, such as poor resource utilization, heterogeneity of wireless networks, and tremendous mobile traffic growth and demands [13, 117].

2.1.2 The Software-Defined Networking Paradigm

Recently, the concept of Software-Defined Networking has emerged as a promising paradigm to solve some of the issues mentioned above and offers solutions such as reducing upgrade costs and using network resources efficiently [71, 120]. Some of the problems in the current networks are inflexibility, difficulty of scaling and management, and high cost [54]. Therefore, the SDN paradigm has stepped forward to solve many of those issues. The concept of SDN is based on abstracting intelligence and decision-making from network forwarding devices to a higher level. Unlike the conventional networking, the SDN paradigm divides the network architecture into three planes: application, control, and data, which makes it possible to program and configure the network dynamically and globally. See Figure 2.1.

SDN provides three main benefits: programmability, centralized intelligence, and abstraction [83]. First, programmability reduces higher costs of proprietary hardware and accelerates adoption of new inventions as well as compatibility between different vendors' hardware [120]. Second, centralized intelligence optimizes overall network performance and enforces different levels of granularity. In fact, SDN allows better traffic management by enforcing fine-grained policies, which in turn provides flexible and dynamic traffic routing [102]. Thirdly, the abstraction aspect of SDN offers easier deployment of policies

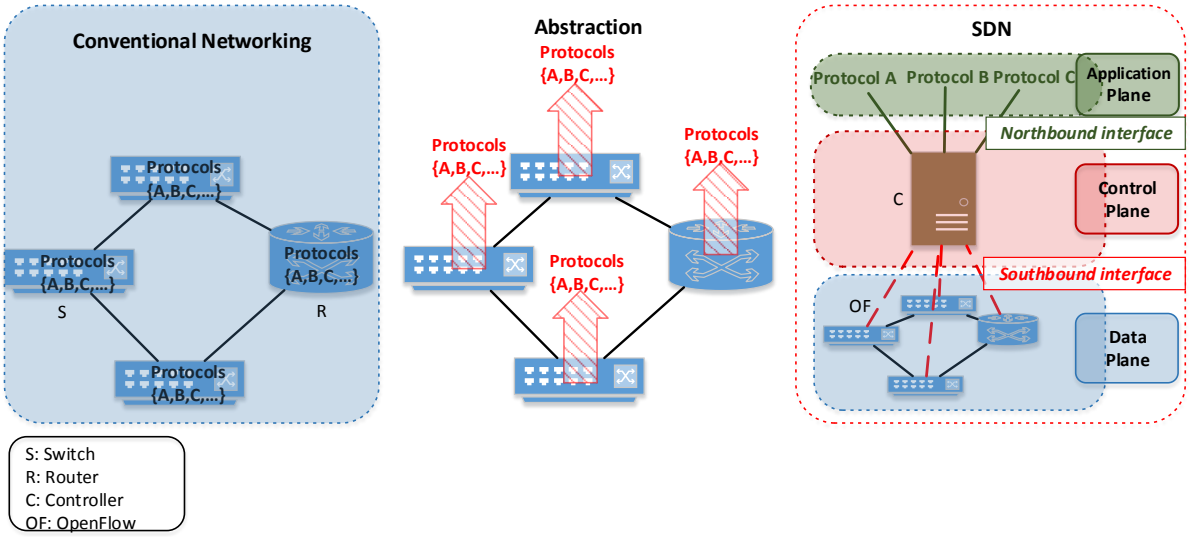


Figure 2.1: Conventional networking concept vs. SDN concept.

and protocols, thus allowing remote control of network forwarding devices, better management of faults, traffic control, and network virtualization [83]. In fact, SDN facilitates the management of large-scale congested networks such as datacenters and cloud infrastructures. For instance, on Aug 21, 2017, Cisco announced their plan to integrate their SDN product, called Application Centric Infrastructure (ACI), into the top three major public cloud platforms in the market: Amazon Web Services, Microsoft Azure, and Google Cloud Platform [25]. This incentive shows the expected widespread of the SDN concept in the future.

The fine-granularity forwarding proposed by SDN reduces the communication overhead between the data plane and control plane [54]. Packet-level granularity, as in conventional networking, causes extra delay since a forwarding device must make a decision regarding each packet. However, in flow-based granularity in SDN, the controller must make a decision about the first packet of a new flow, and then all subsequent packets follow the same decision [81]. Due to the SDN propitious claims, it has gained massive attention in both academia and the industry. This emerging technology has been studied extensively in the literature [33, 34, 44, 56, 81].

2.1.2.1 SDN Architecture Planes

Each plane in an SDN architecture has its functionalities. These three layers can communicate with each other via Application Programming Interfaces (API), namely *northbound* and *southbound* APIs. As Figure 2.1 depicts, the *northbound* interface connects the application layer to the control layer, while the *southbound* interface connects the control and data layers.

- **Application Plane**

In the application plane, the programmability aspect of SDN is realized. This layer consists of a pool of network services and functions such as monitoring, load balancing, security, traffic shaping, routing, etc. These services have access to the data plane state through the control plane, and thus they set the network policies and operation logic [33].

- **Control Plane**

The control plane is the network operating system, the orchestration platform that acts as an intermediary between the application and data layers. It transfers the policies obtained from a service to be applied by network devices [112].

- **Data Plane**

This plane consists of the network forwarding devices, such as switches and routers. Unlike conventional network devices, in SDN, these devices are simple with no embedded intelligence. They act only per instructions installed by the control plane through a *southbound* interface such as OpenFlow [33].

2.1.2.2 OpenFlow Protocol

According to the design of SDN, the communication between the control and data planes has to go through what is referred to as a *southbound* interface. OpenFlow protocol

is the most well-known and is also the industry standard southbound protocol in SDN-based networks. It was proposed by McKeown et al. [71], and is now maintained by Open Networking Foundation (ONF). Despite being the most well-known, OpenFlow is not the only southbound protocol; in fact, there are other alternative protocols that have been used widely such as the Network Configuration Protocol (NetConf) [32], beacon [82], Maestro [20], and DevoFlow [72].

Due to the fact that OpenFlow is open-source and the majority of existed open-source controllers such as Pox [6], Ryu [7], Floodlight [2], and ONOS [5] revolve around the structure of OpenFlow, it has an established community. That is reflected by the magnitude volume of academic proposals that have been built on top of OpenFlow. Therefore, due to its availability, we chose to work with it in this thesis.

The OpenFlow protocol manages the interaction between the data and control planes through the use of particular messages and the configuration of the flow tables of switches. Note that we use the words switch, node, and forwarding device, interchangeably referring to the data plane devices, throughout this thesis. A controller extracts a packet's header and uses fields such as source/destination IP/MAC addresses, TCP/UDP port numbers for matching, and hence installing flow rules and applying actions on the matched packets. In OpenFlow, there is a set of actions: drop, forward to port, rewrite, and send to controller [84]. The flow rules can be fine-grained, which are applied on *microflows*, or coarse-grained , which are applied on *macroflows* [66]. A *microflow* matching requires matching all fields resulting in matching per flow. On the other hand, a *macroflow* matching relies on wildcards that have "do not care" empty fields resulting in matching aggregate flows [107]. It is known that *wildcards* rely on Ternary Content Addressable Memory (TCAM), which is expensive, power-hungry, and takes up a large space of the silicon-space [66]. Note that *microflows* use Static Random Access Memory (SRAM), which is larger than TCAM, but in a hardware-wise perspective is less demanding.

The installed rules can be bounded by timers, namely *hard timeout* and *soft timeout*. After the *hard timeout* period, the switch is triggered to delete the corresponding rule, while the *soft timeout* specifies the inactivity period after which the rule should be erased [84].

The communication between a forwarding device and the controller takes place through an OpenFlow channel where only certain types of messages are standardized [84]. It is a bi-directional channel so that both parties can initiate messages. In OpenFlow, the messages initiated between a device and the controller can be either asynchronous messages or symmetric messages. The asynchronous messages include *Packet-in*, *Port-status*, *Flow-Removed*, and *Error* messages. The symmetric messages include *Hello*, *Echo*, and *Experimenter* messages. Furthermore, there are Controller-to-Switch messages, where the controller initiates messages that may or may not require a response from the switches such as *Flow-Mod* and *Packet-out*. These messages provide means to the controller to implement all kinds of functions, such as forwarding, filtering, blocking, etc., by configuring forwarding devices' flow tables. Also, another feature of OpenFlow is aggregating network statistics to be used by the controller for maintaining a global view and/or for monitoring purposes.

2.1.3 SDN and Mobility Management

The SDN concept has been integrated into both wired and wireless networks. Even though the SDN technology is introduced to solve some issues in the current wireless architecture, the nature of the wireless networks nowadays imposes critical challenges due to the rapid and extensive growth of mobile traffic [11]. Meeting the users' requirements and providing a high quality of service become necessary and more challenging [70]. One of the main concerns of mobility management is maintaining a user on-going session continuity with relatively minimum interruption or, in other words, minimizing handover latency. Another concern, is integrating heterogeneity of wireless technologies.

2.1.3.1 Integration of Heterogeneous Networks

The difference in characteristics among heterogeneous wireless technologies imposes the challenge of designing mechanisms to integrate different access technologies, protocols, and service demands [13]. For instance, small cells, such as Wi-Fi hotspots, provide better data rates and cost-effective connections; yet, the security and privacy aspects can be violated. On the contrary, macrocells offered by cellular service providers, ensure higher security and privacy levels, although their services are costly. However, combining resources between Wireless Local Area Network (WLAN) and cellular systems is highly beneficial to mobile users. Therefore, trade-offs are essential, and hence, should be considered in designing intelligent inter-networking management mechanisms [61].

An interesting discussion was carried on by the authors of [120]. They addressed the challenges in current heterogeneous networks, as well as SDN promising features to solve some of these issues. The main concept that this paper suggests is "openness" in the wireless world. This concept is currently constrained due to the proprietary and closeness nature of different technology providers. For instance, one can be standing at any location where he/she has the coverage of multiple wireless networks (e.g., enterprise networks, coffee-shop networks, cellular networks), yet they are unavailable. The authors promoted the combined open service irrespective of infrastructure in order to gain access to more wireless capacity. This idea can be beneficial to different applications, specifically, load balancing.

In SDN-based heterogeneous networks, a distributed set of controllers are built on top of different wireless infrastructures, where each controller manages a domain/technology. With the increasing demand for real-time applications, streaming, and the Internet of Things (IoT) applications, combined with the increasing use of mobile handheld devices, there has been a significant increase in the control traffic. SDN controllers have to handle this control traffic to accomplish mobility procedures, and since these controllers have

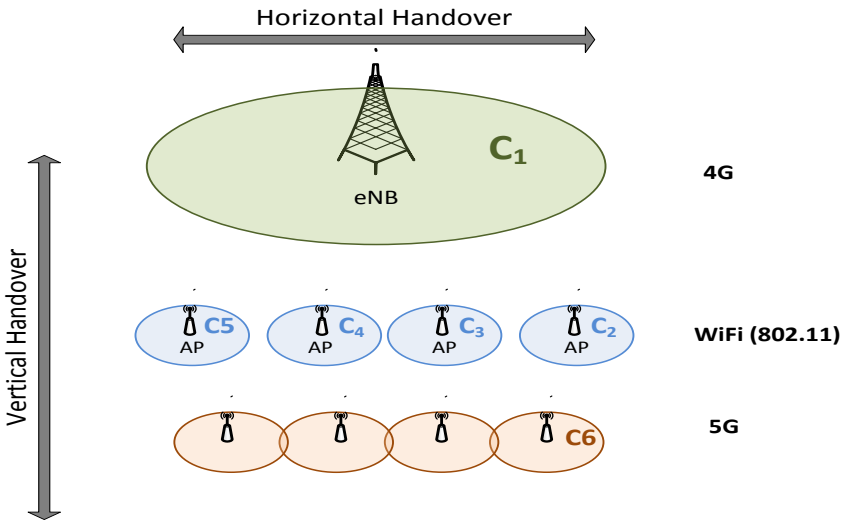


Figure 2.2: Vertical handover vs. horizontal handover illustration.

limited resources, handling a large amount of traffic/flows originating from switches may cause delays [16]. In practice, as a cellular network is expected to accommodate more users compared to other types of networks such as Wi-Fi, these networks' resources can be complementary in order to relieve cellular resources [31]. Vertical mobility is a solution to overcome such an issue, where it can be achieved by switching the mobile device interface to be connected to another network/technology [61], refer to Figure 2.2. Note that C_i, where $i = \{1, 2, 3, 4, 5, 6\}$ represents the managing controller of each network.

A considerable body of literature has focused on integrating WLAN and 3G technologies in traditional networking [19, 61]. Prior studies have proposed the integration using two architectures, known as, loose-coupling and tight-coupling inter-networking. The loose-coupling architecture makes WLAN a complementary access network to a 3G cellular network, where the traffic offloaded to a WLAN (e.g., Wi-Fi) bypasses the cellular core network and directly goes to the Internet. On the other hand, in a tight-coupling architecture, the WLAN is connected to the core network. The inter-network using the loose-coupling design makes the different technologies completely separate from each other (i.e., autonomous domains), which we envision as disconnected graphs. In contrast, the tight-coupling makes the different technologies connected, so they are not isolated from

each other. This design, however, sometimes mandates the same operator to own both the WLAN and 3G parts of the network [19].

All in all, we argue that the inter-networking between heterogeneous wireless technologies can be realized in practice if a confederation is established between different operators and the concept of SDN is present to orchestrate such confederation.

2.2 Related Work

The rapid spread of fast-moving mobile devices and the need to keep them attached at all times urges the need to find solutions that provide better connectivity and smoother transitioning between different cells in a cellular network, or between heterogeneous technologies. Studying mobility management indeed has a huge impact on users' perceived QoS [41]. Thus, this topic has been studied and analyzed thoroughly [21, 37, 46, 62, 117].

In this section, we shed light on several topics related to the handover procedure in an SDN-based network. We divide our literature review into five subsections. First, we start by reviewing some papers that linked SDN and handover. Next, we move to review some papers that provided analytical models that are related to our research. We then go through some papers that studied mobility management solutions for an LTE architecture. Then, we examine the merge between the SDN concept and one of the classical protocols used to handle handover in IP networks, MobileIP. Lastly, following our analysis, we study some previous scholars that linked the control plane load state with the network performance.

2.2.1 SDN and Handover

Karimzadeh et al. in [47] asked the big question "could OpenFlow-based SDN architecture be used to improve mobility management and support session continuity accordingly?" Based on SDN's principles, the authors of [109] argue that many of the existing issues are

alleviated. The programmability feature of SDN can dynamically set the shortest path between two communicating nodes, thus eliminating triangle routing. SDN also allows the mobility function to be transparent to users and mainly done on the network side, which makes handover faster and more efficient. Furthermore, the control centralization feature decreases complexity as well as overhead. The authors of [48], therefore, argue that SDN outperforms other approaches in handling mobility management.

Handover in an SDN-based environment has been studied in different settings from different perspectives. For instance, some papers compare the effect of different approaches of the SDN controller(s)' distribution on handover metrics. In [59], the authors present a comparative analysis of three variants of SDN-based mobile networks: centralized SDN, semi-centralized or distributed, and hierarchical SDN. The authors of [69, 74, 103, 121] argue that in terms of throughput and latency, distributing the control plane has a better outcome than having a central controller. Some papers also have proposed the use of the hierarchical approach given that the hierarchy of controllers may benefit the mobility management in terms of scalability and privacy [14, 46, 73].

From a different angle, some papers such as [16, 39, 40, 116] discuss the controller placement problem, which means finding efficient algorithms to place a given number of controllers in such a way that a pre-defined objective/metric is optimized.

The aforementioned studies have addressed different handover metrics, but to us, the most important was the delay. The authors of [70] studied the causes of overall latency in SDN-based mobile core networks. They linked latency to two major factors: propagation delay and processing delay. Regarding the processing delay, the authors suggest two strategies. The first strategy is to optimize the handling procedures of OpenFlow signaling messages, while the second strategy is to define forwarding mechanisms in order to reduce the number of switches involved in configuring the data plane. Based on their first suggestion, we were inspired to study the theoretical aspects of the processing delay, which led us to develop an analytical model based on the queueing theory. As for their

second suggestion, it led us to the binding-cache problem and motivated us to study such problems in different settings.

2.2.2 Modeling OpenFlow Messages

Most of the work related to SDN that we have come across was based solely on experiments and simulations. Recently, theoretical analysis has been gaining attention [93, 115, 123]. A mathematical model can provide researchers with insights into how an OpenFlow architecture performs according to specific parameters under given circumstances, thus leading to propose efficient algorithms to tackle existing issues.

The author in [101] explored the challenges and benefits of using the queueing theory to model SDN. He presented several challenges in modeling SDN systems, for instance: the latencies imposed by moving data between different caches, resource contention among threads, interruptions triggered by hardware or software processes, and the implementation design of queues. Another challenge involved is the modeling of the finite capacity of a system and limited buffer size. However, most of the related work, as the author mentioned, have omitted these two challenges and have considered them infinite.

The work in [45] by Jarschel et al. is one of the initial attempts to model the OpenFlow-based interaction between a switch and a controller. The authors provided a performance model based on the queueing theory of an OpenFlow system. They derived some performance parameters in order to introduce an analytical model. To do so, they set up an OpenFlow testbed. They considered a feedback-oriented queueing system model, which was divided into a forwarding queue system (i.e., a switch), and a feedback queue system (i.e., a controller). Their model for the switch was based on an $M/M/1$ queue and an $M/M/1-S$ for the feedback queue, meaning the queue size was infinite at the switch as opposed to a finite queue size at the controller. Their model has some limitations; for instance, it captured only the interaction between a single node and the controller.

As Duan et al. showed in [30], the average delay relies on the flow table state within switches. They defined the average packet delay as the result of two kinds of delay. One is a delay within the switch if it has to forward to the controller. The other is a delay after the handover decision has been made, as the two switches that are involved in the process have to contact the controller to accomplish the handover. In their work, the authors provided an analytical framework to quantify the total packet delay in an SDN-based network in a centralized controller setting. They studied the single node model in which they did not capture the handover control messaging exchanged between a switch and the controller.

In [68], the single node model, the authors modeled the feedback interaction between a switch and the controller. They used the Jackson network to model the data plane with modifications to fit the nature of the traffic flow in an OpenFlow-based network. The controller is modeled as an external entity based on an $M/M/1$ queue. They assumed the Poisson distribution for arrival rates at both the switch and the controller, which provided an exponential service rate in both. Their performance measures were the average packets sojourn time and the distribution of time spent by the packet.

In addition to using the queueing theory for evaluating SDN deployment, the authors of [15] provided an analytical model based on network calculus theory. They evaluated the SDN controller and switch in terms of delay and queue length. They provided the upper bound of packet processing delay of switches and the upper bound of the controller queue length. However, they did not consider the feedback between the switch and controller. Additionally, there was no simulation data recorded to validate their mathematical model.

Most of the literature reviewed has modeled the single case node, meaning they have modeled the interaction between one switch in the data plane and one controller. A down-side to single node modeling is that it is an unrealistic model for such architectures. They omit the possibility of incoming flows from multiple switches, meaning that their input to the switch is inaccurate. To the best of our knowledge, the work in [67] by Mahmood et al. is the first that considered the multiple node case, where more than one switch exists in

the data plane. They extended their work in [68] to model OpenFlow *Packet-in* messages in the multiple node case. The authors presented an analytical model to estimate the packets' sojourn time and the density of the packets that can be pumped into the network. They studied both infinite and finite buffer size scenarios. However, their modeling did not consider the propagation delay. Even though this delay might be small, it is essential to be considered while modeling multiple nodes, where the propagation times may differ as the multiple nodes get placed in different locations. Moreover, another study that took into account modeling several switches interacting with one controller is [93]. Shang et al. studied the impact of *Packet-in* messages on the performance of OpenFlow-switches and controllers. Additionally, they modeled switches as $M/H_2/1$, where the service rate follows a two-phase hyper-exponential distribution, whereas the controller is modeled as $M/M/1$. They also showed that the performance of the network drops rapidly as the probability of *Packet-in* messages increases. They provided an analytical model; however, no simulation data was provided.

In the aforementioned studies, the proposed models consider only the cases of exchanging one kind of message, *Packet-ins*. However, based on the direction of our research, we need to incorporate another type of message that is tightly coupled with handover procedures (i.e., *Port-status* messages). Additionally, we need to consider the multiple node case.

2.2.3 Mobility Management Solutions for LTE

As the SDN concept was introduced to overcome challenges in current architectures, many papers have studied integrating SDN into the LTE technology [21, 26, 44, 97, 102]. The authors in [44], for example, argue that virtualization through slicing provides multiple benefits, including stability, better management, and easier innovation adaptation. The slicing can be subject to users, subnets or traffic.

Through analyzing existing LTE architecture, one can find several proposals focusing on SDN for changing the current cellular network. Nguyen et al. in [79] give a general view of challenges faced by the LTE architecture: it is expensive to modify and upgrade the current system, it lacks the support of new network services and applications, and it generates ineffective management of existing network resources. They also provided a new way to fix the current LTE system by using SDN. The authors provide a potential possibility for changing the current cellular network to SDN and virtualization-based architecture.

The authors in [90] focus on ensuring on-demand service in LTE core systems, which means moving active sessions from one network element to another in a transparent manner without causing any interruption in the user end. Two aspects have been focused on in this paper: resilience and load balancing. They proposed an OpenFlow-based control plane architecture based on not moving all the functionalities completely to the control plane to realize their goal. However, their proposal has some challenges such as extending the OpenFlow protocol's functionalities to meet their requirements.

Chourasia et al. in [24] reveal a possible solution for using SDN to replace existing LTE architecture. In this architecture, they proposed the use of a centralized controller that has a global view of the whole network, which makes the resources management easier. Based on their simulation results, this solution has outperformed existing LTE architecture in terms of signaling cost and handover latency. On the other hand, the work in [59] shows that physically centralized SDN has disadvantages; it limits the distribution of handover decisions and has scalability issues.

In [35], the authors discuss the mobility management of future cellular networks. They argue that the mobility management function should be distributed in order to handle an increasing load of data and support the system scalability. They showcase three approaches for distributing mobility management: a PMIPv6-based approach, a routing-based approach, and an SDN-based approach. The first approach is based on Proxy Mobile IPv6 protocol. The idea here is to distribute mobility access gateways (MAG), which works

closely with the local mobility anchor (LMA). The second approach works in a distributed way by removing existing anchors in the system. When users move to new locations, nodes in the system need to calculate new routing tables information. The last approach is based on the SDN concept; the controller in this approach configures rules for switches, which are working as distributed mobility management gateways.

Valtulina et al. in [105] elaborate on an SDN architecture that provides distributed core functions as entities in the network and uses cloud computing technology to process network data. This architecture provides a traffic redirection mechanism during the handover and can keep user's data transfer seamless using the OpenFlow protocol. Moreover, Braun and Menth in [18] describe the operation of OpenFlow and different aspects of SDN-based architecture design. For the control plane, they discuss the controller distribution as well as methods of signaling management. Regarding the data plane, they feature the fault tolerance and resilience aspects.

2.2.4 MobileIP and SDN

Pupatwibul et al. [87] suggest a solution to better handle handover in mobile IP networks using OpenFlow. They rely on the structure of SDN to provide better handling of handover among multi-regions; they do not approach the problem in an optimized way, but instead, they adopt a distributed approach for the control plane. In [109], the authors discuss the shortcomings in the existing MIP protocols, such as inefficient handover, triangle routing, and high signaling overhead. They argue that SDN overcomes these issues. Then, using SDN, they proposed an algorithm to minimize the signaling messaging required to redirect an on-going session after a handover takes place. In other words, they tried to optimize the number of times a binding-cache is downloaded per mobile node movement. Inspired by their approach, we used the same idea but in a different architecture. [110] is a follow-up paper of [109] by the same authors. In this paper, Wang et al. extended their work to in-

clude several controllers and allow inter-controller communication. Their architecture was distributed where controllers need to exchange information to achieve efficient handover. They allowed triangle routing to minimize the number of flow table updates. Also, they suggested the use of broadcasting of inter-domain handover. In fact, broadcasting is an expensive procedure, especially in a high mobility environment. Well, we argue that this can be minimized or even mitigated in a hierarchical setting where a root controller is the only controller to be notified of mobility events between different domains. Therefore, it is worth studying to investigate a hierarchical model, where there is a root controller on top of a set of distributed controllers. Also, authors of [110] had inspired us to approach the multi-region networks in a slightly different way. Since the multi-region environment may represent different service providers or different wireless technologies (LTE, Wi-Fi, WiMax, etc.), we argue that the elimination of some inter-controller communication has a positive impact in terms of signaling overhead.

Another paper that studied mobility management between multiple domains was written by Hata et al. [38]. They proposed a protocol that manages mobility in multi-domain networks. Their architecture was distributed, which requires the cooperation of the involved domains to exchange information about a mobile node. In this approach, the authors suggest that each time an inter-domain handover has taken place, three controllers should exchange the so-called "node connecting information." Controllers must maintain a map to allow route optimization in a distributed way, which needs to be updated periodically. However, this paper does not show the quantitative results of mobility-management metrics; the authors only showed that mobility management can be realized in SDN.

2.2.5 Load Balancing and SDN

As we mentioned earlier, one of the main challenges introduced by SDN is scalability. There have been numerous studies which investigated how to improve the network scalability to

accommodate huge traffic flows without forming a bottleneck or single point of failure (SPoF) [65]. One countermeasure to such situations is load balancing. In an SDN-based environment, load balancing can be achieved at the control plane level or the data plane level, or combined [65].

Due to the wireless characteristics and scarcity, it is intuitive to implement load balancing mechanisms at the infrastructure level, specifically the cellular infrastructure [31]. Due to the fact that cellular resources are limited and challenged by the increasing number of mobile users and their QoS requirements, some cells are unable to sustain such an increase [61]. Therefore, several methods are reported in the literature to address this issue by exploiting network heterogeneity along with SDN. Some methods have focused on offloading data from a cellular network to Wi-Fi networks to relieve the cellular resources, and balance the load [30, 76]. Namal et al. in [76] targeted data plane load balancing by proposing a probabilistic approach leveraged by flow admission control. Their main metric was measuring the user's level of satisfaction in terms of waiting time and drop rates. Their approach, on one hand, offloads the core network to avoid over-utilization, and on the other hand, reserves resources for handed-over users. Interestingly, their approach takes the user preference into account resulting in a drastic drop in the unsatisfied-users percentage. Another approach proposed by Duan et al. in [30], also focused on relieving the cellular resources with the help of nested Wi-Fi networks while maintaining users QoS. Their approach showed a significant reduction in cellular resources utilization, while QoS levels were satisfied. To the best of our knowledge, the majority of work that has focused on wireless access networks' resource allocation in SDN-based environment, was irrespective to the control plane resources.

At the control plane level, it has been proven that a single controller has limited resources, and thus, handling large amounts of control flows originating from switches may cause delays [16]. The thing that directly impacts the routing or the procedure's completion has to be handled by the controller. Tootoonchian et al. in [104], show the tight

relationship between the response time of a controller and its load. They proved that due to the reactivity aspect of an SDN controller to flow setup, the response time directly affects the delay time of the flow completion. In the light of this reported result, it is conceivable that a controller that is in an over-loading state would require a longer period of time to accomplish any procedure, including mobility-related such as handover. A reasonable solution is to replace the single controller with a multi-controller platform where the network is partitioned [112]. However, this solution comes with a toll in terms of load balancing [42].

Incorporating load balancing techniques into SDN multi-controller design enforces the availability and scalability aspects of a network to provide a minimal response time [50]. Drawn to this angle of research, several studies have been conducted [12, 42, 50, 78, 112]. In our direction of research, we argue that the context of heterogeneous networks fits right in the multi-controller modeling given that heterogeneous technologies belong to different operators/service providers.

The control plane load-balancing is correlated to several factors (i.e., design choices); one important factor is the mapping between the controllers and the switches, the so-called switch-controller assignment. The switch-controller deployment can either be fixed (i.e., static) or dynamic based on switch migration. The static assignment implies that the network topology is fixed regarding the connections between the controllers and switches throughout a running system. Alternatively, the dynamic assignment implies that the switches can change their controllers through migration. In this matter, the researchers have been divided into three groups.

One group of researchers has adopted the static assignment of switches to the controllers and exploited other strategies to balance the load among controllers [43, 55, 106, 107]. One of the initial studies in SDN control plane load balancing was done by Hu et al. [43]. The authors of this paper proposed a hierarchical static architecture for wide area OpenFlow networks. Their heuristic partition algorithm achieves a tradeoff between controllers' load

and propagation latency. Their approach is based on a partitioning algorithm that reallocates flows to different controllers, which in return minimizes the flow setup delay because it assigns the nearest controller to the switch. However, this method introduces high complexity and costly deployment. Considering a flat implementation for the control plane, Koponen et al. proposed Onix [55], which is intended to scale out a network. This approach used the Network Information Base (NIB) to allow Onix instances to store, retrieve, and maintain states. Though, as the network size grows, this method may introduce overhead due to memory restrictions. Also, this method did not advise for an over-loading countermeasure. In another work, the authors of [106] proposed a rounding algorithm that showed improvement in the controllers' response times by using link balancing in a static assignment. Nonetheless, they did not provide a clear explanation of how the distributed controllers exchange statuses.

Another group of scholars has focused on the switch migration between controllers to achieve the load balancing [22, 27, 60, 92, 126]. The topic of switch migration was initially triggered by [29], and was supported by the multi-controller feature added to OpenFlow v.1.2 and subsequent versions [84]. The primary issue in dynamic assignment is the selection of a switch-controller pair for migration. A considerable body of this group of studies has adopted the greedy approach by selecting the heaviest switch to be migrated from an overly utilized controller to another lightly utilized controller [60, 92, 122, 126]. For instance, Selvi et al. in [92] adopt a hierarchical design, where a super controller has the managing part of collecting information from other controllers and detecting the imbalance state. They proposed a cost-greedy algorithm for reassignment and allocation of switches and controllers based on previous numbers of handled flows as well as the projected values of flow requests. Yet, the proposed approach introduces higher overhead due to periodic and reactive exchange of load information between the super controller and the other controllers. Targeting the problem of having multiple over-loaded controllers, Cui et al. [27] proposed a load balancing mechanism that works with distributed controllers and handles

multiple overloaded controllers simultaneously. Their imbalance detection is fine-grained and based on response time, and their solution is based on greedy switch migration. They targeted minimizing migration cost by limiting the number of migrated switches. However, this scheme may introduce congestion and overhead due to simultaneous switch migrations. A recent study by Zhang et al. [125] formulated the multi-controller load balancing as an optimization problem to minimize the average response time of the control plane. They then proposed an online lightweight solution that iteratively migrates switches to decrease the mean response time until the load is balanced. However, the dynamic assignment in general, comes with some issues, such as complexity, high cost, and high latency due to migration and re-association [36, 107].

The third group of scholars has presented a so-called partial switch migration, which means the flow requests generated by a switch are distributed among multiple controllers [36, 99, 107]. The authors in [36] proposed a dynamic distributed control plane architecture, where they can dynamically manage the number of controllers, switches, control flow, and traffic flow through in-bound and out-bound channels. Their management aims to re-distribute load among controllers taking into consideration the topology and applications demands through flow-shifting and flow-splitting. Another approach proposed by Wang et al. in [107], was mainly dependent on the idea of balancing the load of multiple controllers based on shifting aggregate flows from the management of one controller to another. The proposed approach is implemented in a hierarchical static architecture, where a set of distributed controllers report their load information and flow entries statistics to a root controller. Then, the root controller determines on aggregating and redirecting the flows to ensure fairness among all the controllers. This approach is pro-active and based on traffic prediction. The aggregation of flows relies on installing wildcard rules on some switches, which can be, hardware-wise, costly. In a recent work by Al-Tam and Correia [99], they proposed a heuristic approach to migrate fractional flow requests while minimizing the number of new mappings. Their approach showed more resilience and better load bal-

ancing in large-scale networks. All in all, the partial switch migration approach implies that a single switch is connected to several controllers, which still suffers from some of the dynamic migration issues.

Besides the switch-controller deployment issue, there are other issues that contribute to degrading the performance of any multi-controller load-balancing mechanism, specifically, the controllers status synchronization. A set of controllers need to inform each other of their load status to avoid forming choke points and, in some cases, make fair load distribution [122]. Consequently, the control overhead increases, which, in turn, can result in a costly process, temporally and spatially [29]. In light of that, numerous studies have targeted minimizing the overhead of exchanging the status among controllers by decreasing the amount of exchanged load messages. For instance, the authors of [126] proposed Dynamic and Adaptive Load Balancing algorithm (DALB). Their mechanism requires each controller to collect its local load information periodically; when its load reaches a threshold, it collects other controllers' load information (i.e., load information aggregation). Their solution adjusts the triggering threshold to the average load of all controllers combined to minimize frequent collection of statuses. Still, this mechanism suffers high inter-communication overhead. In [122], Yu et al. proposed a mechanism based on reporting the load status periodically and then storing these controllers' information. When a controller becomes overloaded, there is no reactive collecting of information before making a decision locally, unlike [126]. Then, to reduce the frequency of load informing, they proposed an inhibition algorithm. Also, in [60], Lan et al. make the frequency of status exchange dependent on a pre-defined threshold, the closer the load to the threshold, the more frequent the status exchange. They proposed the use of a distributed database to synchronize load information among the controllers. Reportedly, retrieving data from a database may cause an unmanageable delay.

On a different note, the authors of [99] showed how the frequency of load informing could be determined based on either speed or accuracy, which means that there is a trade-

off between whether the main network policy is achieving high load balancing or quick avoidance of over-loading and under-loading. Their imbalance detection relies on two thresholds, under-loading and over-loading.

Chapter 3

Modeling Handover-Related OpenFlow Messages

A mathematical model can provide researchers with insights into how an OpenFlow architecture performs according to certain given parameters under given circumstances, thus leading to propose efficient algorithms to tackle existing issues. Therefore, we provide an analytical model based on results from the queueing theory of an OpenFlow-based mobile network. We restrict our analysis of the handover delay to target the delay caused by exchanging handover-related OpenFlow messages. We are, therefore, concerned with minimizing the delay caused by the exchange of signaling messages between the control and data planes. To the best of our knowledge, no other work has modeled OpenFlow messages in relation to the handover procedure completion. In fact, our analysis provides a novel mathematical modeling for such messages.

3.1 Network Model

To serve the purpose of our network, i.e., a mobile network, the two main messages that would frequently be sent by a switch are *Packet-in* and *Port-status*. However, the exchange

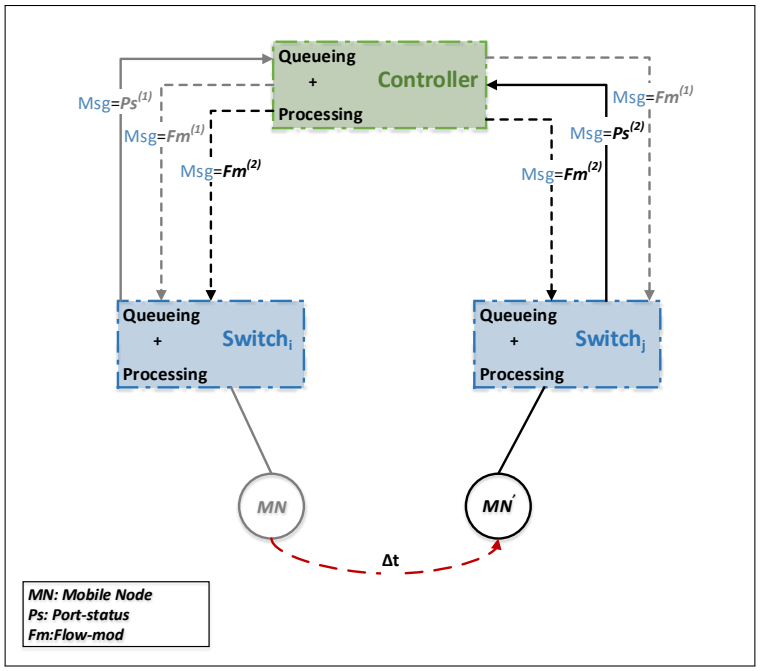


Figure 3.1: The exchange of messages between the switches and controller upon mobility.

of the rest of the messages is relatively small as they are either initiated upon the start up of the network or for testing and monitoring purposes. Therefore, we considered analyzing the behavior of *Packet-in* and *Port-status* messages combined in our system.

Regarding a *Packet-in* message, when a packet arrives at a switch, we have two cases. First, the packet has a flow entry that matches the packet's header, so it is forwarded based on matching fields. Second, the packet has no entry; then a *Packet-in* message is sent to the controller to obtain knowledge. The controller decides on the best forwarding path and updates relevant switches with new flow entries to be installed in their flow tables. Regarding a *Port-status* message, when a mobile node (MN) is disconnected from a switch, a *Port-status* message is initiated by that switch and sent to the controller. Similarly, when an MN is connected to a new switch, another *Port-status* message is initiated by the new switch and sent to the controller. Upon receiving a *Port-status* message, the controller updates some or all of the nodes with up-to-date entries about that MN.

We consider a network where hosts are connected to n OpenFlow-switches. All of the OpenFlow-switches are controlled by a centralized controller, C, which resides in a

relatively remote location and has a global view. Our network is a typical OpenFlow-based network, where *Packet-in* messages are considered a cornerstone of its implementation.

In our network, any host can belong to one of two groups at time t. A group of hosts is connected to a switch, s_i, without attempting to handover while the other group is switching from one switch, s_i, to another, s_j. We refer to the hosts of the second group as mobile nodes (MNs). An MN handover requires two connection requests to the controller, one from its old switch and the other from its new switch, *off-port*, and *on-port*, respectively. The movement of the MN from one location to another requires a clear configuration of all or multiple switches (i.e., sending out *Flow-mod* messages by C). Figure 3.1 gives a visual illustration of the exchange of messages between the switches and controller triggered by the mobility of a node.

In this section, we aim to quantify the handover delay incurred due to the propagation of control messages between switches and controller as well as for processing and queueing those messages at the controller.

3.1.1 Assumptions

We assume that every *off-port* message corresponds to an *on-port* message. Intuitively, the number of *off-port* messages equals the number of *on-port* messages; therefore, we can say that both kinds of control messages can be represented as *Port-status* message. We also assume that the controller has infinite buffer size, which means no packets are dropped. It is debatable whether or not this assumption can be adopted, because, usually, controllers are equipped with huge resources that are virtually infinite but actually limited. However, for simplicity, we assume infinite queue size at the controller. We additionally assume infinite switches' queues in line with previous work [30, 45, 115, 119], and as studying the impact of limited buffers and dropping packets rates are out of the scope of our analysis.

Regarding the handover calls distribution, the authors of [53] and [23] have shown that they can be modeled as Poisson. Convinced by their validated results, we assumed that the occurrences of *Port-status* events follow the Poisson distribution. To put it another way, we argue that since the arrival rate of *Port-status* events are human-initiated, the inter-arrival and service times are independent and exponentially distributed and features the memoryless property. Additionally, in our modeling, we assumed that the service rates of all nodes were load-independent.

3.1.2 Model Description

In this section, we start with modeling the *Port-status* messages; then we expand our model to include another type of messages that are essential in SDN-based networks, *Packet-in* message.

3.1.2.1 Modeling *Port-status* Message

Regarding handover, we consider analyzing the hard handover, i.e., break before make approach. Our analysis focuses on the process that follows a handover decision. When an MN is disconnected from one switch, say s_i, it gets attached to another switch, say s_j, where $i, j \in \{1, 2, ..., n\}$. This process triggers two asynchronous *Port-status* messages to update the controller of MN's movement. On the controller side, its network map has to be in an up-to-date state. Accordingly, the controller initiates *Flow-mod* messages to all or some of the switches changing the flow entries of that MN.

A *Port-status* message generated by a switch can be either *off-port* or *on-port* message. Both types of *Port-status* messages are treated the same, but, of course, the controller can distinguish between them; therefore, we refer to both kinds of messages as one, *Port-status*. In part, every *Port-status* message triggers three other messages: *Flow-mod* message by

C, another *Port-status* message by the new switch, and a second *Flow-mod* message by C upon receiving the new switch's message.

Based on the principles of modeling using the queueing theory, there are a set of parameters that need to be identified, including the arrival rate of requests, service time, the capacity of the system, the size of the source, and the service disciplines [98]. Taking into consideration our assumptions and using Kendall's notations [51] for queueing systems, the switches and the controller are modeled as $M/M/1$ systems, where we assume that the queues are infinite and the service discipline is FIFO. We considered having an open queueing network where events can enter the system from outside and can also depart the network from any node. We additionally assumed a Markovian system, where a system's state is defined by the number of events (i.e., jobs) at nodes. If k_i is considered as the number of jobs at node i, then the state of the network is defined as $(k_1, k_2, ..., k_n)$. Note that throughout this chapter, we use the words jobs and events interchangeably to refer to the buffered elements that need to be processed in a queue.

Now, let us assume that the arrival rate of *Port-status* messages initiated by a node i is λ_{ih}. Then, the flow of *Port-status* events captured by any node i is modeled as:

$$\gamma_{ih} = \lambda_{ih} \tag{3.1}$$

Note that a port can change from *on* to *off*, or a new port gets attached. Either way, the two events are treated alike.

Accordingly, Γ_{ih} is the total net arrival rate of port changes from the whole network to node i (see Figure 3.2), and is given as the following:

$$\Gamma_{ih} = \gamma_{ih} + \sum_{j=1, j\neq i}^{n} (\lambda_{jh} \times v_j^i) \tag{3.2}$$

Here, $v_j^i \in \{0, 1\}$ is an indicator applied to model the effect of the *Port-status* message

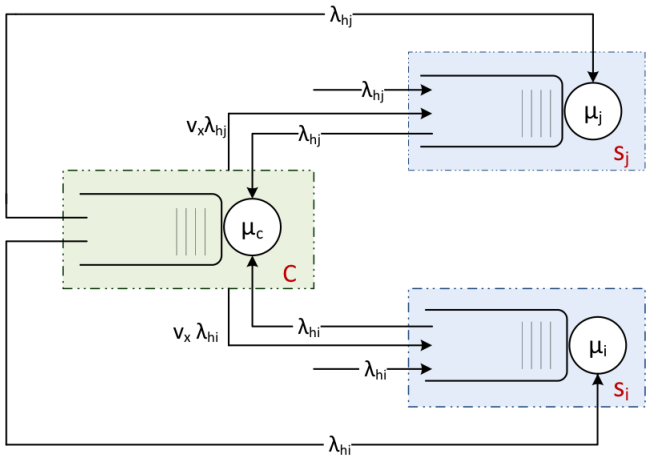


Figure 3.2: A queueing model of *Port-status* messages.

after being handled by the controller. If i's flow table has to be configured based on j's *Port-status* message, then $v_j^i = 1$; otherwise, $v_j^i = 0$. Note that some or all nodes may be configured by the controller upon receiving a *Port-status* message. Regarding the controller, Γ_{ch} is the total net arrival rate of *Port-status* messages at C and is given as the following:

$$\Gamma_{ch} = \sum_{i=1}^{n} \lambda_{ih} \tag{3.3}$$

So far, our modeling includes only *Port-status* messages, meaning that we consider a single class queueing network. However, our model is extended in the following subsection and includes another type of messages, *Packet-in*.

3.1.2.2 Multiclass Open Network Modeling

When we model a network of queues where all the jobs are of the same type with regard to their service times and routing probabilities, then this is called a *single class network*. If the network model, however, is extended to include multiple job classes, then it is called a *multiclass network*. In this extended model, different types of traffic can be combined in an open or closed setting [17]. In our case, so far, we have considered modeling the *Port-status* messages in a network, which is unrealistic. In any SDN-based network, *Packet-in* messages

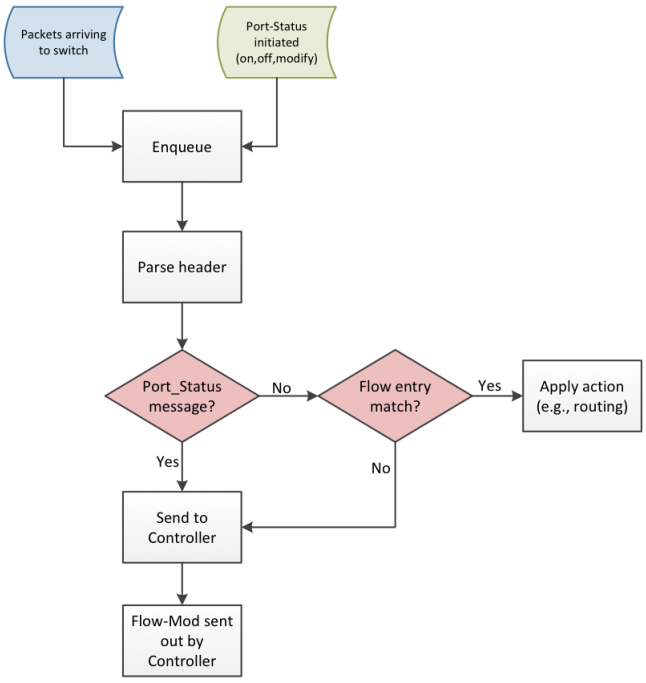


Figure 3.3: A flow chart of an OpenFlow-switch handling two messages, *Packet-in* and *Port-status*.

are essential. As has been shown by [84], other types of messages are exchanged between the data plane and the control plane for different purposes. However, in a mobile network, the most common two messages are *Packet-in* and *Port-status*. These two messages are treated differently, as illustrated in the flowchart in Figure 3.3. Therefore, our network is modeled as a *multiclass network*, where we restrict our analysis to the two types of messages.

To our queueing network that contains two job classes, we need to define the following symbols:

- R: is the number of job classes, where in our network, $R = 2$.
- k_{ir}: is the number of jobs of class r at node i, let $r = h$ if the the intended class is *Port-status*, and $r = p$ if the class is *Packet-in*. Similarly, k_{cr} is the the number of jobs of class r at the controller.

- S_i: is the state of node i, where $S_i = (k_{ih}, k_{ip})$. Similarly, S_c represents the state of C.

- S: is the overall state of the network with multiple jobs where the state probability is represented by $\pi(S_1, S_2, ..., S_n)$. Note that the sum of the probabilities of all states should be one since we assume stability in the system.

- μ_{ir}: is the service rate of node i for the job class r. In our network, we have μ_{ip}, which represents the service rate at a switch for *Packet-in* messages, while μ_{ih} is the service rate for *Port-status* messages. In our model, we assume $\mu_{ip} = \mu_{ih}$. Similarly, μ_{cr} is used to represent the service rates for the two messages at C.

- λ_{ir}: is the arrival rate of job class r at node i, so λ_{ir} is either λ_{ip} or λ_{ih}.

- Γ_{ir} and Γ_{cr} represent the traffic of class r at node i and C, respectively, and they are both functions of λ_{ir} and λ_{ip}.

So far, we have defined the parameters of modeling the *Port-status* message in the previous subsection. Now, we define *Packet-in* message parameters in addition to the aforementioned parameters. Let the arrival rate of packets, in general, to a node i be λ_{ip}. Then the probability of a *Packet-in* message to be sent to the controller from node i is q_i^{nf}, which means there is no flow entry regarding that packet. On the other hand, in case a switch matches an entry for incoming packets, our modeling should include the traffic between nodes. Therefore, the probability between any two nodes i, j is p_{ij}. At any node i, the flow of input packets can be modeled as:

$$\gamma_{ip} = \lambda_{ip} + \sum_{j=1, j\neq i}^{n} p_{ij}\gamma_{jp} \tag{3.4}$$

Then, the total net arrival rate of packets, excluding *Port-status* messages, is given as:

$$\Gamma_{ip} = \gamma_{ip} + q_i^{nf}\lambda_{ip} + \sum_{j=1,j\neq i}^{n} (q_j^{nf} \times u_j^i)\lambda_{jp} \tag{3.5}$$

where $u_j^i \in \{0, 1\}$ is an indicator of whether a flow has to be routed between i and j or not, as used in [67]. Using this indicator, the updates the controller sends to all or a subset of the nodes upon receiving *Packet-in* is included. Accordingly, the arrival rate of *Packet-in* messages to the controller is given as:

$$\Gamma_{cp} = \sum_{i=1}^{n} q_i^{nf}\lambda_{ip} \tag{3.6}$$

3.1.3 Model Constraints

Here, we list the constraints of our model. Firstly, we modeled the controller and switches as a single queue and not per interface. Also, we did not consider the case of dropped packets at either the controller or switches due to overload, which goes in line with our assumptions that we have infinite queue sizes.

OpenFlow protocol has different message types to be exchanged between a switch and the controller, and in our model, we only focused on two types of messages that are the most frequent messages in the networks we studied, i.e., *Port-status* and *Packet-in* messages. Moreover, we assumed TCP traffic only, meaning only the header of the first packet of each new flow is sent to the controller, in contrast to UDP traffic, where the incoming packets of a new flow are relayed to the controller until the associated flow entry is installed.

3.2 Performance Measures

In this section, based on our model, the goal is to quantify the handover delay that occurs as a result of exchanging OpenFlow-related messages. It is imperative to distinguish between two main delays: sojourn time and total delay. The sojourn time, T_{ir}, is the time an event spends at node i, including the time it is being serviced. Regarding our main metric, the total time, D_{tot}, is the time an MN experiences before completing the handover procedure as it will be discussed in further detail in this section.

Let W_i^c be the time of interaction between node i and C upon a *Port-status* event. Then, W_i^c can be defined as:

$$W_i^c = T_{ih} + T_{i(prop)}^c + max\{T_{x(prop)}^c\} + T_{ch}$$

Note that $T_{i(prop)}^c$ is the time for propagating a *Port-status* message from s_i to C, given that the T_{prop} between the controller and all switches are assumed to have the same link parameters. In OpenFlow design, the controller reacts to a *Port-status* message by sending out a *Flow-mod* message to all or a subset of switches in parallel. In other words, C sends reactive *Flow-mod* message(s) to switches in parallel, which may take roughly the propagation time required to reach the furthest switch; hence, we define the maximum propagation time as:

$$max\{T_{x(prop)}^c\} \text{ where } x \in \{1, 2, ..., n\} \text{ and } x \neq i \quad .$$

Then, the expected value of W_i^c can be further written as:

$$E[W_i^c] = E[T_{ih}] + T_{i(prop)}^c + max\{T_{x(prop)}^c\} + E[T_{ch}] \tag{3.7}$$

Based on the above explanation of how the controller handles *Port-status* messages, we infer that to complete the handover procedure of an MN, the total delay experienced by that MN is:

$$D_{tot} = W_i^c + W_j^c \tag{3.8}$$

where W_i^c represents the time of an *off-port* message handled by an old switch s_i, and W_j^c represents the time of an *on-port* message handled by a new switch s_j. This yields to:

$$D_{tot} = E[T_{ih}] + T_{prop} + E[T_{jh}] + E[T_{ch}] + E[T_{ch}^{(2)}] \tag{3.9}$$

where $T_{prop} = T_{i(prop)}^c + max\{T_{x(prop)}^c\} + T_{j(prop)}^c + max\{T_{y(prop)}^c\}$ and $x \neq i, y \neq j$

Note that $T_{ch}^{(2)}$ represents the sojourn time of an *on-port* message at the controller. We distinguish the two times because they are independent.

To find the mean response time of a *Port-status* event, we use the Mean Value Analysis (MVA) solution. MVA has been introduced as an iterative technique to obtain an exact solution for some performance measures including sojourn times in separable queueing networks. However, the MVA's original version was proposed for closed networks only. In 1981, Zahorjan and Wong in [123] presented the MVA solutions for open and mixed networks. Note that we are in line with their assumptions, which are the existence of a FIFO queueing discipline and a single server. In this section, we use their findings to compute our performance measures of interest. The MVA for open queueing networks depends on two theorems, Little's theorem [63] and the arrival instant theorem [75]. Therefore, we consider the Poisson Arrivals See Time Averages (PASTA) property of Poisson arrivals, which indicates that in a network in statistical equilibrium, the time averages equal the arrival averages [113]; hence, the sojourn time can be formulated as:

$$E[T_{ih}] = \frac{1}{\mu_i}(1 + (E[K_{ip}] + E[K_{ih}])) \tag{3.10}$$

where K_{ip} and K_{ih} represent the response times regarding *Packet-in* and *Port-status* mes-

sages, respectively. As proven by [123], for any two classes, in our case h and p, the following relation is satisfied:

$$E[K_{ih}] = \frac{\rho_{ih}}{\rho_{ip}} E[K_{ip}] \tag{3.11}$$

where ρ_{ih} and ρ_{ip} are the utilization (i.e., load) of node i with respect to *Port-status* and *Packet-in* messages, respectively. As we mentioned in our model description, the utilization is load-independent.

With the help of Little's theory, $E[K] = \lambda.E[T]$, and by substituting it in the previous equations, we get:

$$E[K_{ih}] = \frac{\rho_{ih}}{1 - (\rho_{ip} + \rho_{ih})} \tag{3.12}$$

Similarly, the previous equation is applied to find the controller's response time regarding *Port-status* messages.

So far, based on our assumption, and the findings and theories of previous works, we formulated the expected response time of a handover procedure in an OpenFlow-based network. However, there is a constraint need to be satisfied for our model to work. The utilization of queues has to be less than unity. Let ρ_i and ρ_c represent the total utilization of switch i and controller C, respectively; then $\rho_i < 1$ and $\rho_c < 1$, given that:

$$\rho_i = \rho_{ih} + \rho_{ip} \tag{3.13}$$

Similarly, ρ_c is defined.

3.3 Proposed Approach

In our approach, we aim at targeting two delay contributors. Firstly, we need to minimize processing latency, and to do so, we propose offloading handover handling to dedicated entities that are separate from our controller and switches. We call these entities Mobility

Handling Entity (MHE). Secondly, we aim at minimizing propagation delay by placing the MHE physically closer to switches. Essentially, by offloading the burden of handling *Port-status* messages to entities other than the controller, we efficiently minimize latency. In short, the role of each element in our solution is as follows:

- Controller: The controller maintains the information of all devices in the network, MHEs, OpenFlow-Switches, etc. Mainly, it has the global view.
- MHE: MHEs are placed physically close to switches and handle handover procedures asynchronously with the controller.
- OpenFlow-Switch: Switches focus on keeping data flowing from the source to the target. They are managed by the controller through OpenFlow protocol. They exchange commands with the controller to perform several tasks, such as updating the flow table, triggering *Port-status* messages, and other stats-related messages.

In our solution, the functionality of handling handover is installed on MHEs to allow them to work asynchronously with the controller. In this case, the controller has to maintain specific information about MHEs in the network. Each MHE is associated with a table of four columns, MHE_ID, MHE_IP, Status, and Update_Time, as shown in Table 3.1. MHE_ID and MHE_IP are used to locate a specific MHE. The Status is used to indicate whether that MHE is "active" or "inactive". Lastly, the Update_Time marks the time an MHE gets updated by the controller. As for the switches, we assume that they exchange handover-related messages with MHEs through OpenFlow channels. Therefore, in our approach, MHEs are considered SDN-controllers with limited functionality and a specific task.

Table 3.1: MHE information table.

Content	MHE_ID	MHE_IP	Status	Update_T
Data Type	Unit64	IPv4 address	Unit16	Time

3.4 Numerical Results and Analysis

In this section, we divide our analysis into two directions. Firstly, we verify and validate our analytical model by comparing its results to a conducted simulation experiment output. Secondly, we evaluate and analyze our proposed solution in more detail.

3.4.1 Verifying Analytical Model

An experiment was conducted to compare the simulation results to our analytical model results. We have developed a discrete event simulator using Matlab, and our simulation parameters are listed in Table 3.2. For the value of q^{nf}, Jarschel et al. in [45] showed that in a production network, the probability of *Packet-in* messages is 4%. We also borrowed some of the measurements in [45], such as the service rates of controller and switches. The size of OpenFlow-messages are mentioned in [84]. We needed the sizes to determine the overall delays based on our assumptions of the link type and speed, as listed below.

For the arrival rates of *Port-status* and *Packet-in*, we used the aforementioned values and substituted them in the following equations.

$$\rho_c = \frac{\Gamma_{cp} + \Gamma_{ch}}{\mu_c} \tag{3.14}$$

Table 3.2: Simulation parameters.

Parameter	**Value**
Probability to send *Packet-in* to controller q^{nf}	0.04
Average service time of the controller	240 µs
Average service time of a switch	9.8 µs
Size of *Packet-in* message	$128B$
Size of *Port-status* message	$128B$
Size of *Flow-mod* message	$128B$
Distance between switches and controller	$1-5$ kms
links between switches and controller	Optical
link speed	$1Gbps$

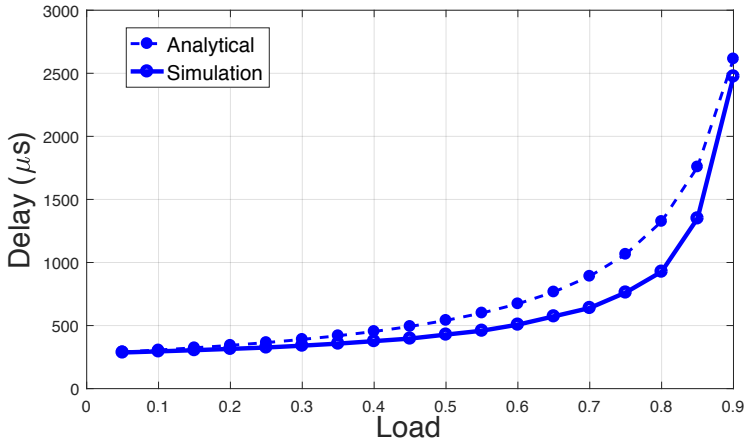


Figure 3.4: The simulation and analytical model results.

$$\rho_i = \frac{\Gamma_{ip} + \Gamma_{ih}}{\mu_i} \tag{3.15}$$

By equating Eqs. 3.14 and 3.15 at load $\simeq 1$, we got values of λ_{ip} and λ_{ih}. Note that λ_{ih} has to be much much smaller than λ_{ip} in practice.

Our comparison of the simulation results and the analytical computations shows similar trends that start and end almost the same, as depicted in Figure 3.4. Additionally, they both show a rapid increase in delay around $load = 0.7$, which is expected. However, the divergence that occurs around $load = 0.3$ can be contributed to the simulation running time and/or hardware specifications.

After gaining confidence in our simulation setup, we carry on our analysis in the following subsection.

3.4.2 Evaluating Proposed Approach

To show the difference that an MHE causes, we simulated the response time *Port-status* messages experience in both approaches, a network of queues excluding MHE, and a network of queues including MHE. Note that our network of queues consists of two switches, one controller, and an offloading entity in the second topology. Initially, we compared the two topologies when both MHE and C have the same processing time, which is con-

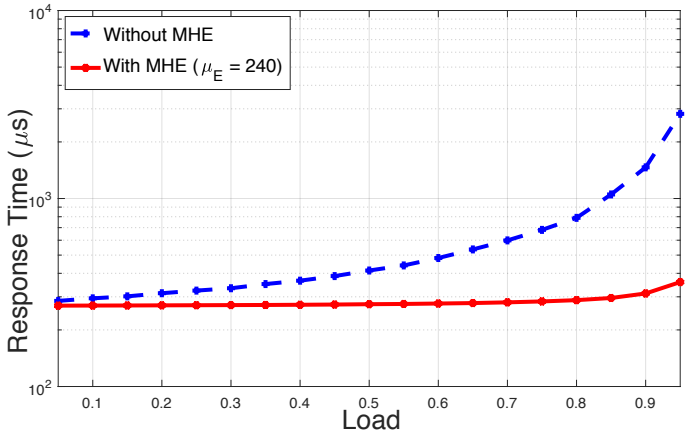


Figure 3.5: Comparing *Port-status* response time of a model with MHE to a model without MHE.

sidered the worst case scenario regarding MHE's service rate. As depicted in Figure 3.5, the response time drops tremendously in the case where MHE handles the *Port-status* messages.

MHE are entities assigned by the network designer to handle *Port-status* messages separately from the controller for the sole purpose of decreasing the time that a handover procedure may take. Those entities can be designed to be separate queues in the controller or can be separate physical devices that can be placed anywhere in between the controller and the switches. Additionally, they can be given different service rates to indicate different capabilities. Therefore, we break down our analysis in the following part into the impact of different service rates and the impact of physical placement.

3.4.2.1 Impact of Different Service Rates

MHEs' design is determined by the network operator. In this part, we try to give different options and analyze their impacts.

As a matter of fact, the service rate plays an important role in queues' processing times. As we are adding MHEs, we have the freedom to choose the service rate. However, we will compare three service time values, 9.8 μs, 240 μs, and 125 μs. We deliberately chose those

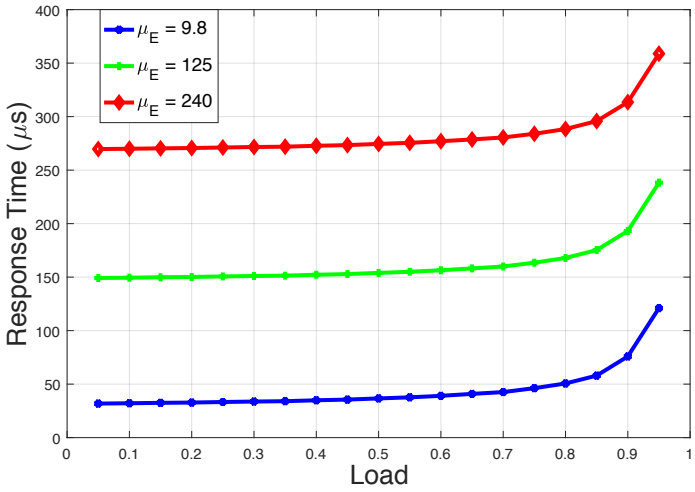


Figure 3.6: Response time of MHE with different service rates.

three values. We wanted to see the impact of having an MHE with a service time as fast as that of the switch (i.e., 9.8 μs). Then we studied the impact of having a service time as slow as that of the controller (i.e., 240 μs). Lastly, we suggested a service time value that is in between the two aforementioned values (i.e., 125 μs). Intuitively, the impact is quite obvious, as depicted in Figure 3.6. Based on the purpose and resources that are available, the network designer has to determine a suitable service rate.

3.4.2.2 Impact of Physical Placement

In networking systems, delivery time can be broken down into transmission time and propagation delay. The transmission delay represents the time from the beginning until the end of a message transmission, so it is correlated with the packet size and the bit rate of the medium. The propagation delay is the time the first bit takes to travel from a source to a destination, and, therefore, it depends on the physical medium as well as the distance separating the correspondents. In our analysis, we assumed 1 Gbps-optical links over 1 to 5 kms distance. We found that in this setting, the difference between 1 km and 5 kms is a matter of microseconds. Therefore, we argue that the location of MHEs in our setup does not have much impact on the total delay, so the entities can be placed in convenient

locations whether within the controller unit or physically separate in a particular place. However, using different link parameters may have an apparent impact on transmission and propagation delays.

3.5 Summary

It is important to model the OpenFlow controller to switch interaction in terms of the two kinds of messages that are commonly used in a mobile networks. Indeed, this modeling helps us better understand the underlying causes of handover delay and then helps us to propose effective methods to minimize it. In this model, we have modeled two OpenFlow messages: *Packet-in* and *Port-status* in a multiclass open network. We have modeled each message independently since they are two different traffic and need to be treated differently by the controller. Our aim has been to quantify the handover delay incurred due to queueing, processing, and propagating handover signaling messaging between switches and the controller. Then, we have proposed offloading the mission of handling *Port-status* messages to separate entities in order to overcome some of the shortcomings of an SDN-based solution.

Chapter 4

Functionality-Distributed Approach to Improve Handover Metrics in LTE

In this chapter, we focus on our problem in a cellular network setting, specifically LTE. We show that with the support of SDN and virtualization, the handover handling procedure can be improved. In cellular networks such as LTE, the SDN concept has been incorporated into different parts, especially into the Evolved Packet Core (EPC). EPC is made of multiple components such as Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (S-GW), and Packet Data Network Gateway (P-GW). MME is a key module responsible for mobility management tasks. Module HSS is considered a central database that has all the subscribers' information. S-GW and P-GW are gateways; S-GW forwards data between eNodeBs, while P-GW connects the end-user to external networks.

In SDN-based LTE architecture, the deployment of the control plane can be centralized or a distributed. As for the centralized approach, all EPC's components are virtualized and placed into a single controller. Alternatively, the distributed EPC controller architecture can be represented in different ways: either through implementing a distributed set of EPC controllers, or by distributing the functionalities of an EPC controller. Thus, the mobility management can be centralized or distributed based on the control plane deployment.

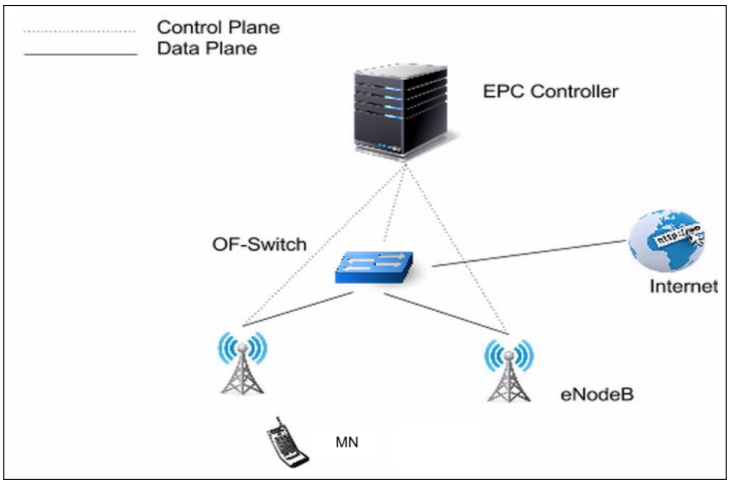


Figure 4.1: Centralized EPC controller architecture [24].

In our research, we consider a cellular network where mobile nodes are connected to eNodeBs, assuming that every eNodeB is connected to an OpenFlow-switch. All OpenFlow-switches are controlled by a centralized controller that resides in a relatively remote location and, so, has a global view; thus, it governs the handover procedure. We approach the problem specifically by minimizing the delay incurred as a result of the signaling messages exchanged between the controller and OpenFlow-switches. Additionally, we consider a hard handover, i.e., that which is in practice in LTE systems. Then we applied the approach we proposed previously in Chapter 3, and study its implication on some handover-related metrics.

4.1 Existing Work

As we mentioned before, some papers in the literature have approached the handover delay problem in a similar setting; for example, [24], where Chourasia et al. considered a centralized EPC controller. Meaning that the authors substituted EPC's MME, S-GW, and P-GW and combined them all in a virtualized central entity, see Figure 4.1. Their architecture keeps eNodeBs for handling radio communication and data transmission. The handover protocol flow proposed in [24] is illustrated in Figure 4.2. This approach has its

pros and cons. Some of the advantages are a network-wide global view and the rendering insignificant of the exchange of signaling messaging. On the other hand, the centralized approach makes a mobile network less scalable and less reliable, where the controller can form a single point of failure or a bottleneck [105]. Moreover, all the network traffic has to be processed by a single controller, which is something that may worsen the handover-related metrics.

4.2 Proposed EPC Controller Architecture

In this section, we apply the proposed solution in Chapter 3 to a cellular network where mobile nodes are connected to eNodeBs, assuming that every eNodeB is connected to an OpenFlow-switch.

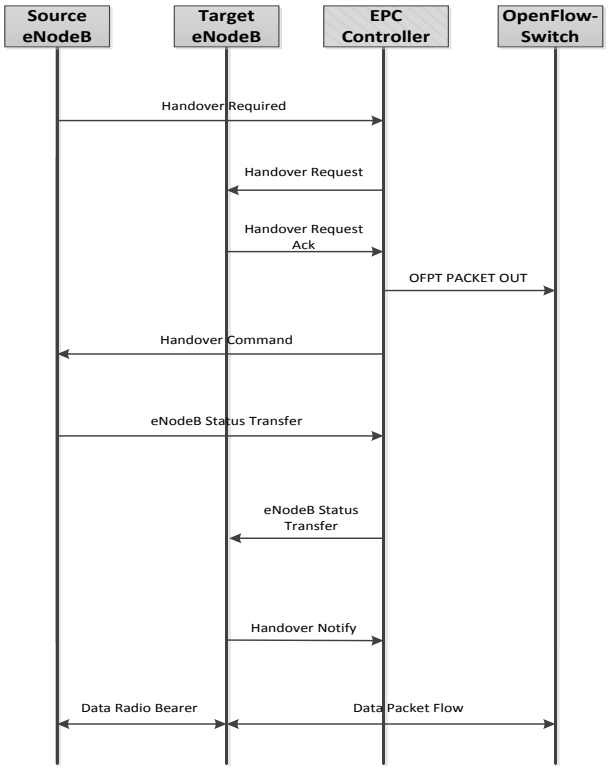


Figure 4.2: Handover procedure flow in [24].

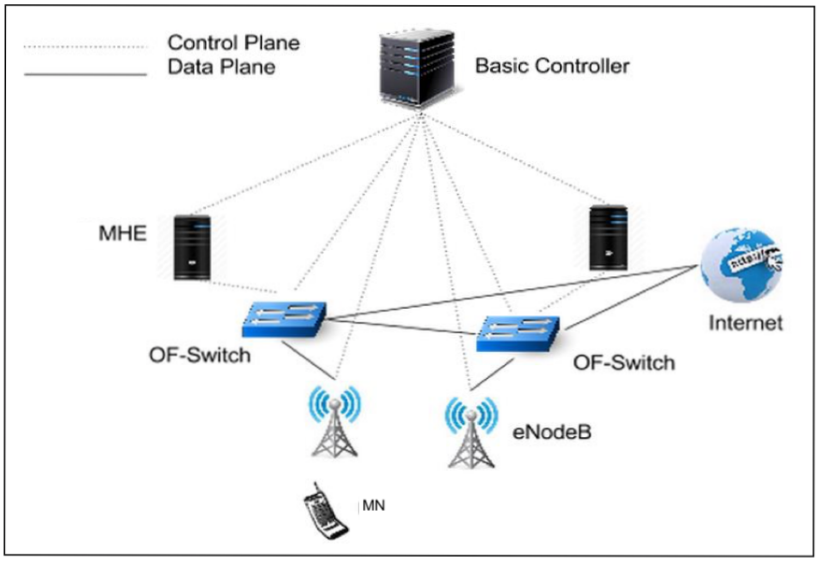


Figure 4.3: Proposed EPC controller architecture.

As previously stated, the handover signaling latency can be reduced by minimizing the processing delay of managing entities [70]. Therefore, we present our proposed solution in the LTE setting, where we assign entities that have to be responsible for handover messages. Thus, we distribute the functionality of the EPC controller. Our proposed EPC controller architecture enables multiple functional entities to work in conjunction with the controller, which operate as MME and other handover-related functions, see Figure 4.3. With proper routing information from the controller, these entities can work asynchronously to handle handover in the network. Using these functional entities, we can potentially reduce the workload of the controller and make the handover process more efficient and the system more scalable.

In our architecture, we make the following assumptions:

- Each eNodeB is operating within its own data entity, and the latency within the entity is not calculated.
- Within each MHE, all functions are located in the same physical location, while the target and source entities are located in different physical locations.

As shown in Figure 4.3, our proposed EPC controller architecture consists of a basic controller, MHEs, OpenFlow-Switches, and eNodeBs.

Different from the centralized EPC controller architecture, the proposed EPC controller architecture divides the traditional controller into a basic controller and multiple MHEs. Hence, the control plane functionality is not exclusive to a single controller. MHEs serve functions such as MME, S-GW, and P-GW combined. They are placed physically close to eNodeBs and perform handover procedures asynchronously with the controller. On the other hand, the basic controller maintains the information of all the devices in the network. It is also responsible for updating flow tables in OpenFlow-Switches. The protocol of our proposed solution is illustrated in Figure 4.4.

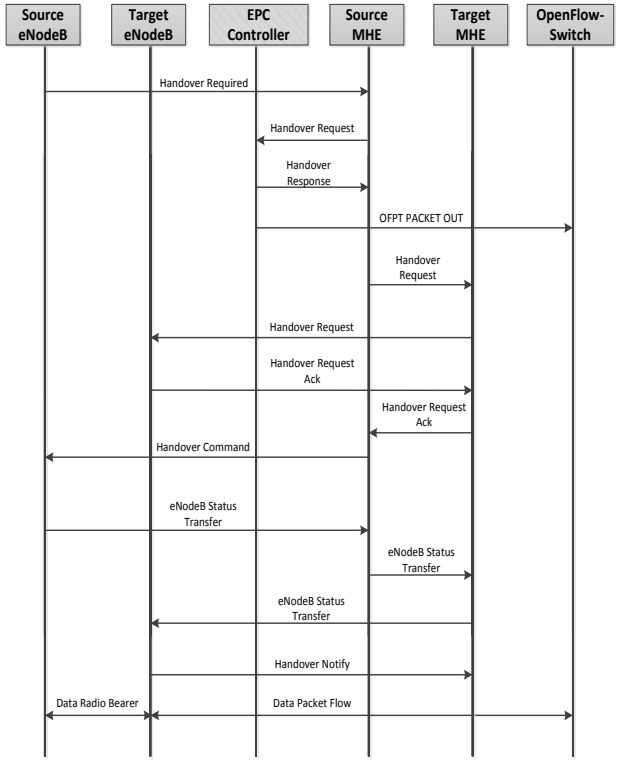
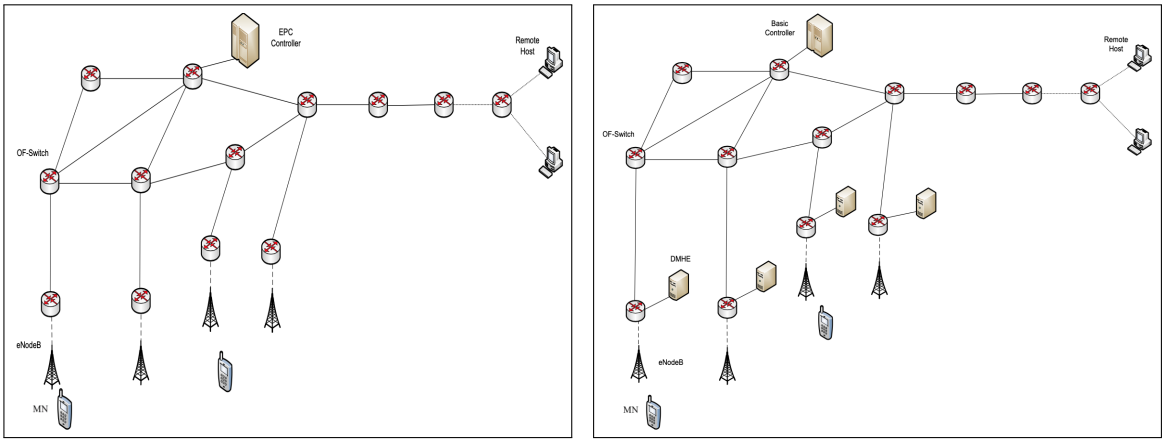


Figure 4.4: Proposed EPC controller handover procedure flow.

4.3 Simulation

We conducted a comparative analysis between the centralized EPC controller [24] and our proposed EPC controller architectures. This was undertaken with the widely used discrete event simulator NS3 [4]. In our experiment, we utilized the available cellular network modules and OpenFlow to simulate LTE functions in SDN architecture. Both architectures share the basic simulation topology. The link parameters we used for simulation are backbone link data rate = $100Mbps$ and Internet link data rate = $10Mbps$.

Figure 4.5a and Figure 4.5b show the network topologies for centralized and distributed EPC controller architectures, respectively.



(a) Centralized EPC controller [24]. **(b)** Proposed EPC controller.

Figure 4.5: Simulation topologies.

4.4 Evaluation

The handover latency and the average throughput per user are always crucial metrics across the different architectures of cellular networks. Primarily, the handover latency measures the time that elapses from the time the source eNodeB sends the handover request to the controller or to another handover handling entity until it receives the handover notification from target eNodeB. On the other hand, the average throughput per user describes the system's ability to transfer data with a handover for a user within a certain environment. It is a critically important metric to measure system usability and scalability. In fact, it measures the system's ability to deliver good services to users.

In this section, we study and analyze the aforementioned two metrics, the handover latency and the average throughput per user, to validate our proposed solution.

4.4.0.1 Handover Latency

Handover latency is a critical metric for measuring the handover procedure and to determine the level at which it affects data transferring. Note that the information exchange latency between the basic controller and MHEs is too small to be considered. As mentioned previously, the involvement of the basic controller is needed to provide information about the target MHE in case multiple MHEs are involved. In such a case, the controller and MHEs will be working asynchronously.

We performed simulations under different conditions. We generated UDP packets for each link from one end to the other with different occupancy percentages of background traffic to simulate a real-life environment. The percentage number indicated how much the link was already being used. In order to achieve an overall comparison of the two architectures, we took three scenarios into consideration: the idle network as 30% of background traffic, the standard network as 50%, and the busy network as 70%. In each simulation, we made mobile nodes handover from source eNodeB to target eNodeB and generated the

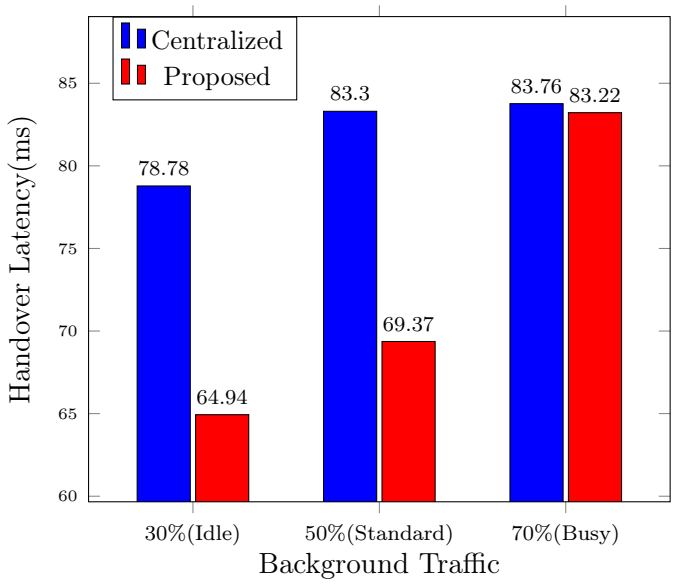


Figure 4.6: Handover latency under different traffic conditions.

sum of the latency of all MNs. The handover latency for two architectures was tested under conditions of different background traffic. As shown in Figure 4.6, for both architectures, the handover latency increased along with the growing background traffic from 30% to 70% whereas the proposed EPC controller architecture continuously outperformed the centralized EPC controller architecture with a smaller handover latency under the same background traffic. In standard traffic conditions, our architecture experienced 20% less handover latency than the centralized architecture. However, in busy networks (i.e., 70%), the performance of both architectures degraded in terms of handover latency. Regardless, these results indicate that our proposed EPC controller architecture has better performance when dealing with handover under the same circumstances.

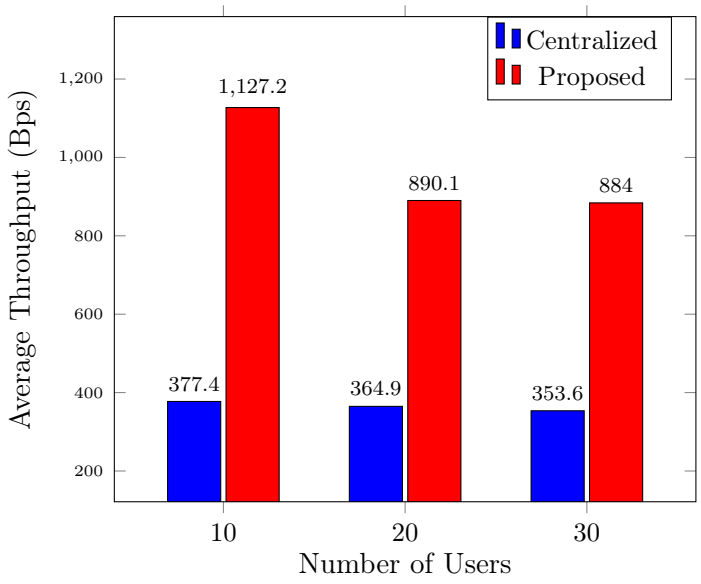


Figure 4.7: Average throughput per user.

4.4.0.2 Average Throughput per User

The average throughput is a critically important metric that measures the system usability and scalability. It can be calculated using the following formula:

$$V_T = \frac{B}{(T \cdot n)} \tag{4.1}$$

where V_T is the average throughput per user, B represents the total bytes successfully delivered, while T represents the time that elapses during the reception of the data, and n represents the number of users in the system. We set 20% of the users to perform handover while simulating a real-life environment. Each MN sent 5 packets per second to a remote host. The size of each packet was 38 bytes. After recording the data, Eq. 4.1 is applied to calculate the average throughput per user.

The average throughput per user for both architectures is recorded and calculated under the same background traffic. As shown in Figure 4.7, for the same number of users in each system, we can see that our proposed EPC controller architecture had greater value on throughput over centralized EPC controller architecture. With a growing number of users,

the average throughput for proposed EPC controller architecture decreased as it did for the centralized EPC controller architecture. Nonetheless, our proposed EPC still outperformed the centralized EPC under each user category. These results indicate that the proposed EPC controller architecture can provide better data service for each user under the same conditions.

In part, the proposed functionality-distributed EPC controller architecture sacrifices the general view held by the MHEs and may potentially increase the operational cost. However, it has distinct advantages in the areas of handover latency and carries a higher average throughput per user when compared to the centralized EPC controller architecture. As a result, this is a better option for service providers who prefer low handover latency and a high average throughput per user of the system.

4.5 Implementation Issues

According to the specifications of OpenFlow v.1.3.1, a switch initiates a *Port-status* message if a port was added, a port was removed, or an attribute of a port has changed. In our case, the switches have to send *Port-status* messages upon the connection/disconnection of mobile nodes. Note that in our system, MNs are connected to eNodeBs and not directly connected to the OpenFlow-switches. Therefore, the current deployment of SDN-based LTE systems should be modified to enable direct linking between MNs' mobility and the switches. To overcome this challenge, one option, which we assumed in our simulation, is to embed the OpenFlow-switches in the eNodeBs to make the OpenFlow-switch's interfaces connected directly to MNs. Another option is to make adaptations and modifications to the OpenFlow protocol. For this option, the modification should allow interpreting connection/disconnection alerts from eNodeBs into *Port-status* messages in the switches.

4.6 Summary

In this chapter, we have applied our proposed functionality-distributed solution to an LTE setting. We have performed a comparative analysis of a centralized EPC controller architecture and our architecture in the areas of handover latency and average throughput per user. Simulations have shown that the proposed distributed EPC controller architecture has better performance on handover latency and average throughput per user, as compared to the centralized approach under the same network conditions. In effect, our approach reduces the handover latency and improves the average throughput per user, which in turn, increase handover efficiency. In effect, OpenFlow-based SDN and mobile networks as well as distributed mobility management approaches have been extensively discussed in the past but so far not materialized into actual 3GPP networks. SDN is assumed to be applied for the backhaul services rather than directly involved in the operation of a mobile network. In this work, we have tried to fill in that gap and propose a solution that increases handover efficiency, especially in highly dynamic networks.

Chapter 5

Handling Inter-region Mobility Using MobileIP

In this chapter, we approach the handover delay problem by targeting another contributing factor to the processing delay of signaling messaging, which is the number of to-be-configured switches upon mobility, also known as the binding-cache placement problem. Due to the existing of dense and heterogeneous wireless networks nowadays, we got inspired to approach our problem in a multi-region setting. The different regions may represent different service providers, different wireless technologies, or different slices. Note that throughout this chapter, we use the two words domains and regions interchangeably. Moreover, we study a version of the MobileIP (MIP) protocol to handle handover in an SDN-based environment. Specifically, our aim is to minimize the total delay caused by handover among different regions. Our work in this chapter is divided into the following.

- We provide a brief introduction about the merge of MIP and SDN.
- We propose a hierarchical architecture that provides better scalability over the centralized architecture and less delay compared to the distributed approach.

- We have modified a solution proposed by a previous study [110] to solve the binding-cache placement problem. We use a heuristic approach.
- In our evaluation, we test our approach on two architectures and compare the results.

5.1 MobileIP in SDN

One of the earliest protocols used to handle handover in IP networks is MIP, an IETF standardized protocol [86]. This protocol has been used widely; however, it has some issues, such as inefficient handover, triangle routing, and high signaling overhead. After SDN was introduced, it seemed to solve these issues, so MIP has been investigated in this new setting. The separation between the control plane and data plane in SDN allows the mobility anchors to route the traffic data optimally, since locating mobile nodes is easily obtained by a central entity [127].

According to Wang et al. [109], MIP can be implemented using the SDN paradigm, where each MN, has ID-to-Locator mapping. The ID is the IP address given to MN wherever it moves, while the locator specifies the current MN's location and is the MN's first hop switch address. It goes without saying that the mapping (i.e., binding-cache) changes as MN moves from one switch to another. In fact, this movement may trigger inter-region handover. Note that, before a handover is triggered, a communication session might have been started with a correspondent node (CN). Therefore, traffic should be redirected to the MN's new location. Accordingly, two sub-functions are performed by the control plane. First, maintaining up-to-date ID-to-Locator mapping of each MN. That can be done by making a forwarding device (FD) report a new attachment to its controller (that is, locator). Second, updating or downloading an MN mapping to FDs involved. It is an optimization problem of choosing the smallest subset of FDs to be updated about the mapping rather than flooding or broadcasting one or more regions with each new update; thus, the signaling overhead is reduced. For this sub-function, there are two triggers:

- When a controller receives a notification about a new attachment, it updates some of the FDs with the new mapping, so the MN remains reachable for all its on-going sessions.

- When an FD explicitly requests an MN's mapping from a controller.

5.2 Architecture

The existence of large-scale dynamic mobile networks creates many challenges such as scalability, security, and availability. To overcome these challenges, we need to partition the network into multiple connected regions rather than having one big network controlled by a single controller. One of the primary drivers for a distributed mobility management is that a mobile network operator has to provide coverage to a large geographical area. Also, having different regions can represent multiple service providers of one technology, or heterogeneous technologies. Regarding inter-technology handover, it has been shown that SDN simplifies that kind of handover which is currently hard to manage in conventional networks [87].

In order to scale out any architecture, there are two approaches: *vertical approach* and *horizontal approach*. In the *vertical approach*, we have a hierarchy where a master controller controls and manages a set of other controllers and has a global view of the whole network. In the *horizontal approach*, a set of multiple controllers establishes peer-to-peer communication. However, if the control part is distributed among multiple controllers, these controllers need to maintain a consistent global view of the whole network, which in turn introduces higher messaging overhead. Taking into account a network's privacy, security, and policy, each region is not keen to expose its information with the other regions. Therefore, in this thesis, we study the *vertical approach*, in which we have two levels of controllers: a master controller, Root-C, and a set of distributed controllers, Dist-Cs, see

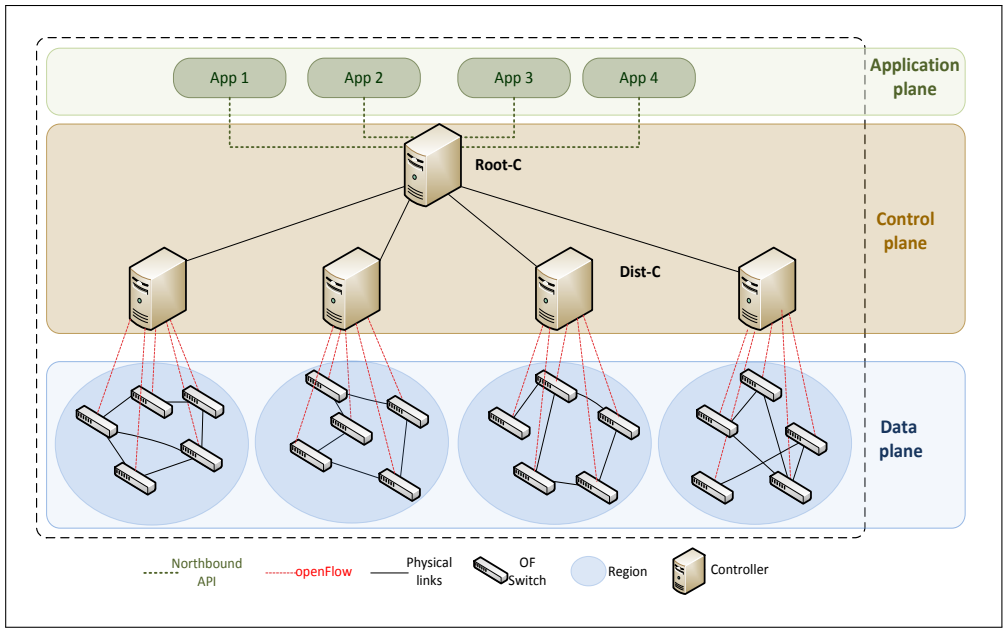


Figure 5.1: SDN hierarchical architecture.

Figure 5.1. Note that Root-C has the global view and thus the managing role. Root-C controls the other Dist-Cs, where each is managing a set of forwarding devices in a particular region. Hierarchy in the control plane has already been studied in the literature, for example, SOFTMow [73], and CROWD [14]. In fact, in other research domains, the benefits that the hierarchical architecture brings to the control level of any system has encouraged researchers to adopt it [28, 85].

Having a two-tier SDN controller hierarchy allows better aggregation of control information and reduction of signaling overhead [14]. Having a Root-C mitigates the need to make each controller exposed to all mobility-related information of every MN in other regions, the thing that needs a periodic distribution of status, consumes resources, and limits scalability. Therefore, in our work, we assume that different domains' administrations have agreed to appoint a coordinator among them.

The hierarchical architecture has several benefits over the other architectures. It overcomes the scalability issue imposed by the centralized approach, where a central controller cannot handle the increasing traffic growth over a large-scale area efficiently. On the other

hand, the flat distributed approach based on peer-to-peer communication is costly, since it requires maintaining a consistent view (i.e., exchanging status periodically) among the controllers. Also, dividing roles between controllers of different layers distributes the heavy load over the different controllers, also reducing overhead generated by control messaging exchange. Moreover, the hierarchical approach combines the benefits of both the centralized and the distributed approaches. It allows the Root-C to have the global view while allowing the distributed controllers of subsequent levels to handle increasing traffic growth, hence improving scalability. However, since this approach inherits some of the properties of both architectures, it is more complex and needs to be designed efficiently in such a way that enhances the intended metrics. Also the reliability may be affected, since it still has some of the centralized properties, such as a single point of failure or forming a bottleneck at the root controller.

Since we have two layers of controllers in addition to the forwarding devices, we assign duties to each layer as the following.

- **The Role of Root Controller**
  - Acquiring a global view.
  - Assigning fixed communicating nodes between regions, as we will see in Section 5.3.
  - Setting up a new connection to a Dist-C by updating its entries.
  - Handling multi-region mobility or communications.
  - Optimizing routing between two communicating nodes.
- **The Role of Distributed Controllers**
  - Assigning or allocating routable IP addresses to joining MNs and storing a mapping between this address and the locator.

- Notifying Root-C regarding new attachment and detachment of MNs.
- Handling mobility within their regions that includes updating flow tables and placing binding-caches.

- **The Role of Forwarding Devices**
  - Notifying their Dist-C of new attachments.
  - Forwarding according to their flow table.
  - Requesting their Dist-C when a mapping is missing.
  - Receiving updates.
  - Handling locator to ID conversion.

5.3 Handover Handling Protocol

In this section, we present the method we follow to handle mobility. But first, we should make some analysis regarding the locations of MN and CN during the handover. In a multi-region environment where more than one region is involved, Root-C should fulfill its managing role. Otherwise, a Dist-C handles the handover and communication within its region. In this work, we consider one case, when an MN moves from one region to another and communicates with a CN that belongs to a third region. Note that not only MN is mobile, but CN can also be mobile. However, we assume without loss of generality that CN is a stationary node.

When a node leaves its switch, which we call home-switch (Hs), this switch reports the disconnection to its Dist-C using *Port-status* message. This kind of notification is necessary because it allows faster redirection of on-going sessions. On the other hand, once MN attaches to a new switch, which is called foreign switch (Fs), a new connection notification is sent to its Dist-C and then Root-C.

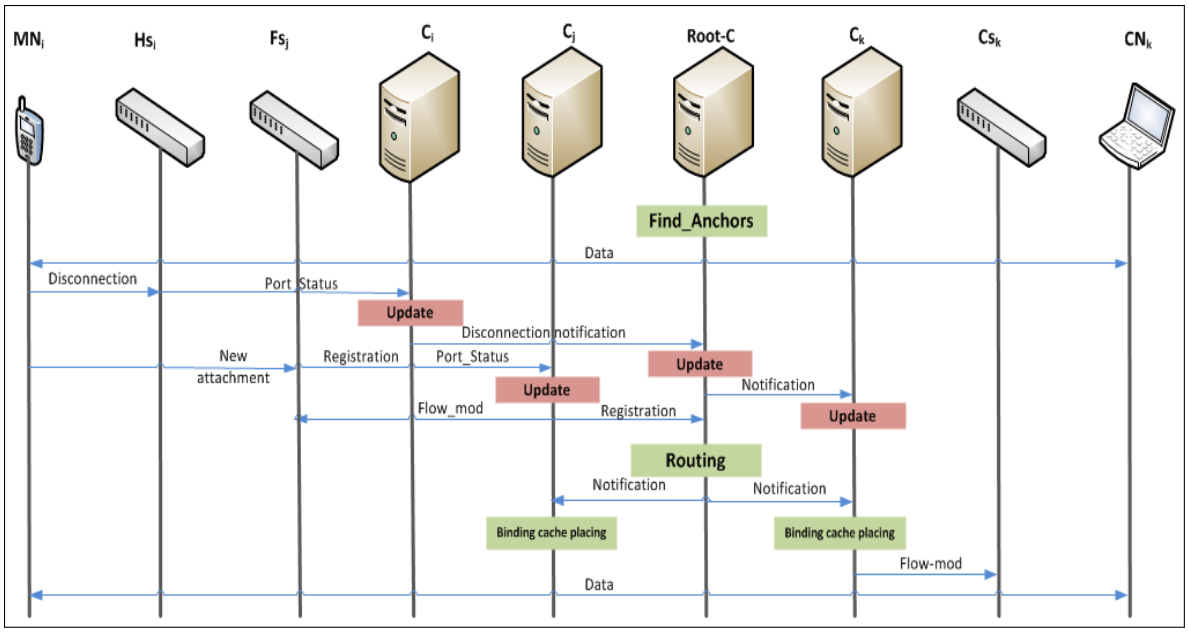


Figure 5.2: Handover protocol after a session initiated.

To study handover, the analysis can be divided into two scenarios: handover before a session is initiated and handover after a session is initiated. Here, we only consider the second scenario. In this scenario, we study the case in which a MN moves while there is an on-going session with a CN. The protocol flow is shown in Figure 5.2. This protocol ensures quick response to MN's movement upon Root-C receives a disconnection notification. Having such mechanism redirects traffic with minimal delay.

5.4 Algorithm

As traffic redirected from the old path to a new one upon MN's handover, a number of switches should be updated. To minimize the number of updates, we need to specify fixed points to be connecting switches among regions, we call them anchors. The choice of these anchors is predefined and based on distance function(s). The definition of the distance function is important to determine the path. In our approach, we incorporate two different distance functions.

- f_1 can be defined as protocol type in the case of heterogeneous networks
- f_2 can be defined as the geographic distance.

Definition 1 (Geographic Function)**.** *A function is a geographic distance function if and only if it satisfies the following:*

1. $dist(s_i, s_j) \geq 0 \ \forall s_i \in S(R_A), s_j \in S(R_B)$
   Given the region R_X, $S(R_X)$ will return the set of switches belonging to region R_X
2. $dist(s_i, s_j) = 0$ *iff* $s_i = s_j$.
3. $dist(s_i, s_j) + dist(s_j, s_k) \geq dist(s_i, s_k)$ *(Triangle inequality).*

Property 1. *if f_1 and f_2 are distance functions, then so is $f_1 + f_2$.*

The distance function can then be written as the following:

$$f = w_1 f_1 + w_2 f_2 \qquad w_1, w_2 \geq 0, \tag{5.1}$$

Here, w_1, w_2 are coefficients that the network operator can use to adjust the relative significance of the two distance functions based on whether the switches belong to one technology or different ones. Accordingly, it is important to assume that the distance between region A, (R_A), and region B, (R_B), is much larger than the diameter of a region.

Definition 2 (Region Diameter)**.** *Given that $S(R) = \{s_1, s_, ..., s_n\}$ is the set of switches in region R, then:*
$diam(R) \mapsto \mathbb{R}_{\geq 0}$ and $diam(R) = max\{dist(s_i, s_j) : (s_i, s_j) \in R\}$.

For our approach to work, we assume the following:

Assumption 1. *$dist(s_i, s_j) >> diam(R_K) \quad \forall (s_i, s_j) \in S(R_A) \times S(R_B)$, where $A, B \in \{1, 2, ..., M\}^2, A \neq B$, $K \in \{1, 2, ..., M\}$, and M is the number of regions.*

In our algorithm, we adopt a heuristic approach to minimize binding-cache placement updates; thus, we minimize the overall signaling messaging overhead and delay. By having predetermined anchor switches between regions by Algorithm 1, we affix part of the path to get a prefix that does not need to be updated after a handover takes place, see Figure 5.3.

Algorithm 1 is the initial procedure that is done by Root-C. Root-C chooses the best candidate anchors based on Eq. 5.1 to connect the different regions through. Within a region, the controlling Dist-C calculates the shortest path between two anchors, egress and ingress, along the path specified by Root-C. Also, that might include placing binding-caches in case any or some of that region's switches are needed to be updated. This involves finding what we refer to as *fork anchor*, see Algorithm 2.

Definition 3. *Fork Anchor, whether a fork anchor or switch, is the node from which* $path_{old}$ *diverges from* $path_{current}$.

5.4.1 Over-loaded Anchor Case

If an anchor becomes over-loaded, one solution would be to appoint another switch to replace the over-loaded anchor. However, this solution would defy the purpose of minimizing the number of configured switches. Also, it requires the multiple regions to have multiple connecting edge switches which is not always practical. From the anchor switch point of

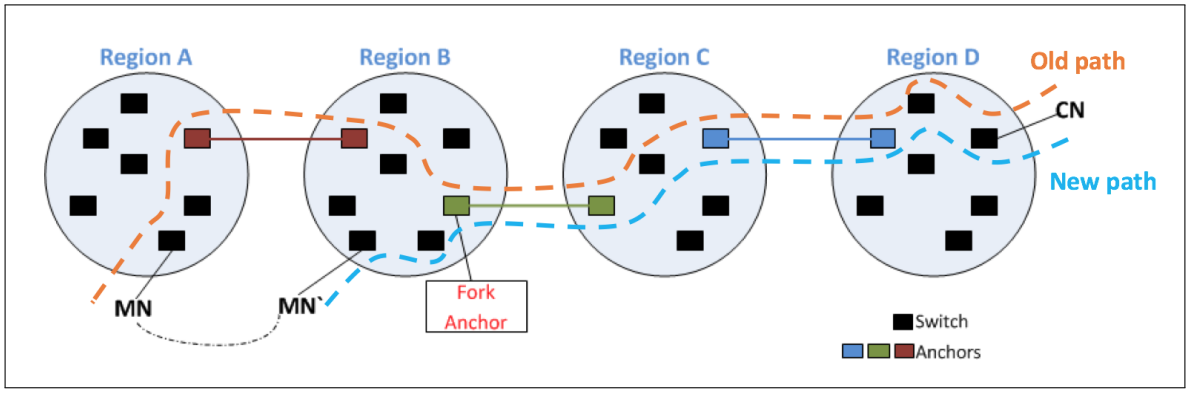


Figure 5.3: Fixed Anchors between regions.

Algorithm 1: Find_Anchors

Data: Candidate switches between regions
Result: Anchor switches between regions

```
1  begin
2      initialization
3      for R_A and R_B where A ≠ B do
4          find s_a, s_b where s_a ∈ S(R_A), s_b ∈ S(R_B) s.t. f(s_a, s_b) is minimized
5          Anchor_{R_A,R_B} ← (s_a, s_b)
6          RETURN Anchor_{R_A,R_B}
7      end
8      if Anchor_{R_A,R_B} becomes over-loaded then
9          place-wildcards to re-steer traffic
10     end
11 end
```

Algorithm 2: Binding-Cache Placement

Data: All MNs and CNs
Result: binding-cache placement for each connection.

```
1  begin
2      initialization
3      forall MN and CN do
4          Find shortest path through Anchors
5          Compare path_old with path_current
6          if Anchors(path_old) ≠ Anchors(path_current) then
7              Find fork Anchor
8              Place binding-cache on fork Anchor and beyond
9          else
10             Place binding-cache on fork switch and beyond
11         end
12     end
13 end
```

view, there are two types of traffic, one that is going through an affixed path based on our proposed algorithm, and the other type is every other traffic. As an extension to Algorithm 1, we recommend not to appoint another anchor, but instead to relieve the over-loaded anchor by re-steering other traffic to other switches until over-utilization state is resolved. This can be realized by placing wildcard rules to re-steer traffic, other than those involved through on-going affixed paths. Yet this solution comes with limitations due to TCAM limited capability [66].

5.5 Evaluation

5.5.1 Experiment

In our evaluation, we implemented our experiments on Mininet v.2.1.0 [3]. We used POX SDN controller [6] as our framework to communicate with the SDN data plane and to develop our network control applications. The simulation parameters are listed in Table 5.1.

Table 5.1: Simulation parameters.

Parameter	**Setting**
Tool	Mininet 2.1.0
Links Delay between switches	2 ms
Links Delay between switches and hosts	10 ms
OpenFlow Message	v.1.3.0
Controllers	POX
Test Tools	iperf v.2.0.5 ping
UDP Datagram	1470 Byte
UDP Buffer Size	208 KByte
TCP Window Size	15 KByte

We set two experiments based on the architecture of controllers: hierarchical and distributed (as proposed in [110]). In the hierarchical setting, we set a network of a Root-C and two sub-controllers, C1 and C2, as depicted in Figure 5.4. In the distributed setting, we had two controllers that constructed a peer-to-peer relationship. In both settings, each controller, except for Root-C, is responsible for a separate domain to represent either different service providers or different technologies. As mentioned earlier, mobility happens between different domains, which in turn involves the intervention of at least two controllers. We used a TCP socket as our messaging system to get messages moving between POX controllers.

In the hierarchical setting, Root-C coordinates the controlling messaging between C1 and C2 to handle traffic re-routing after a host, h1, moves from region of C1 to region of

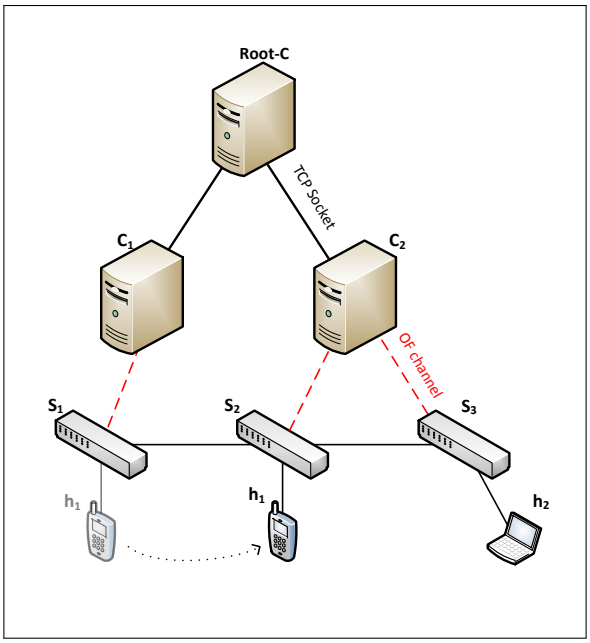


Figure 5.4: Simulation topology of the hierarchical architecture.

C2. While in the distributed setting [110], controllers involved in the handover exchange controlling messages to coordinate handover and re-routing of on-going traffic. In other words, they need to maintain a consistent view of the network mapping. As we show in the result section, maintaining a consistent view among controllers has its toll on delay.

Since we used Mininet as our simulator, we had to code the mobility of h1 by detaching it from its old port and then attaching it to a new port; the movement of h1 mimicked the hard handover scheme. Based on the purpose of our experiment, the POX controllers have three event handlers: connection-up, packet-in, and port-status. The controllers proactively install flow entries during the connection-up stage in order to minimize initial time for set up. Then they dynamically change the switches' flow entries upon arrival of *Port-status* events.

As our first performance measuring tool, we started UDP iperf traffic between h1 and h2 for 20 seconds, and we reported results every 0.5 second. Then at the seventh second, h1 moved to s2. Meanwhile, we recorded the changes of that on-going session in term of throughput and jitter. Also, we ran TCP iperf traffic between the two hosts under the

same circumstances as UDP traffic, and recorded the changes on TCP sequence values. Moreover, as another testing tool, we used ping to estimate the packet loss during that session.

5.5.2 Results

In our experiment, we tested the responsiveness of a hierarchy of controllers against a distributed set of controllers [110]. For our results, we recorded the impact of handover and how the two settings of controllers handle handover between different regions and re-steering of traffic. The performance metrics we collected were UDP throughput, UDP jitter, packet loss, and TCP sequence numbers.

As shown in Figure 5.5, at the seventh second, the throughput drops to 0 due to mobility of h1 in both settings. As shown, the hierarchical setting restores traffic after one second. The distributed setting however needs almost an extra half-second to re-steer traffic to the new location of h1. The reasoning behind that difference is that upon *Port-status*, messages are sent to controllers and then to Root-C, anchors are appointed by Root-C to be configured according to our Algorithm 1. Note that, based on our approach, s2 is the only fork node that needs to be configured. On the other hand, distributed controllers have to react differently, since they have to exchange more control information to redirect on-going traffic and decide on the fork node(s) to be configured.

For applications that are sensitive to the variation in delay such as Voice over IP (VoIP), mobility would cause a high jitter. As shown in Figure 5.6, both settings experienced high jitter due to mobility of h1, but the hierarchical setting restore its steady rate of flowing packets faster.

Regarding the results we collected for TCP traffic, we used tcpdump [9] as our packet analyzer, and Wireshark [10] to capture TCP traffic. As presented in Figure 5.7, the captured packets' sequence numbers halted at around the seventh second due to mobility

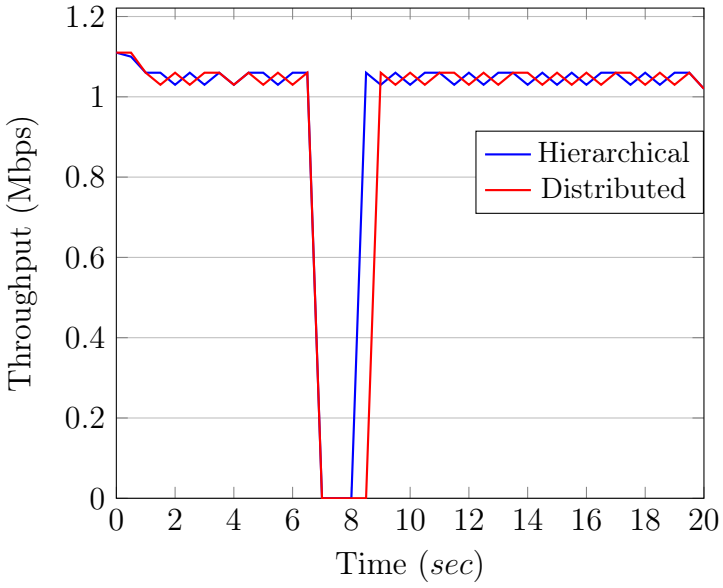


Figure 5.5: Hierarchical and distributed architectures impact on UDP throughput.

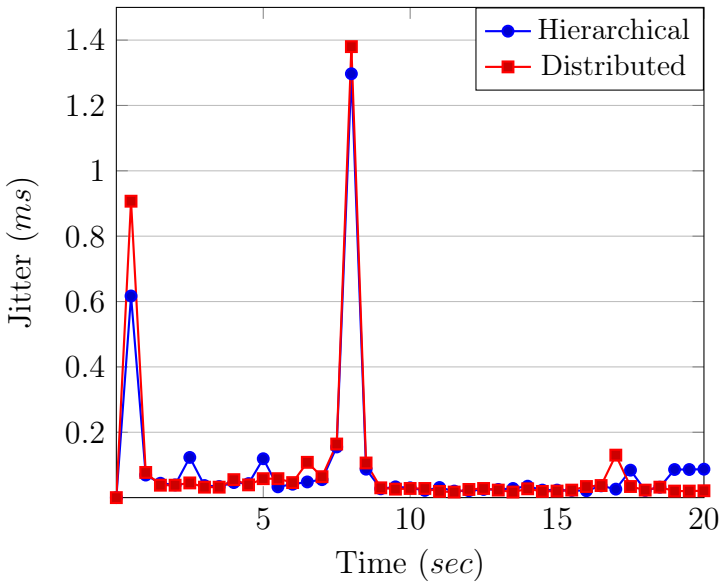


Figure 5.6: Hierarchical and distributed architectures impact on jitter.

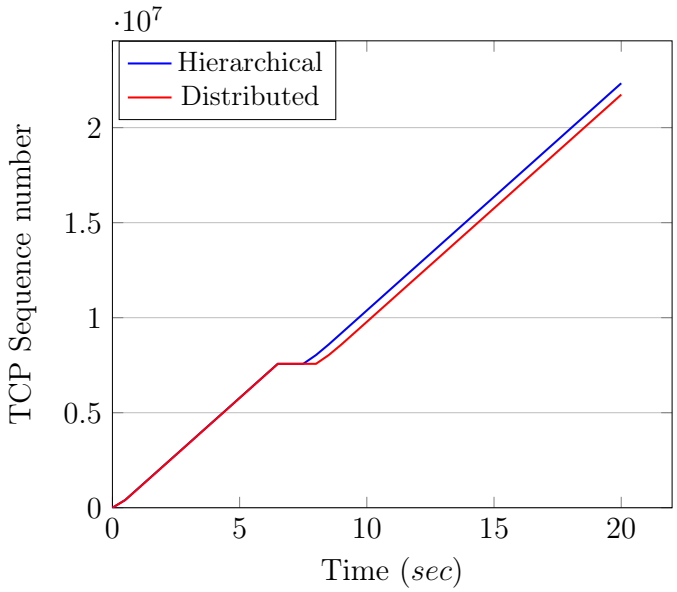


Figure 5.7: TCP sequence values of simulated hierarchical and distributed architectures.

in both settings. Then, when traffic restored, the sequence number of packets in both setting started increasing. There was a difference in the restoration time, again due to the fact that the distributed architecture introduced a higher delay. As for the packet loss, the hierarchical architecture experienced 5% packet loss, whereas the distributed architecture showed a slightly higher percentage, almost 7%.

In the light of our experiments, in the distributed setting, controllers have to reactively exchange their up-to-date view of their domains. That kind of information is important for configuring switches and re-routing, but it can increase overhead and elongate service disruption. Not to mention that the more domains involved, the higher the overhead. However, we argue that the hierarchical setting can perform differently and efficiently, since Root-C has the global view, which in fact plays a major role in minimizing delay.

5.5.3 Discussion

In this section, we briefly present a comparison between three architectures (centralized, distributed, and hierarchical) in a mobile network where MIP is used, refer to Table 5.2.

In our comparison, we compare the centralized approach proposed by [109] against the distributed approach in [110], and our hierarchical architecture. Regarding each criterion, the least satisfying architecture is left without a check mark. As the table is showing, there are trade-offs among the different criteria. Indeed, a network designer should decide on which architecture based on the network size, purpose, and compromises.

Table 5.2: A comparison between three architectures.

Criteria / **Architecture**	GlobalView	Reliability	Complexity	Scalability	SPoF
Centralized [109]	√				√
Distributed [110]		√	√	√	
Hierarchical	√		√	√	√

5.6 Summary

All in all, in this chapter, we have presented our approach to handle handover efficiently in a multi-domain SDN-based networks using a modified version of MIP protocol. We have modified an existing work to fit the hierarchical setting of controllers. We have proposed a heuristic algorithm that minimizes the number of binding-cache placement in switches, which in return minimizes overall delay of handover signaling messages. We have shown that the hierarchical architecture can perform better compared to other architectures, such as the centralized and distributed ones. We have conducted some experiments to evaluate our approach against a distributed approach. Indeed, the hierarchical architecture has some flaws, e.g., its complexity; it requires approval among different administrations to be coordinated; and the existence of a single root controller imposes a single point of failure. However, we argue that in highly-dynamic and heterogeneous networks, choosing a hierarchy of controllers provides better scalability over the centralized architecture and less delay compared to the distributed approach.

Chapter 6

Improving the Response Time of SDN Controllers based on Vertical Mobility

Incorporating the SDN paradigm into mobile networking has its strengths and weaknesses. Aside from the promising benefits that SDN brings to the realm of networking, there are still some challenging issues. For instance, integrating heterogeneous networks, the drastic increase in using real-time demanding applications, combined with mobility, have led to a significant increase in the control traffic and can burden managing controllers. Therefore, we need dynamic adjustments and a redistribution of the load among the controllers to maintain an acceptable level of QoS being delivered to mobile users. In this chapter, we address the problem of balancing the load among a set of controllers which are managing heterogeneous wireless networks in order to minimize the response time of an over-loaded controller. We propose a framework that employs vertical handovers to enable load balancing. Our main metric is the controller response time, since it affects the completion of any procedure associated with mobile users. The response time can be defined as the sojourn time of a request at the controller from its arrival until a reactive rule is sent back to one or more switches.

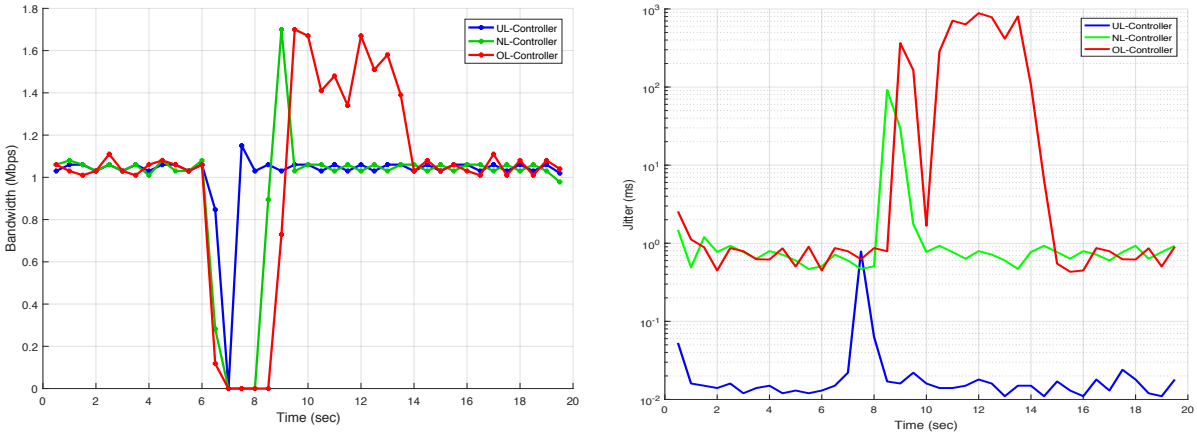
Regarding the handover procedure, in this part, we target the decision before a handover is in effect to make sure that the upcoming/handed-over users will experience a satisfactory level of service and mitigate long delays or breakage. That cannot happen if the new network's controller is over-loaded; therefore, we need to ensure having efficient load balancing. Our primary contribution, in order to balance the load in such a setting, is a novel approach that mainly aims at vertically handing over some edge users, while considering some context information regarding the users and controllers. Consequently, a controller's response time to any mobility-related procedure will decrease.

The work in this chapter is divided into the following:

- We discuss the impact of different controller loads on the handover procedure.
- We discuss some of the approaches that have addressed load balancing among multiple controllers and how they can relate to our direction of research.
- We describe our system components.
- We then go through the modelling and formulation of our system.
- We propose a management framework that includes three main aspects. First, we identify candidate users based on their context information. Second, we propose a novel mechanism that reduces the frequency of load informing between multiple controllers, and hence, reducing processing and communication overhead. Third, after the candidate users are determined, we optimize the decision problem on the selection among several candidate networks.

6.1 Controller Load and Handover

Handover, along with other procedures associated with mobile users, can exhaust the SDN controllers' resources. Thus, load balancing can be achieved earlier, as a part of the



(a) UDP throughput during a handover.

(b) UDP jitter during a handover.

Figure 6.1: Impact of different loads on handover during a udp traffic.

handover preparation stage, to alleviate longer delays due to saturated buffers that are unable to process handover decisions (i.e., signaling messages) efficiently.

Different controller loads have different impacts on the completion of a critical procedure such as a handover. To demonstrate this, we conducted an experiment by running a UDP traffic between two hosts, where one is mobile. Figure 6.1 shows the effect on two metrics, UDP throughput and jitter. As it is depicted, we started the UDP traffic between two nodes for 20 seconds. As one of them moved from its point of attachment (PoA) to another, a hard handover happened at the seventh second; we recorded the measured throughput every 0.5 seconds. We repeated the experiment three times under different loads in terms of the number of OpenFlow messages handled by the managing controller, when the controller is under-loaded (UL), normally-loaded (NL), and over-loaded (OL). The recorded numbers of OpenFlow packets handled by the managing controller were 1078, 4293 and 14081, respectively.

Figure 6.1a reflects the time a controller took to handle a handover under different loads. It was seen that the handover delay was higher when the controller handled more OpenFlow packets. That being said, as the number of packets increased, the controller response time increased as well, and thus, a handover's completion took longer; the handover delay has

almost tripled as the controller's load changed from under-loaded to being over-loaded. This worsens the user experience during a hard handover.

Regarding jitter, a high variation in delay degrades the user experience while using sensitive applications such as VoIP. As Figure 6.1b shows, as the controller load increased the variation in delay prolonged, thus disturbing or even disconnecting the service.

The results of the experiment found clear support for the importance of load balancing in minimizing handover delay, since the controller response time is actually a main contributor in handling handover signaling messages.

6.2 Multi-Controllers Load Balancing Solutions

Scalability of the control plane is a challenge for the SDN control logic centralization [85]. The negative impact of such a challenge is partially mitigated when the control logic is distributed/delegated into either a one level of controllers (i.e., flat) or two levels of controllers (i.e., hierarchical) [112].

In the multi-controller setting, load balancing is crucial. Prior research has shown that the load-balancing is correlated to several factors (i.e., design choices), including the type of mapping between switches and controllers, and the network state dissemination method.

6.2.1 Switch-Controller Assignment

An important factor that contributes to a load-balancing mechanism's efficiency is the mapping between the controllers and the switches [57, 107]. The switch-controller deployment can either be fixed (i.e., static) or dynamic. Each deployment method has its advantages, disadvantages, and applications.

6.2.1.1 The Static Assignment

The static assignment means that the same groups of switches are connected to specific controllers without changing their assignments at any point during running the system. The following summarize the pros (+) and cons (-) of this assignment:

(+) Simple control plane management.

(+) Reduced infrastructure and complexity costs due to its static nature, so there is no need to migrate or change connection.

(+) Flexibility to be applied to connected or disconnected domains.

(-) Some controllers are susceptible to overloading and failure due to heavy and fluctuating loads [57, 99].

(-) Inefficient resource utilization when the load shifts, compromising the network ability to efficiently react to changes, such as failure and updates [64].

6.2.1.2 The Dynamic Assignment

In contrast to the static assignment, in the dynamic assignment, the mapping between switches and controllers can be altered at any point to achieve particular tasks, e.g., load balancing [99]. One main enabler for this deployment is switch migration. The target controller of a migrated switch should obtain enough information about the new switch from its previous controller in order for a migration to be completed. The pros (+) and cons (-) of the dynamic assignment are as follows:

(+) More adaptability to sudden network events.

(+) Improving the network QoS [57], resource utilization [99], and security [114].

(-) High deployment cost, since each switch is physically connected to multiple controllers.

(-) High control overhead due to longer re-association delay that can be unacceptable; for instance, in Opendaylight Beryllium, the re-association delay can reach up to 800ms [107].

(-) Complexity of migration protocols [29].

(-) Switch migration frequency has to be limited in number and time as frequent and longer changes in mappings can lead to network instability [99].

(-) This assignment is not suitable for disconnected graphs, isolated domains.

6.2.2 State Dissemination

To acquire a global view of a whole network, multiple distributed controllers need to exchange their load statuses in a periodic manner [122]. Besides the switch deployment issue, there are other issues that contribute to degrading the performance of any control plane load-balancing mechanism; specifically, the controllers status synchronization. There have been numerous studies to investigate the synchronization overhead [60, 122, 126]. There are issues related to maintaining a consistent view of the network such as stale information, overhead due to frequent updates, and privacy and security concerns.

6.2.3 Previous Work

Several solutions have been proposed to the multi-controller load-balancing problem, some focusing on the static deployment, others on the dynamic deployment while taking their limitations into account. Some approaches have also focused on minimizing the overhead imposed by exchanging the status among the controllers. A summary of some scholars' contributions are listed in Table 6.1.

Table 6.1: Comparison between some load-balancing approaches scholars.

Ref.	S-C deployment	Context	LB Method	Advantages	Limitations
[55]	Static	Distributed-DataCenter	State disseminating using NIB	Scalable, simple control platform	Susceptible to overloading (hindering QoS)
[107]	Static	Cluster of controllers-DataCenter	Flow re-directing	Minimizing response time drastically	No functionalty-distribution among controllers, hardware limitations
[118]	Static	Hybrid wireless sensor net.	Grouping controllers into clusters	Reducing load jitter (stable)	Overhead and throughput not evaluated
[43]	Static (multiple controller-switch connections)	Hierarchical WAN	Partitioning traffic, allocating controllers based on coarse-grained flows	Reducing flow-setup time	Complexity, costly deployment
[106]	Static	Datacenter, WAN	Rounding-based algorithm to solve the routing problem	Minimizing response time.	Status exchange strategy not clear
[99]	Dynamic	Distributed controllers, large-scale net.	fractional flow migration	resilient, flexible flow mapping	Synchronization overhead not investigated
[126]	Dynamic	Distributed controllers	Adaptive load collection	Balancing the whole system load	High overhead, switch migration cost not investigated
[122]	Dynamic	Distributed controllers	Inhibition algorithm for load collection and informing	Increasing controllers throughput	Synchronization overhead not evaluated
[60]	Dynamic	Hierarchical controllers	Adaptive threshold, distributed database	Achieving overall even load distribution	Database retrieval delay, no evaluation of overhead
[92]	Dynamic	Hierarchical WAN	Cost-greedy algorithm	Short imbalance state periods	Overhead due to frequent syncronization
[27]	Dynamic	Distributed-DataCenter	greedy switch selection	considering multiple over-loaded controllers	Overhead due to simultaneous migrations

Continued on next page

Table 6.1 – continued from previous page

Ref.	Deployment	Context	LB Method	Advantages	Limitations
[125]	Dynamic	Distributed-DataCenter	On-line light weight algorithm	Reducing average response time based on real-time requests distribution	Overhead due to frequent switch re-associations
[36]	Dynamic	WAN	Managing num. of controllers and switches	Effective management, scalable control plane	High complexity

The applicability of previous strategies can be seen in practice in datacenter or WAN, assuming the interconnectivity between different regions. However, if these regions are autonomous and administratively disjointed, some of the aforementioned approaches may fail. With such limitation, the static assignment is the sole option. Even though the static assignment is restrictive, its shortcomings can be alleviated in different ways such as with the following:

1. Flow redirection.
2. Intelligent routing.
3. Controller placement.
4. Exploiting users' mobility.

Our approach is based on the static deployment, yet we exploit the dynamism of mobile users to shift loads from over-loaded controllers. However, as far as we know, no prior study has addressed disconnected domains; which can be a severe challenge for most load balancing strategies.

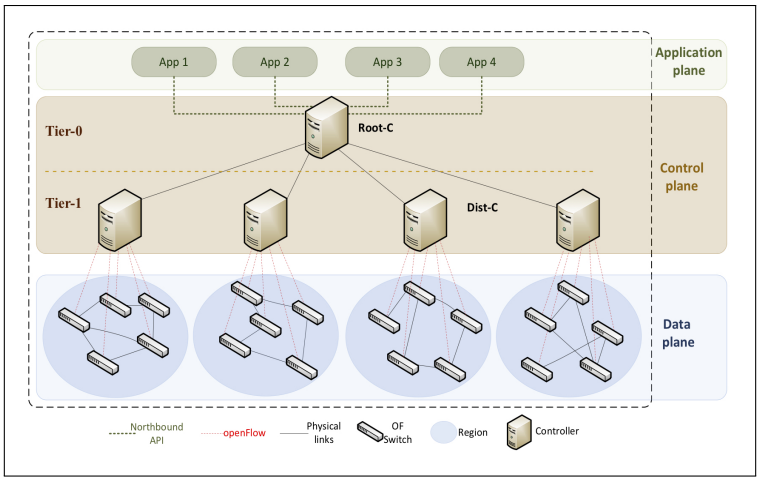


Figure 6.2: SDN hierarchical architecture.

6.3 System Description

We consider integrating the SDN paradigm into a network of heterogeneous technologies, such as LTE, Wi-Fi and WiMAX. In our work, our system model is depicted in Figure 6.2. We propose a hierarchy of controllers forming the control plane, where tier-1 controllers are different service providers (i.e., Dist-C) that manage different domains/technologies. Each controller only manages a domain of a particular wireless network. We omit direct communication among them to ensure privacy, confidentiality and guarantee that security standards are met. Additionally, a tier-0 controller represents the Root-C, where the connection and management between the different domains take place. Considering SDN-based heterogeneous networks, we argue that a hierarchical architecture provides better scalability over a centralized architecture and less delays, compared to a flat approach.

All the traffic that goes between the switches and controllers (i.e., through any south-bound protocol) is control traffic. We adopt the OpenFlow southbound protocol as an enabler for our model.

As for the data plane, the Point of Attachment (PoA), whether they are base stations (BS) or access points (AP), are each connected to a switch. MNs are hence, connected to PoAs. Within a domain, the set of MNs can be divided into edge-users (MN_e) that are

within the coverage of other domain(s), and non-edge-users (MN_{ne}) that are within the coverage of only their current domain. Edge-users can be further divided into candidates and non-candidates, based on their mobility parameters, including the preference and position as we will explain in the next sections.

6.4 Modeling and Formulation

We consider a discrete-time model, where the length of each time slot matches the time scale at which the control requests can be recorded.

Our network is divided into M domains, where each domain is operated by a service provider, single controller, and may represent a different wireless technology; an example is provided in Figure 6.3. Thus, we have a set of M distributed controllers, C_i, where $i \in \{1, 2, ..., M\}$. Each C_i has a capacity in terms of the maximum number of control requests it can handle per a unit of time t, which is given as α_i. In order to handle traffic peaks and unexpected computational demands, we advise including a spare processing capacity, or a so-called decay factor β_i, for each C_i. The decay factor has a value of

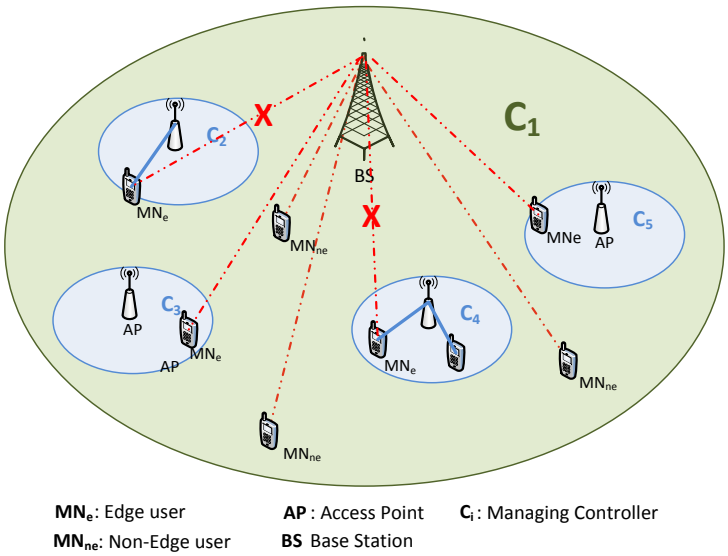


Figure 6.3: The system model: consisting of a macrocell domain and multiple Wi-Fi domains.

$\beta_i \in [0, 1]$; thus, the total capacity of C_i can be written as: $\alpha_i \beta_i$. Note that the decay factor can provide a network operator with the flexibility to adjust the capacity given different times of the day, congested places or even handle synchronization overhead with the Root-C. On top of the set of controllers and connected to each of them is Root-C. The capacity of Root-C is not taken into consideration, as it does not handle control requests generated by the users; thus, the load on Root-C is beyond the scope of this work.

Each area i consists of a set of inter-connected switches $s_j^i \in S$ where, $j \in \{1, 2, ..., N\}$, and S represents the set of all switches. We define the switch-controller adjacency matrix X as a binary $N \times M$ matrix that associates a group of switches to each controller. Note that each switch is connected to exactly one controller.

$$X = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_N \end{array} \begin{array}{c} \begin{array}{ccccc} C_1 & C_2 & C_3 & \dots & C_M \end{array} \\ \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1M} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2M} \\ x_{31} & x_{32} & x_{33} & \dots & x_{3M} \\ \dots & \dots & \dots & \dots & \dots \\ x_{N1} & x_{N2} & x_{N3} & \dots & x_{NM} \end{bmatrix}_{N \times M} \end{array} \tag{6.1}$$

Likewise, we define the user-switch adjacency matrix Z_{s^i} for each switch in region i to denote k associated users to n switches at time t as follows:

$$Z_{s^i} = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_k \end{array} \begin{array}{c} \begin{array}{ccccc} s_1 & s_2 & s_3 & \dots & s_n \end{array} \\ \begin{bmatrix} z_{11} & z_{12} & z_{13} & \dots & z_{1n} \\ z_{21} & z_{22} & z_{23} & \dots & z_{2n} \\ z_{31} & z_{32} & z_{33} & \dots & z_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ z_{k1} & z_{k2} & z_{k3} & \dots & z_{kn} \end{bmatrix}_{k \times n} \end{array} \tag{6.2}$$

where $z_{bj} \in \{0,1\}$, and $\sum_{j=1}^{n} z_{bj} \leq 1 \; \forall b$, where $b = \{1,2,..,k\}$. Knowing that the main focus of our work is on the hard handover, each user is connected to exactly one switch at any time t. Each switch is directly connected to a PoA, so we can refer to both s_j^i or its PoA as one entity. There are a limited number of channels provided by any PoA, where each channel can be assigned to one user. Let $l_{s_j^i}$ denote the number of channels provided by s_j^i; then, there are at most $l_{s_j^i}$ users connected to s_j^i, which means $\sum_{b=1}^{k} z_{bj} \leq l_{s_j}$.

As for the mapping between the users and controllers, it can be presented as a $K \times M$ adjacency matrix as the following:

$$Y_t = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_K \end{array} \begin{array}{c} \begin{array}{ccccc} C_1 & C_2 & C_3 & \dots & C_M \end{array} \\ \begin{bmatrix} y_{11} & y_{12} & y_{13} & \dots & y_{1M} \\ y_{21} & y_{22} & y_{23} & \dots & y_{2M} \\ y_{31} & y_{32} & y_{33} & \dots & y_{3M} \\ \dots & \dots & \dots & \dots & \dots \\ y_{K1} & y_{K2} & y_{K3} & \dots & y_{KM} \end{bmatrix}_{K \times M} \end{array} \tag{6.3}$$

The adjacency matrix above shows the connectivity between the users and controllers, where $y_{xm} \in \{0,1\}$. At any time t, each user has to be connected to only one controller; thus, the adjacency matrix reflects that: $\sum_{m=1}^{M} y_{xm} = 1$, where $x = \{1,2,...,K\}$, and the number of users attached to each controller can be obtained from: $\sum_{x=1}^{K} y_{xm}$.

Flows (i.e., control traffic) that are reported to C_i by switch s_j^i, at time t is denoted as $\xi_{s_j^i}(t)$; therefore, the load on C_i, in terms of the total number of control requests, is given as (θ_i) and can be modeled as:

$$\theta_i = \sum_{s_j^i \in S^i} \xi_{s_j^i}(t) \tag{6.4}$$

Defining the load is essential to model the behavior of the controllers, since the load is closely related to the response time of the control traffic as the Little's theory showed [63]. In line with the results of [106], and inspired by their approach, we model Dist-C as

$M/M/1$, and we assume that the flow requests follow the Poisson distribution. Therefore, the average response time of C_i, given its capacity and load, can be defined as:

$$R_{C_i}(t) = \frac{1}{\alpha_i \beta_i - \theta_i(t)} \tag{6.5}$$

In our model, $\bar{U}^i$ represents the set of attached users to C_i, $\bar{U}^i = \{u_1, u_2, ..., u_k\}$. $\bar{U}^i$ can be further divided into edge users (MN_e^i) that reside in the overlap area with another domain, as in Figure 6.3, and non-edge users (MN_{ne}^i) that represent all other users. Accordingly, the total attached users of a region i can be represented as $\bar{U}^i = MN_e^i \cup MN_{ne}^i$, where $|\bar{U}^i| = k$. These users are currently, at time t, are utilizing the C_i's resources.

The set S^i is composed of two subsets of switches, edge switches S_e^i and every other switch S_{ne}^i. Intuitively, users connected to the edge switches are edge users. This means;

$$\forall u_x \in MN_e^i \quad \exists\, s_j^i \in S_e^i \text{ , where } z_{xj} = 1$$

Naturally, the expected change in user attachments is reported by a subset of edge switches, S_e^i. Note that not all the traffic generated by the switches are user-related; there is other management-related traffic, including statistics [84]. Out of each switch s_j^i, a set of control messages is generated by user u_x with variable rates c_x^i (modeled as Poisson [92, 108]); then, having k attached users to the area of controller C_i, at time t, makes the following:

$$\theta_i - \sum_{s_j^i \in S^i} c_j = \sum_{u_x^i \in \bar{U}} c_x^i \tag{6.6}$$

where c_j is management-related traffic generated by the switches; however, they are relatively small compared to the user-related traffic [16] and can be neglected. Therefore, we

define the remaining capacity of C_i as (r^i), as follows:

$$r^i(t) = \alpha_i \beta_i - \sum_{u_x^i \in \bar{U}^i} c_x^i(t) \tag{6.7}$$

Proposition 1. *One main influencing factor on a controller's load, and hence its response time, is the number of attached users.*

$$\theta_i(t) = (|MN_{ne}^i + MN_e^i|)c_x^i \tag{6.8}$$

As a result, we can minimize the controller's load by reducing the number of attached users to that controller.

The traffic going through southbound channels for connected users, whether *Packet-in*, *Port-status* or any control requests, is better processed by controllers providing shorter response times. Therefore, our objective function is to minimize the maximum response time of an over-loaded controller, as follows:

$$\textbf{min} \quad max\ R_{C_i}(t) \tag{6.9}$$

subject to:

$$\theta_i \leq \alpha_{th_i} \tag{6.10}$$

$$\sum_{m=1}^{M} x_{jm} = 1 \quad \forall\, s_j \in S \tag{6.11}$$

$$\sum_{j=1}^{n} z_{bj} = 1 \quad \forall\, u_b \tag{6.12}$$

$$\sum_{b=1}^{k} z_{bj} \leq l_{s_j} \quad \forall\, s_j \in S \tag{6.13}$$

The set of the above constraints impose the following. No controller is overloaded, meaning that the load of a controller is less than that of its pre-defined threshold, α_{th_i}, (Eq. 6.10). Each switch is managed by exactly one controller (Eq. 6.11). To enforce hard handover, each user has to be connected to exactly one switch at a time (Eq. 6.12). Each switch/PoA has a limited number of channels that it can provide to the users (Eq. 6.13).

In this case, load-balancing among tier-1 controllers is required to relieve an over-loaded cellular network controller. Therefore, we propose a heuristic approach to balance the load among the controllers based on context-aware vertical mobility. We design a framework that tackles three main aspects. First, it relieves the over-loaded controller by exploiting the vertical mobility whenever it is applicable. Second, it reduces the status synchronization overhead. Finally, it optimizes the network selection for handed-over users.

6.5 Management Framework

Our management framework is constructed and distributed into a two-tier control plane. It is based on a collaboration between modules that reside on both Dist-Cs and Root-C. So, our proposed approach (proposed-LB) is not operated entirely by Root-C; we divide the functions into local ones done by Dist-Cs, and global ones done by Root-C, refer to Figure 6.4. Consequently, the exchanging overhead between Dist-Cs and Root-C is minimized. The five main modules that shape our load balancing framework are:

- The monitoring module.
- The controller status module.
- The user context module.
- The load balancing decision module.
- The network selection module.

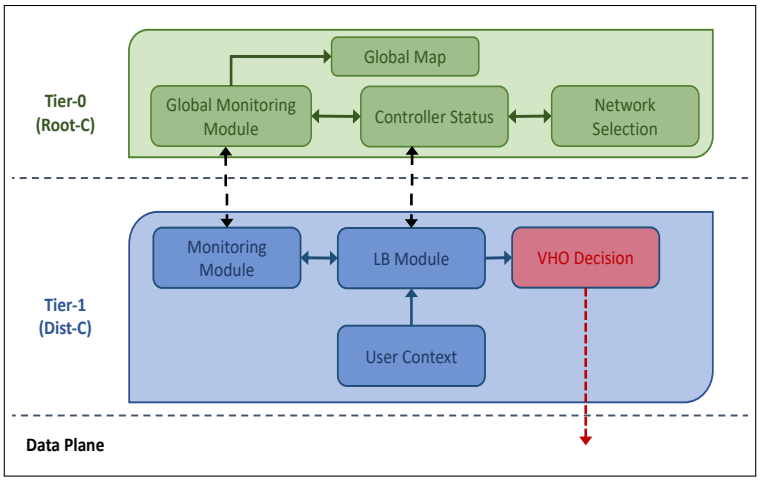


Figure 6.4: A two-level load balancing management framework based on context-aware handover.

6.5.1 Monitoring Module

This module has a global version placed on Root-C, and a local version on Dist-Cs.

The local module gathers the statistics (i.e., stats) that reflect the switches' state regarding flow tables and port stats. For example, a controller can periodically collect port, table and flow information within its domain through the use of particular stats OpenFlow messages, such as OFPST_PORT, OFPST_FLOW, and OFPST_TABLE [16]. The local module is a part of the Dist-Cs' frameworks, as controllers fetch stats information from their switches regularly to check the liveliness of connections and maintain an updated view of its network.

The global module placed at Root-C periodically collects load state information from Dist-Cs. Based on the controllers' loads at time t, they report their load information to Root-C. As maintaining a consistent view among the distributed controllers can result in significant overhead, we need to design a mechanism that reduces the frequency of load informing, and hence, reducing processing and communication overhead. That leads us to our proposed status dissemination strategy as part of the network status module.

6.5.2 Controller Status Module

The authors of [122] proposed an inhibition algorithm to suppress an unchanged controller status from being reported to other controllers; otherwise, it is a redundant information. Inspired by [122] to reduce the frequency of status exchange, we propose the following load informing strategy. We define two thresholds; maximum threshold and minimum threshold as $\overset{+}{\alpha}_{th}$, and $\overset{-}{\alpha}_{th}$, respectively. Using the two thresholds, any controller at time t belongs to only one of the three sets: UL, NL and OL, as in the Figure 6.5.

Figure 6.5: The three load categories at any time t.

In other words, a controller C_i belongs to a load set based on the following:

- $\theta_i \leq \overset{-}{\alpha}_{th_i} \implies C_i \in UL.$
- $\overset{-}{\alpha}_{th_i} < \theta_i < \overset{+}{\alpha}_{th_i} \implies C_i \in NL.$
- $\theta_i \geq \overset{+}{\alpha}_{th_i} \implies C_i \in OL.$

Note that these thresholds are adjustable to each controller based on its network policy [99]. Root-C, in return, updates Dist-Cs with the others' statuses in terms of what load set they have. Meaning, the distributed controllers see only the load set that others belong to, without going further into other details. As long as the mapping between the Dist-Cs and their load sets do not change, the Root-C refrains from updating. Thus, we mitigate the case of reporting unchanged statuses. See Algorithm 3 that describes our load informing strategy.

Algorithm 3: Adaptive Load Informing

Data: $L_{current}$: Current load set at t
L_{former}: former load set at $t-1$
Two thresholds: $\alpha^{-}_{th_i}$, $\alpha^{+}_{th_i}$ $\forall i \leq M$
Result: report = True or False: Report load status to C_j $\forall j \leq M,\ j \neq i$.

```
begin
    report=False
    if L_current(C_i) ≠ L_former(C_i) then
        report=True
        if L_current == OL then
            Start load balancing
        end
    else
        report=False
    end
    L_former = L_current
    Return report
end
```

6.5.3 Candidate Users Selection Module

A vertical handover decision is based on criteria that can be divided into the following [49]:

- Network context, such as link quality (RSSI, CIR, BER, SNR), coverage, bandwidth, latency, cost and security level.
- Terminal context, such as velocity, battery and location information.
- User context, such as user profile and preferences.
- Service context, such as service requirements and QoS.

A context-aware handover does not only consider the traditional ways, such as signal strength to trigger handover, but it also takes into consideration the knowledge of the mobile node and network's context information. Consequently, intelligent and improved handover decisions can be made [49]. However, combining a large number of criteria would

considerably affect the decision delay, due to the increased complexity of the decision algorithm [124].

In order to achieve our goal of relieving an overloaded controller, we exploit the fact that multiple coverage areas are overlapping and mobile users can switch to other networks based on a combination of criteria. We think that the involvement of users' preferences is essential and beneficial to both the user and network.

In the literature, several strategies are applied to include a user's preferences, such as Simple Additive Weighting (SAW), Multiplicative Exponent Weighting (MEW), Technique for Order Preference by Similarity to Ideal Solution (TOPSIS), Analytic Hierarchy Process (AHP) and Grey Relational Analysis (GRA) [49, 77]. Every strategy has its attributes and compliant situations. In our case, we look for a strategy that helps us evaluate different preferences inputted by a user and compare them to each other. On top of that, we want to get a rank to help prioritize those preferences, thus making a decision. In line with other studies, for example [111], we consider having a supporting module in the users' devices, allowing them to initially specify their preferences and then dynamically changing them as needed.

The AHP tool is considered a powerful decision-making tool originally proposed by Saaty [89]. Typically, the AHP tool defines a hierarchy of at least three levels, where the top level is the goal, the bottom level is the solution alternatives, and the middle level is the input to this tool (i.e., decision factors). These factors can be obtained from measurements such as weight, cost, etc., or from subjective opinions such as satisfaction and preference, which can be conflicting in some cases. Therefore, we think this tool in alliance with our direction of measuring users' perspectives.

AHP has the following four stages: decomposition, pair-wise comparison, weight calculation, and weight synthesis [89, 95]. By going through an example, we explain in detail the four stages.

- *Stage-1: Decomposition.* In this stage, a thorough understanding of the problem and then dividing it into subproblems are crucial. As a result, a hierarchy of the problem is obtained. In our problem, for the user's perspective, we can show the hierarchy as depicted in Figure 6.6. We see that the goal is to include the user's preferences. There are different factors that affect the user's decision; for example, a user's inclination, response time, and cost. In our scenario, we focus on edge users that are in the coverage area of two networks or more and based on that, we have a set of candidate networks.

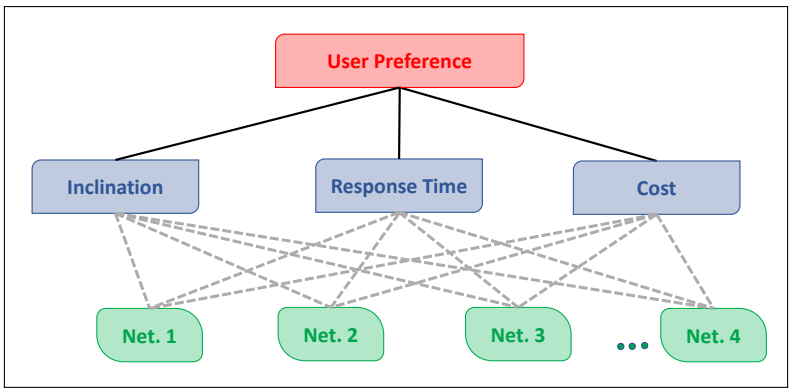


Figure 6.6: A taxonomy of a user's perspective using AHP method.

- *Stage-2: Pair-Wise Comparison.* In this stage, each user declares their subjective opinions regarding each pair of decision factors. For example, let us model the hierarchy shown in Figure 6.6 in a matrix form (Δ), and consider three decision factors: inclination as I, response time as R, and cost as C. Now, we draw a pair-wise comparison, to show how each factor is preferred over the other.

$$\Delta = \begin{array}{c} \\ I \\ R \\ C \end{array} \begin{array}{c} \begin{array}{ccc} I & R & C \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \end{array} \tag{6.14}$$

As shown by Eq. 6.14, we have a 3×3 matrix. The elements of this matrix are

determined by selecting values that show how the user is in favor of one factor over the other (i.e., pair-wise comparison). Note that the number of comparisons is a function of the number of decision factors (dn); thus, the total number of comparison is $\frac{dn(dn-1)}{2}$. There are different attempts to quantify such scale, yet the most famous one is the linear scale from 1 to 9, where each number describes the factor importance degree as in Figure 6.7.

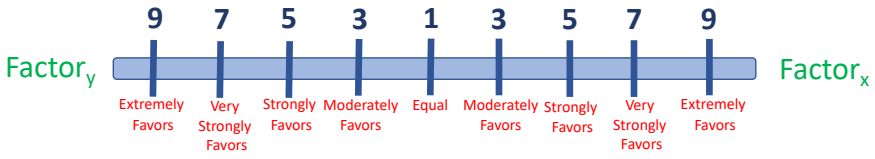


Figure 6.7: A preference scale in pair-wise comparison.

Based on the user's input, the pair-wise matrix in Eq. 6.14 is filled with either a value (1-9) or a reciprocal value, while the diagonal is always one. If the importance value is picked from the left side of 1, then we fill the matrix element with that value, otherwise, the matrix element is the reciprocal of that value. At this point, we end up filling the upper triangular matrix, to get the values of the lower triangular, we use the relation $a_{ij} = \frac{1}{a_{ji}}$. Carrying on with our example in Figure 6.6, we show a sample of a user's choices as the following matrix:

$$\Delta = \begin{array}{c} \\ I \\ R \\ C \end{array} \begin{array}{c} \begin{array}{ccc} I & R & C \end{array} \\ \begin{bmatrix} 1 & 1/7 & 1/5 \\ 7 & 1 & 3 \\ 5 & 1/3 & 1 \end{bmatrix} \end{array} \tag{6.15}$$

Apparently, to that user, the response time to their control traffic is the most critical factor; however, we show this result mathematically as we proceed further to the following stages.

- *Stage-3: Weight Calculation.* In this stage, we prioritize the factors by computing the priority vector. In order to do that, we find the normalized Eigenvector. First, for each column of matrix Δ, we sum its elements and then divide each by that sum, for instance, for the first column, we use the formula:

$$a_{i1} = \frac{a_{i1}}{\sum_{i=1}^{3} a_{i1}} \tag{6.16}$$

Back to our example, the resulting matrix is:

$$\underline{\Delta} = \begin{array}{c} \\ I \\ R \\ C \end{array} \begin{array}{c} \begin{array}{ccc} I & R & C \end{array} \\ \begin{bmatrix} 1/13 & 3/31 & 1/21 \\ 7/13 & 21/31 & 15/21 \\ 5/13 & 7/31 & 5/21 \end{bmatrix} \end{array} \tag{6.17}$$

Note that:

$$\frac{a_{ic}}{\sum_{i=1}^{3} a_{ic}} = 1, \quad \text{where } c = 1, 2, 3. \tag{6.18}$$

Next, we obtain the normalized Eigenvector by taking the average of each row according to:

$$v_{rj} = \frac{\sum_{j=1}^{3} a_{rj}}{3} \quad \text{where } r = 1, 2, 3. \tag{6.19}$$

Thus, we get the following Eigenvector V:

$$V = \frac{1}{3} \begin{bmatrix} 1/13 + 3/31 + 1/21 \\ 7/13 + 21/31 + 15/21 \\ 5/13 + 7/31 + 5/21 \end{bmatrix} = \begin{bmatrix} 0.0738 \\ 0.6434 \\ 0.2828 \end{bmatrix} = \begin{bmatrix} v_I \\ v_R \\ v_C \end{bmatrix} \tag{6.20}$$

Here, we obtained our priority vector, $V^T = [0.0738, 0.6434, 0.2828]$, where the sum of elements of our priority vector is 1.

As indicated by the priority vector of our example above, the response time factor

has the highest priority with 64.34%, while the cost factor is the second in ranking with 20.28%, and finally, the least important factor is the inclination factor with only 7.38%. At this point, we know the ranking of each decision factor. To find the relative weight, we simply obtain the ratio among every pair; for example, the cost factor is 3.83 (i.e., 28.28/7.38) times more important than the inclination factor. Thus, the relative weight matrix can be obtained as the following:

$$W = \begin{array}{c} \\ I \\ R \\ C \end{array} \begin{array}{c} \begin{array}{ccc} I & R & C \end{array} \\ \begin{bmatrix} v_I/v_I & v_I/v_R & v_I/v_C \\ v_R/v_I & v_R/v_R & v_R/v_C \\ v_C/v_I & v_C/v_R & v_C/v_C \end{bmatrix} \end{array} \tag{6.21}$$

- *Stage-4: Weight Synthesis.* Since the judgments are made by humans, they can be inconsistent. In this stage, we follow the steps proposed by Saaty in [89] to check the consistency of the user's input. Firstly, we find the maximum Eigenvalue. Given the Eigenvector as $V = [v_{11}, v_{21}, v_{31}]$, we obtain the maximum Eigenvalue (v_{max} as the sum of each column in Eq. 6.14 multiplied by each element of the Eigenvector as the following:

$$v_{max} = 13(0.0738) + \frac{31}{21}(0.6434) + \frac{21}{5}(0.2828) = 3.097 \approx dn \tag{6.22}$$

Then, Saaty in [89] proposed the use of a Consistency Index (CI) to measure the deviation of consistency using the following formula:

$$CI = \frac{v_{max} - dn}{dn - 1} \tag{6.23}$$

In our example $CI = 0.0485$. Then, we compare it to the Random Consistency Index ($RC(dn)$), whose values are given in Table 6.2.

Table 6.2: The random consistency index

dn	1	2	3	4	5	6	7	8	9	10
RC	0	0	0.58	0.9	1.12	1.24	1.32	1.41	1.45	1.49

Finding the Consistency Ratio (CR) of CI and RC estimates whether the inconsistency is acceptable or not.

$$\begin{cases} CR = \frac{CI}{RC} \leq 10\% & \text{Acceptable inconsistency.} \\ \text{Otherwise} & \text{Unacceptable inconsistency.} \end{cases}$$

Back to our example, $CR = 0.0836 < 10\%$, so the user's input is acceptable, and to be considered. Otherwise, the process is repeated with new pair-wise comparison matrix. When the inconsistency is unacceptable, the subjective judgment of a user needs to be revised. That can happen if the choices of that user do not satisfy the transitive property of ranking. For instance, according to the user's preferences, R is preferred over C ($R \succ C$), and C is preferred over I ($C \succ I$); logically, R should be preferred over P. However, the user makes P more preferred than R, which results in unacceptable inconsistency.

We then, consider the users that provide higher weight for response time, along with their locations, retrieved from the Global Positioning System (GPS), as the candidate users. Also, note that other users can be handover candidates if their AHP results indicate so.

6.5.4 Load Balancing Module

This module is local at Dist-C, where the information is gathered from the monitoring modules, user context module and other networks statuses. Upon the event of C_i being overloaded, this module is triggered in collaboration with Root-C. This module identifies the candidate edge users. Those users who preferred "response time (R)" as their main

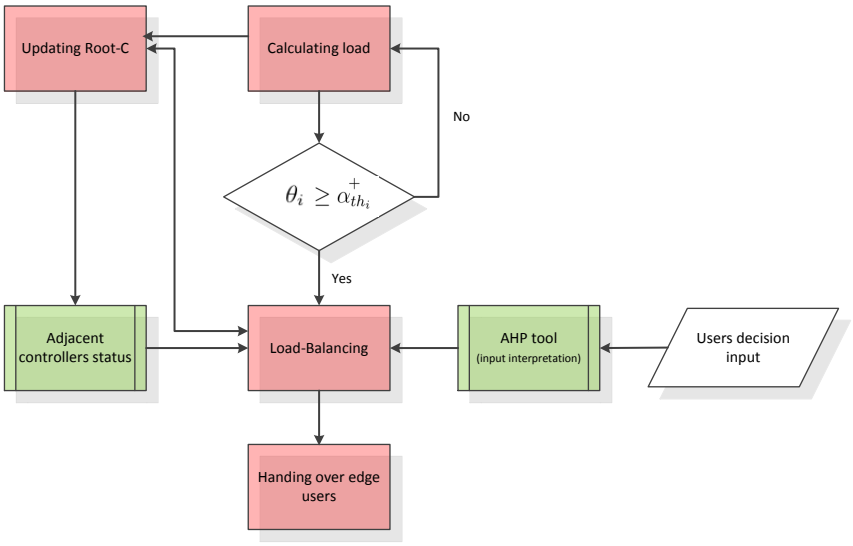


Figure 6.8: A flow chart of an over-loaded controller load balancing method.

metric, are candidates to be vertically handed over to other controllers with better response times without their involvement (i.e., network-controlled handover).

We know that there are four handover execution strategies: network-controlled handover, mobile-controlled handover, network-assisted handover, and mobile-assisted handover [124]. For our approach to work, the vertical handover is a network-controller that is initiated and mainly controlled by the controller as a resolution method for load balancing. The flow chart in Figure 6.8 summarizes the work-flow of our proposed framework from C_i's perspective.

Let C_i control a cellular domain, with the coexistence of multiple Wi-Fi controllers within its domain. C_i vertically hands over edge users $\in MN_e^i$ to candidate Wi-Fi networks that have their load set $\in NL \cup UL$. Given that Dist-Cs with load belonging to UL are preferred over those in NL. The selection method is presented in the following part.

6.5.5 Network Selection Module

This module is done by Root-C. Based on the system modeling presented in Section 6.4, now we formulate the network selection problem as a 0-1 integer optimization problem. Let $C_1 \in C$ be considered as overloaded at time t, then, C_1 triggers the load balancing module. First, it identifies edge users with R as their main criterion as U_h^1, where $U_h^1 \subseteq MN_e^1$ and the number of those candidates is $|U_h^1| = H$. Second, for each candidate user, there is a set of candidate networks that currently cover that user. Note that they can be obtained from Eq. 6.3 as the corresponding vector to each row (user). For a user u_h, let $\overrightarrow{P_h}$ denote the possible candidate networks vector, where $\overrightarrow{P_h} = \{p_h^2, p_h^3, ..., p_h^M\}$, and the elements of vector $\overrightarrow{P_h}$ are either 0 or 1, with each element corresponds to a controller. Third, the corresponding remaining capacity vector of candidate controllers is obtained from Eq. 6.7 as $\overrightarrow{R_h^m} = \{r_h^2, r_h^3, ..., r_h^M\}$. Now, based on the remaining capacities of candidate networks, Root-C selects the new network/controller for each edge user, u_h currently attached to the overloaded controller, C_1.

With the main objective of matching mobile users with better alternatives, taking into consideration their remaining capacities, we formulate our problem as a 0-1 integer programming problem, as follows:

$$\textbf{max} \quad \sum_{h=1}^{H} \sum_{m=1}^{M} (\, p_h^m . \, r_h^m \,) \tag{6.24}$$

subject to:

$$p_h^m \in \{0, 1\} \tag{6.25}$$

$$\theta_i < \overset{+}{\alpha}_{th} \tag{6.26}$$

$$\sum_{m=1}^{M} y_{xm} = 1 \quad \forall \, u_x \tag{6.27}$$

$$\sum_{b=1}^{k} z_{bj} \leq l_{s_j} \quad \forall \, s_j \in S \tag{6.28}$$

Our objective is vertically handing over as many users as possible, while maintaining their requirements in terms of response time. The value of p_h^m is either 1 or 0 to indicate whether the corresponding network is a candidate or not, respectively (Eq. 6.25). The load on the candidate networks have to be below their thresholds in order to accommodate more users and provide them with satisfied experiences (Eq. 6.26); meaning, only under-loaded and normally-loaded controllers can be candidates. Each user is connected to only one network (Eq. 6.27). When a user, u_h is switching to a new network, the new PoA has to have an available channel to be assigned to that u_h, otherwise, the vertical handover fails (Eq. 6.28). To solve this, a greedy search would give us a solution in a linear time with respect to the number of available networks and candidate users. In the worst-case scenario, the search would consume $O(HM)$, where H is the number of candidate edge users and M is the number of available networks. We argue that the computation overhead introduced by this step of our proposed-LB is negligible, as the number of candidate users and controllers are limited in practice.

6.6 Experiment and Evaluation

In this section, we describe our simulation environment, including the parameters and conditions. We also divide our simulations into four experiments, where we analyze the results and draw conclusions.

6.6.1 Setup

In our experiments, we chose Ryu [8] as our SDN controllers to communicate with the data plane, and to test our management framework. Ryu is an open source project hosted on GitHub and supported by the open Ryu community [7]. It is a component-based controller with well-defined APIs [58], that is written in Python and its codes is available under the

Apache 2.0 license [57]. The processing capacity of RYU is 6000 flows per second; this is the maximum requests rate that Ryu controller can handle in practice [42].

Table 6.3: Simulation parameters.

Parameter	**Setting**
Tool	Mininet 2.1.0
Links Delay between switches	2 ms
Links Delay between switches and hosts	5-10 ms
OpenFlow Message	v.1.3.0
Controllers	Ryu
Capacity α	6000 flows/s
Decay Factor β	0.2,1
Threshold α_{th}	900,5780 flows/s
Test Tools	iperf, ping, Tcpdump
UDP Datagram	1470 Byte
UDP Buffer Size	208 KByte
TCP Window Size	15 KByte

We implemented our data plane on Mininet v.2.1.0 [3].The data plane is composed of eight OVS-switches that support OpenFlow v1.3. We ran Ryu and Mininet on a virtual machine with intel core i7 processor and a 4GB of RAM. The virtual machine is running Ubuntu 16.04 LTS.

In our setup, we implemented a two-tier hierarchical control plane; we had two controllers as tier-1 Dist-Cs (C_1, C_2), and a third controller connected to each of them as the Root-C. We used the TCP socket as our messaging system to get messages moving between the Ryu controllers. C_1 represented the controller of a cellular network with six OVS-switches and 20 users, while C_2 represented a Wi-Fi network with a smaller capacity and connected to two OVS-switches and four attached users. The users randomly generated different types of traffic, including UDP and TCP. The list of simulation parameters is shown in Table 6.3.

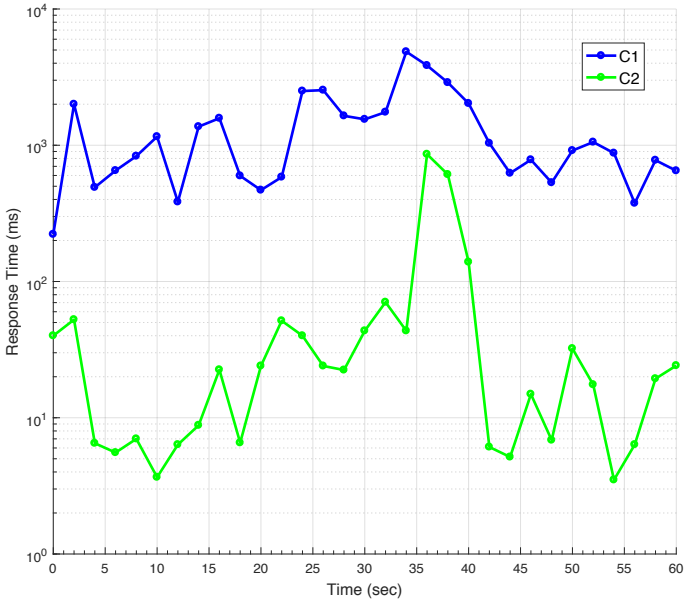


Figure 6.9: Recorded users' perceived response time.

6.6.2 Experiment 1

In the first set of experiments, we run the experiment ten times and we recorded the average perceived response time by the connected users every two seconds, as the load varied on their managing controllers C_1 and C_2, refer to Figure 6.9. We simulated C_1 as a cellular network controller; thus, it handled more users and more load. As the load on C_1 reached its threshold at around the 30^{th} second, our proposed-LB started by handing over candidate users to C_2. Afterwards, the handed-over users' traffic was handled by C_2. We noticed a drop in perceived response time by 20%, as the load decreased by 14% of the connected users and their on-going traffic. Meanwhile, users connected to C_2 experienced an increase in C_2's response time, which was still acceptable since C_2 was under-utilized and it resolved that increase rapidly.

6.6.3 Experiment 2

In the previous experiment, we recorded the impact of our proposed-LB on users within their domains. Now, we test the impact on a candidate user, let us assume it is u_x, in

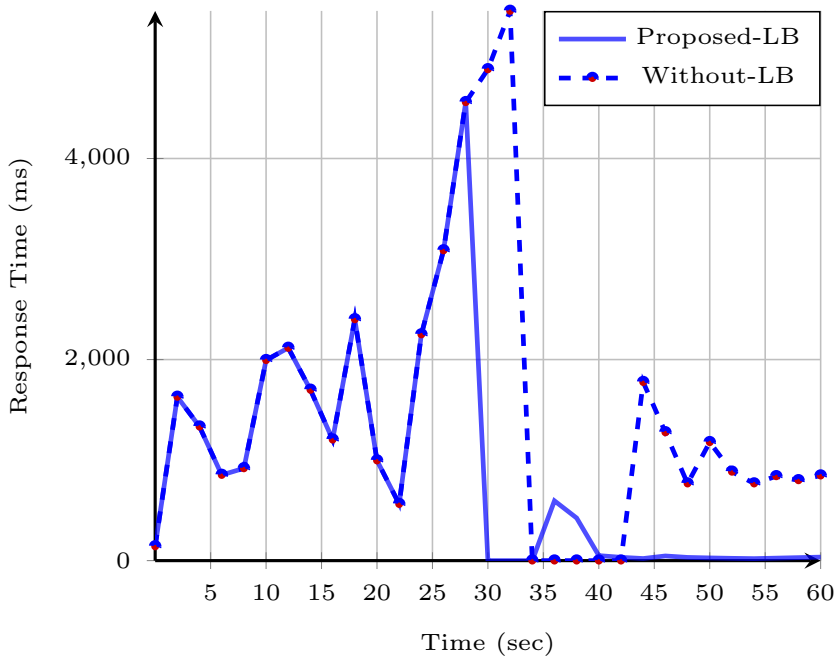


Figure 6.10: The perceived response time by a mobile users in two scenarios.

two scenarios. In one scenario, we recorded the impact of our proposed-LB in terms of the perceived response time, every two seconds, including the effect of the vertical handover. In the second scenario, we studied what would happen if that user stayed connected to an over-loaded controller, without a countermeasure. As Figure 6.10 shows, as the managing controller approached its threshold at around 28^{th} second, it took longer to respond to u_x. Then, since u_x was a candidate user, it got vertically handed over to another available under-utilized controller. Thus, u_x experienced around four seconds of disconnection, and then restored connection with very low response time from the new controller, compared to the previous case, which is around an 80% decrease in the perceived response time. For a user with response time as their main concern, that shift is favorable. However, if u_x does not get handed over, the response time would reach unacceptable levels (the dashed line), and then the managing controller would start dropping requests causing disruption for longer periods, until the over-loading state is resolved.

6.6.4 Experiment 3

In this experiment, we compared our proposed-LB against other approaches, [55] and [107]. We chose these two approaches because they share a common ground with our approach, the static assignment of controller and switches. The authors of Onix [55], proposed a flat static structure to scale out their topology. We used their approach as a benchmark of the static mapping, so we refer to it as SM. The other approach proposed by Wang et al. in [107] is mainly dependent on the idea of balancing the load of multiple controllers based on shifting *macroflows* from the management of one controller to another. A *macroflow* is a set of flows originating from a particular switch and destined to the same switch. On the other hand, with *microflows*, the controller installs rules in the most granular level for each connection. Fine granularity provides means to adapt to network changes and helps realize the SDN benefits; however, it can cause heavy load on switches and controllers. Even though *a* macroflow can relieve the controller from handling each flow, it requires the use of *wildcards* that rely on TCAM which is expensive compared to SRAM.

The proposed approach by [107] is implemented in a hierarchical architecture, where a set of distributed controllers report their load information and flow entries stats to a root controller. Then, the root controller determines the *macroflow(s)* to be redirected to ensure fairness among all the controllers.

As shown in Figure 6.11, we compared the controller load between three approaches: SM [55], GFRD [107], and our proposed-LB. Expectedly, a controller's load in SM increases linearly as the requests rate per second increases. As depicted, GFRD is able to minimize the load by 50%, and that is due to the pre-installation of wildcard rules in adjacent edge switches. Our approach, however, increases linearly as in SM except that once the requests reach a pre-defined threshold (i.e., $\alpha^{+}_{th} = 0.75 \times \alpha$), the balancing module is triggered to hand over candidate users. Hence, the controller is relieved as candidate users' traffic is shifted to other controllers. In this experiment, we chose $\beta = 0.2$ to make our results

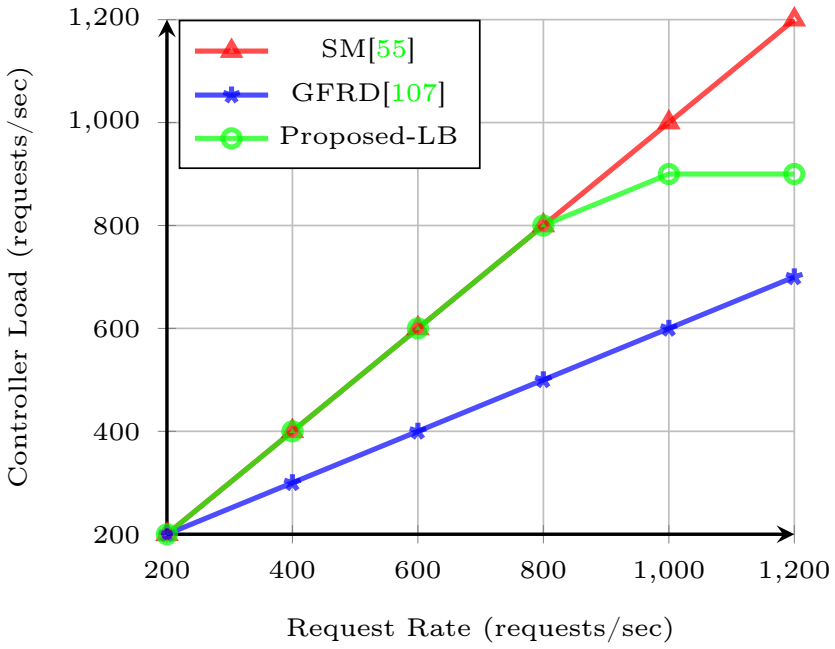


Figure 6.11: Controller load as request rate increases in three approaches.

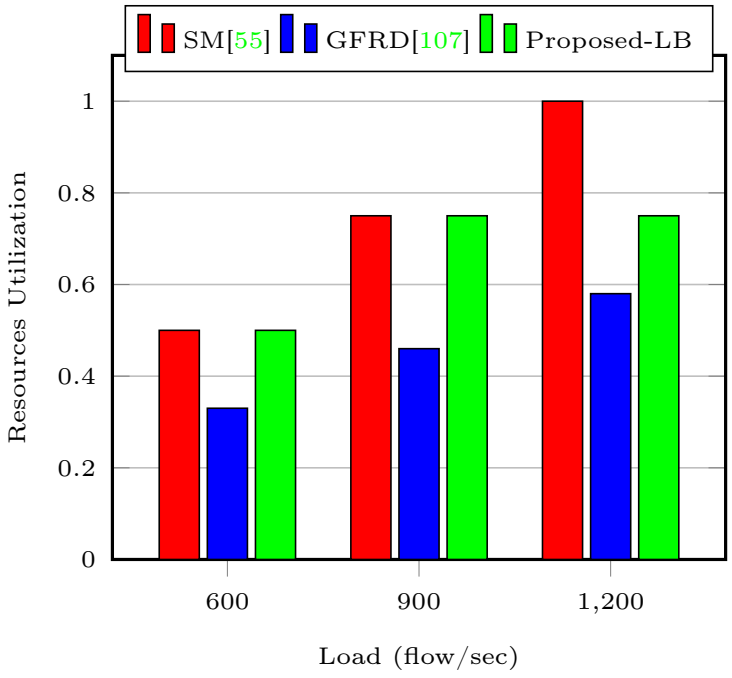
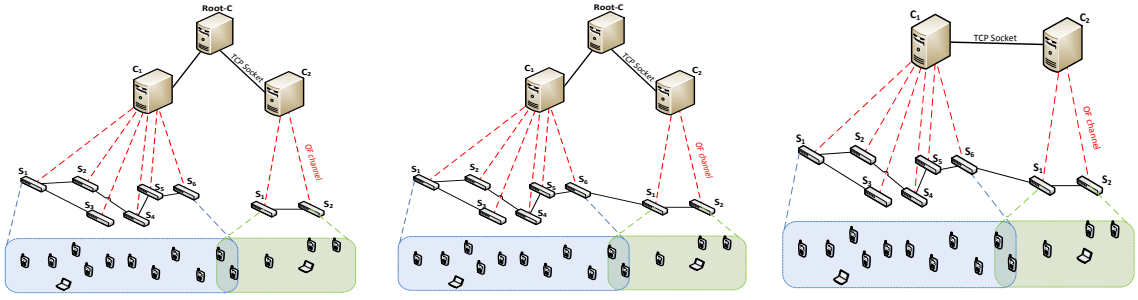


Figure 6.12: Resource utilization at different loads.

comparable to the results reported by [107], making the controller capacity 1200 request/s. Based on the results shown in Figure 6.11, we estimated the resources utilization under different loads for the three approaches in Figure 6.12. Knowing that resource utilization is computed as: $util = \frac{\theta}{\alpha}$. Apparently, [107] reports 50% lower controller load compared to SM and that is because of the pre-installation of *macroflows* wildcard rule for traffic redirection. This rule sends out those flows as chunks from one controller to another through switches on the path, the thing that shows reduction in the load from the beginning even before load balancing is triggered. In their approach, their target is to ensure fairness between controllers as well as reducing the response time of the whole system; unlike our target, which aims to relieving the over-loaded cellular controller.

We simulated the three aforementioned approaches on the topologies that are shown in Figure 6.13 to measure their reactiveness and effectiveness. We triggered traffic randomly from users for 120 seconds, and recorded the response times that the users perceived of their controllers. The simulations were repeated ten times and the average response times were taken. We plotted the results of our approach against the other two approaches in Figure 6.14.

From Figure 6.14, we observe the following. There were two periods of over-loading, one started around the 28^{th} second, while the other started at the 92^{nd} second; the three approaches handled these periods differently. GFRD did not show drastic changes as



(a) Proposed-LB topology. **(b)** [107] topology. **(c)** [55] topology.

Figure 6.13: Simulation topologies.

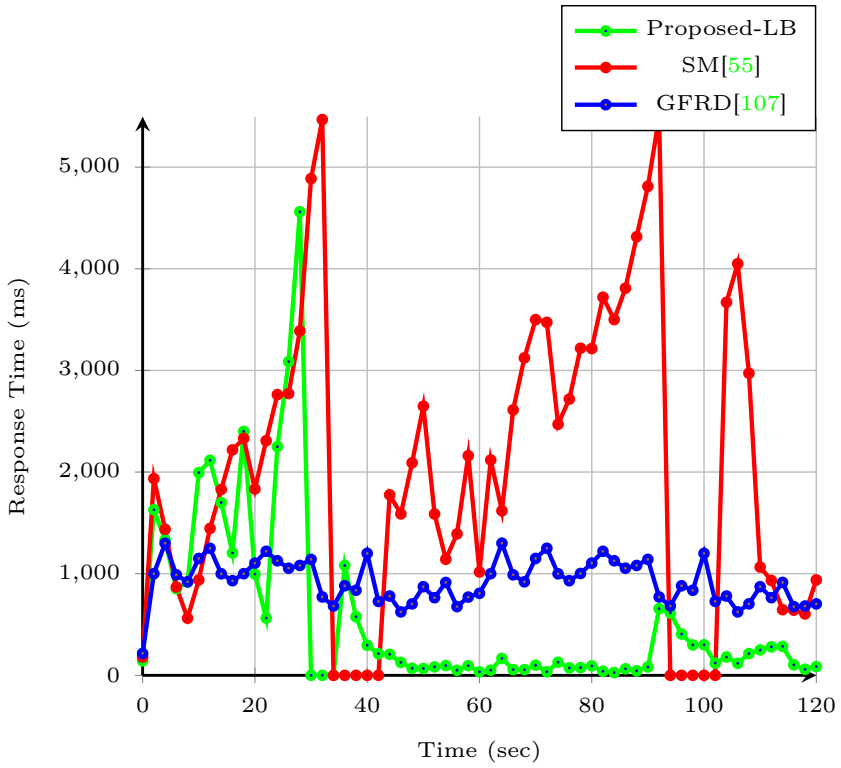


Figure 6.14: The perceived response time by users in three approaches.

this approach based on a proactive strategy that ensures no over-utilization may happen. Regarding SM, there was no reactive mechanism to over-loading states; therefore, the controller load reached its cap and started dropping flows as it could not handle more traffic. As for our proposed-LB, once the controller load reached a pre-defined threshold (i.e., $\alpha_{th} = 5780$ flows/s), 10% of the connected users and their on-going traffic got handed over to an under-utilized controller. Those users helped in relieving the over-loaded controller resources enhancing the response time for other connected users and gaining better response time from their new controller. Yet, they experienced some data loss due to the vertical hand over that was around 8%, while in the SM, the data loss was at least 15% until the overloading state got resolved. In this part, we refer to an essential metric called "balance time" [27].

Definition 4 (Balance Time)**.** *The period of time starting from when the unbalance state is detected until that state is resolved.*

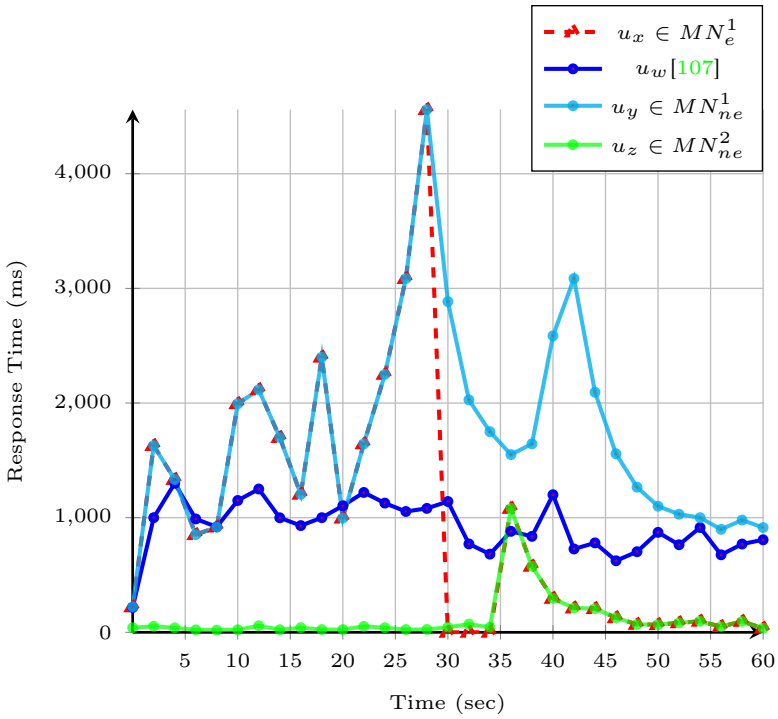


Figure 6.15: The perceived response time by different users in two approaches.

For instance, the balance time in our approach starts from reaching a threshold in term of load, then identifying the candidate users, selecting more suitable network(s), and then vertically handing over the users. Intuitively, the lower the balance time, the more efficient the load balancing strategy. In our simulation, the balance time was the period between $[28 - 36]$ seconds.

To further analyze and measure the effect of our approach, we need to address the impact on different types of users, including those that remain connected to the over-loaded controller and the ones that are connected to the under-loaded controller before, during and after the balance time. Therefore, we recorded the response time of four categories of users. Since both periods of over-loading showed the same trend, we focus on the first 60 seconds that contained the first over-loading period only as in Figure 6.15. Here, u_y is a non-edge user connected to C_1, u_z is a non-edge user connected to C_2, u_w is a user in GFRD, and u_x is an edge user to be vertically handed over when the load balancing mechanism is triggered. Note that C_1 got over-loaded, while C_2 was under-loaded at the

time of network selection. The user connected to C_1, $u_y \in MN^1_{ne}$, experienced a response time around 4.5 sec at the 28^{th} second, then about 10% of users got handed over resulting in 36% drop in the perceived response time from C_1. Meanwhile, as the load on C_2 was increased by the handed-over users, C_2's response time increased at the 36^{th} second for a short period of time, which is still acceptable given that C_2 was able to accommodate more users and provided less delay. Regarding the user perceived response time in GFRD, we can see that they experienced a nearly-steady pattern of response time due to pre-configuration of wildcard rules that maintain the whole system in a fair state among controllers. Unlike the candidate edge users connected to C_1, $u_x \in MN^1_e$, after the balance time, as they got connected to another controller C_2, they experienced in average 28% drop in the response time compared to GFRD, during the period between $[36 - 60]$ seconds.

In this part, we discuss the three aforementioned approaches in detail. The SM method proposed in [55], is based on the idea of scaling out a network but without consideration of over-loading cases. Therefore, it did not perform well in the simulation in terms of handling the heavy traffic. Regarding the method in [107], the algorithm mainly is done at a root controller, introducing high computation overhead as the network size grows or traffic arrival increases. The majority of computation has to be done on a single controller, which still suffers SPoF or a choke point; there is no functionality distribution since the root controller has to run the algorithm and the distributed controllers only report stats. This approach also suffers from the restriction imposed by TCAM switches on the number of wildcard rules that can be installed.

Based on multiple criteria, we evaluate our approach and the other two, GFRD and SM, in Table 6.4. Note that we use (—) to indicate that this property has not been investigated, and the (✖) to indicate unsatisfied property by that method. An important property that we want to shed light on is the fact that the methods followed by [55] and [107] require inter-connection between domains for their algorithms to work; meaning that the concept of completely isolated domains are not taken into consideration. Unlike our approach,

Table 6.4: Evaluation of the three approaches.

Property	SM[55]	GFRD[107]	Proposed-LB
Scalability	✔	✔	✔
Static switch-controller assignment	✔	✔	✔
Minimized synchronization	✖	—	✔
Functionality distribution	✔	✖	✔
Mitigating bottlenecks	✔	✖	✖
Isolated domains	✖	✖	✔
Low response time	✖	✔	✔
Efficient resource utilization	✔	✔	✔
Hardware restrictions	—	✔	✔
Data loss percentage	15%	—	8%
Mobility consideration	✖	✖	✔
User preference inclusion	✖	✖	✔
Fairness	✖	✔	✖
Considering non-uniform capacities	✖	✖	✔

we provide a solution for both connected and disconnected domains, since our proposal is irrespective to infrastructure. Still, the SPoF issue persists; however, not as severe as placing the control logic of a network entirely on a single controller.

6.6.5 Experiment 4

On another note, in our approach, we aim at reducing the number of status synchronization messages between tier-1 controllers in order not to introduce intercommunication overhead. In this part, we compare the complexity in terms of the number of messages, in the worst case scenario, between [27], DALB [126] and our strategy. As reflected in Figure 6.16, the approach in [27] has the highest synchronization messages among the three. As for the DALB, it performs well when there are fewer controllers interconnected; however, as the number of controllers grows, the overhead increases. Our strategy performs well regardless of the number of interconnected controllers. As the number grows, our proposed strategy outperforms the other two approaches, since Root-C limits the frequency of updating when a controller does not change its load set.

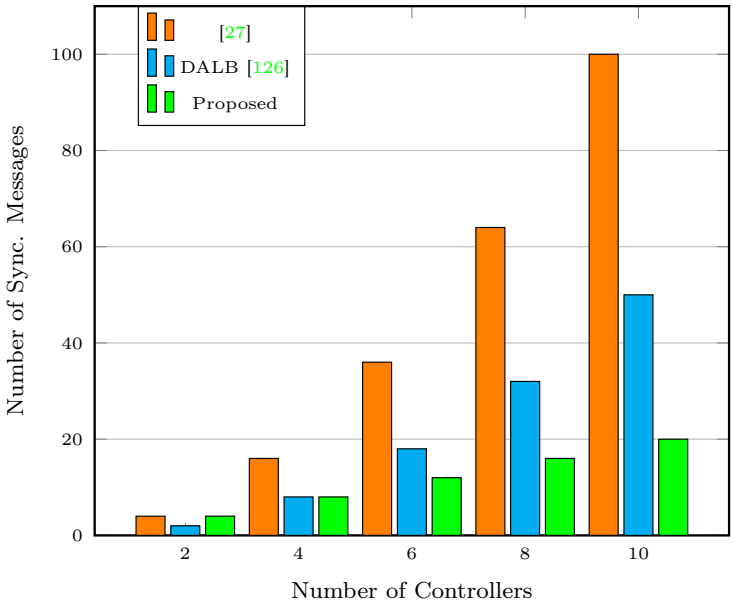


Figure 6.16: Synchronization complexity in term of number of messages.

6.7 Discussion

A plethora of wireless access networks overlap, especially in urban areas. The "Openness" advocated by [120] is appealing considering wireless scarcity and limited SDN controller resources. Therefore, we suggest that cellular providers deploy Wi-Fi hotspots strategically to apply our approach. This can be a merit, or a bonus service that would benefit the users and raise the competition among service providers. Those hotspots can be owned by the service providers or rented from a third-party to act as complementary networks.

There are challenging issues though. The authors of [19] reported some of the challenges that would arise when integrating 3G and Wi-Fi technologies. First, an efficient authentication mechanism, e.g., contacting Authentication Authorization and Accounting (AAA) server, have to be implemented to allow a Wi-Fi provider to authenticate switching users. Second, on-going sessions should be maintained seamlessly. Third, the switching should happen automatically without the user's intervention; meaning that the handover is better when network initiated. Fourth, the contract policies, i.e. QoS, guarantee that the user has with the cellular service provider should be maintained after switching.

From the under-loaded or normally-loaded controllers' owners perspective, usually Wi-Fi providers, they are willing to accommodate edge users handed-over by over-loaded controllers. Business-wise, they benefit from bringing in more customers to their hotspot (coffee shops, campuses, retailers), which is called "value-added resource" [1]. So, it is a win-win situation, where overloaded controllers get to relieve their resources and under-loaded controllers to increase their benefits. However, there are limited number of users that can be accommodated based on the available channels, mobility pattern and velocity [100].

6.8 Summary

To summarize, multiple controllers can efficiently improve the scalability and reliability aspects. However, unavoidable issues may arise, including load imbalance and synchronization overhead. In our approach, a sense of coalition has been established between controllers of different domains, as they initially appoint a Root-C to coordinate management tasks among them. Yet, more importantly, they collaborate to minimize the maximum response time by exploiting the concept of context-aware vertical handover. In this chapter, we have proposed exploiting the dynamism of mobile users to improve their experience by minimizing the response time of their managing controllers, and thus, minimizing delay. Over-loaded controllers are relieved by vertically handing-over users, with higher preferences for response time. Our experiments have showed improvements in the completion of mobility-related procedures, as the response time of controllers decreased. Our proposed heuristic approach targets other delay causes, such as status synchronization and network selection.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The Software-Defined Networking paradigm has stepped in to tackle several issues and challenges in conventional networking, especially in mobile/cellular networks. Therefore, in this thesis, we focused on the impact of the SDN paradigm on the handover procedure and analyzed the main causes of delay. After thorough research, we identified the processing time of handover signaling messages as one of the main causes. Consequently, we studied the contributing factors to that type of delay. Our objective then turned to find solutions to deal with each factor. First, we targeted the handling procedures of handover signaling messages within the network's devices. We provided a mathematical model using queueing theory to model two OpenFlow handover-related messages to help us design effective solutions. Accordingly, we proposed a solution based on offloading handover handling to dedicated entities that are separate from the main controller and switches. Our solution has been validated and evaluated in simulations and applied to an LTE system. We performed a comparative analysis of the architecture of a centralized EPC controller and our proposed functionality-distributed EPC controller architecture in the LTE setting. We then studied two main metrics, handover latency and average throughput per user.

Simulations have shown that under the same network conditions, the proposed EPC controller architecture has better performance in both metrics compared to the centralized approach. Therefore, we argue that the distributed mobility management approach does benefit handover handling in one domain.

As for multi-region networks, we proposed a hierarchical architecture that provides better scalability over the centralized architecture and less delay compared to the distributed approach. We targeted a second contributing factor to the processing delay, the number of switches to-be-configured upon mobility. Note that the fewer the number of configured switches, the faster the re-routing of traffic. In this setting, we studied MIP protocol, and then followed a heuristic approach to solve the binding-cache placement problem. Through simulations, our approach has shown improvement in metrics like UDP/TCP throughput, packet loss, and jitter.

Since the signaling messages have to be handled by a controller, that controller's response time has a vital impact on the completion of every user's procedures, especially handover. Evidently, a controller's load has a direct relationship with its response time; hence, the bigger the load, the longer the response time and the worse the users' perception of services . Therefore, we approach the over-loading controller problem, specifically to minimize the maximum response time. We mainly tackled the following scenario. Given a cellular network of a controller managing network devices and mobile users, and multiple nested other wireless networks where each has a managing controller, how do we relieve the cellular controller by exploiting users' preferences and the complementary resources of other cooperative controllers? Here, our main contribution is a novel approach that mainly aims at vertically handing over some edge users, considering some context information regarding the users and controllers. As a result, a controller's response time to any mobility-related procedure decreases. We proposed a management framework that includes three main aspects. We identified candidate users based on their context information; we used AHP, a decision-making tool to include user's input into our framework. We proposed

a novel mechanism that reduces the frequency of load disseminating between multiple controllers, and hence, reducing processing and communication overhead. Once the candidate users are determined, we optimized the decision problem on the selection among several candidate networks. We compared our mechanism against two other static ones, and in our approach, the vertically handed-over users experienced lower response time compared to other approaches.

As a final remark, throughout this thesis, we have adopted OpenFlow as an enabler to study the effects of our proposed algorithms and approaches. However, OpenFlow can be replaced by any other southbound protocol. Other protocols can easily be incorporated according to their designs, yet our algorithms still can be implemented.

7.2 Future Work

In this section, we discuss some possible research directions to extend our work.

So far, in our proposed-LB, we assume the vertical handover happens to users and their on-going traffic with a guarantee of an available channel only, yet, there was no guarantee of minimum channel bandwidth in respect to their service requirements. An interesting research direction would be coupling the control plane response time with the service requirement at the data plane level in our network selection approach. In this case, we would extend our vertical handover decision process to include the service type as a mobility parameter to ensure the channel capacity of the new attachment guarantees the bandwidth requirement of the handed-over user QoS.

An interesting idea that is worth further investigation is the role a mobility prediction scheme could have on our proposed-LB. For instance, we can model pedestrians in urban areas. The prediction can be estimated as part of the monitoring phase, where we assume that each user is equipped with a GPS service that a managing controller uses

to collect location information (i.e., including coordinates, velocity, and direction). The prediction could have a positive impact on the network selection part of our proposed-LB since resources will be reserved for the upcoming users. Another interesting idea that could be beneficial to our approach is clustering the handed-over users and their service requirements.

As part of our over-loading mitigation solution in Chapter 6, we only consider the case to incorporate different technologies to work as complementary resources. Yet, horizontal handovers can be implemented as part of the solution. In our system, we assume having one controller to manage all PoAs in a cellular network. With a large number of PoAs, we can assume having multiple controllers and thus incorporating horizontal handover as another option to relieve an overloaded controller. It can be one of the solutions available for cellular service providers.

Another solution that could benefit a cellular service provider is owning and deploying Wi-Fi cells strategically within its coverage zone to handle high demands and keep the load manageable at the control level. That would open another research direction to optimize the number of supporting Wi-Fi controllers and their capacities.

A number of questions regarding integrating heterogeneous wireless technologies and the concept of "openness" remain to be addressed. Looking forward, further attempts to overcome vertical handover delay, matching QoS requirements, and service provisioning issues among different operators/service providers could be quite beneficial to the future of wireless networks and mobile users.

References

[1] Everybody wins with wifi. http://www.devicescape.com/2012/07/24/everybody-wins-with-wifi/.

[2] Floodlight sdn controller. http://www.projectfloodlight.org/floodlight.

[3] Mininet: An instant virtual network on your laptop (or other pc). http://mininet.org.

[4] Ns-3 tutorial. https://www.nsnam.org/docs/release/3.17/tutorial/singlehtml/index.html.

[5] Onos sdn controller. https://https://onosproject.org.

[6] Pox controller. http://www.noxrepo.org/pox/about-pox/.

[7] Ryu official site. http://osrg.github.io/ryu.

[8] Ryu sdn framework. https://osrg.github.io/ryu-book/en/ryubook.pdf.

[9] Tcpdump. https://www.tcpdump.org.

[10] Wireshark. https://www.wireshark.org.

[11] Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021. Technical report, 2017.

[12] I. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1–30, 2014.

[13] I. Akyildiz, J. Xie, and S. Mohanty. A survey of mobility management in next-generation all-ip-based wireless systems. *IEEE Wireless communications*, 11(4):16–28, 2004.

[14] H. Ali-Ahmad, C. Cicconetti, A. De la Oliva, V. Mancuso, M. Sama, P. Seite, and S. Shanmugalingam. An sdn-based network architecture for extremely dense wireless networks. In *Proceedings of the IEEE SDN for Future Networks and Services*, 2013.

[15] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, et al. An analytical model for software defined networking: A network calculus-based approach. In *Proceedings of the IEEE Global Communications Conference*, 2013.

[16] M. Bari, A. Roy, S. Chowdhury, Q. Zhang, M. Zhani, R. Ahmed, and R. Boutaba. Dynamic controller provisioning in software defined networks. In *Proceedings of the 9th International Conference on Network and Service Management*, 2013.

[17] G. Bloch, S. Greiner, H. de Meer, and K. Trivedi. *Queueing networks and Markov chains: Modeling and performance evaluation with computer science applications.* John Wiley & Sons, 2006.

[18] W. Braun and M. Menth. Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet*, 6(2):302–336, 2014.

[19] M. Buddhikot, G. Chandranmenon, S. Han, Y. Lee, S. Miller, and L. Salgarelli. Integration of 802.11 and third-generation wireless data networks. In *Proceedings of the 22nd IEEE International Conference on Computer Communications*, 2003.

[20] Z. Cai, A. Cox, and T.Ng. Maestro: A system for scalable openflow control. Technical report, 2010.

[21] T. Chen, M. Matinmikko, X. Chen, X. Zhou, and P. Ahokangas. Software defined mobile networks: Concept, survey, and research directions. *IEEE Communications Magazine*, 53(11):126–133, 2015.

[22] G. Cheng, H. Chen, Z. Wang, and S. Chen. Dha: Distributed decisions on the switch migration toward a scalable sdn control plane. In *Proceedings of the 14th IEEE IFIP Networking Conference*, 2015.

[23] E. Chlebus and W. Ludwin. Is handoff traffic really poissonian? In *Proceedings of the 4th IEEE International Universal Personal Communications Conference*, 1995.

[24] S. Chourasia and K. Sivalingam. Sdn-based evolved packet core architecture for efficient user mobility support. In *Proceedings of the 1st IEEE Network Softwarization Conference*, 2015.

[25] Cisco Blogs. https://blogs.cisco.com/news/aci-anywhere.

[26] J. Costa-Requena. Sdn integration in lte mobile backhaul networks. In *Proceedings of the 28th IEEE International Conference on Information Networking*, 2014.

[27] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu. A load-balancing mechanism for distributed sdn control plane using response time. *IEEE Transactions on Network and Service Management*, 15(4):1197–1206, 2018.

[28] J-P. Delahaye, J. Palicot, and P. Leray. A hierarchical modeling approach in software defined radio system design. In *Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation*, 2005.

[29] A. Dixit, F. Hao, S. Mukherjee, TV. Lakshman, and R. Kompella. Towards an elastic distributed sdn controller. *ACM SIGCOMM Computer Communication Review*, 43(4):7–12, 2013.

[30] X. Duan, A. Akhtar, and X. Wang. Software-defined networking-based resource management: Data offloading with load balancing in 5g hetnet. *EURASIP Journal on Wireless Communications and Networking*, 2015(1):181, 2015.

[31] X. Duan, X. Wang, and A. Akhtar. Partial mobile data offloading with load balancing in heterogeneous cellular networks using software-defined networking. In *Proceedings of the 25th IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communication*, 2014.

[32] R. Enns and and A. Bierman M. Bjorklund, and J.Schoenwaelder. Network configuration protocol. Technical report, 2011.

[33] H. Farhady, H. Lee, and A. Nakao. Software-defined networking: A survey. *Computer Networks*, 81:79–95, 2015.

[34] N. Feamster, J. Rexford, and E. Zegura. The road to sdn: An intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[35] F. Giust, L. Cominardi, and C Bernardos. Distributed mobility management for future 5g networks: Overview and analysis of existing approaches. *IEEE Communications Magazine*, 53(1):142–149, 2015.

[36] B. Görkemli, S. Tatlıcıoğlu, A. Tekalp, S. Civanlar, and E. Lokman. Dynamic control plane for sdn at scale. *IEEE Journal on Selected Areas in Communications*, 36(12):2688–2701, 2018.

[37] G. Hampel, M. Steiner, and T. Bu. Applying software-defined networking to the telecom domain. In *Proceedings of The 32nd IEEE International Conference on Computer Communications*, 2013.

[38] M. Hata, S. Izumi, T. Abe, and T. Suganuma. A proposal of sdn based mobility management in multiple domain networks. In *Proceedings of the Network Operations and Management Symposium*, 2016.

[39] B. Heller, R. Sherwood, and N. McKeown. The controller placement problem. In *Proceedings of the ACM 1st workshop on Hot topics in software defined networks*, 2012.

[40] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia. Pareto-optimal resilient controller placement in sdn-based core networks. In *Proceedings of the 25th IEEE International Teletraffic Congress*, 2013.

[41] K. Hong, S. Lee, and M. Shin. Mobility management in wlan-based virtualized networks. *Wireless Personal Communications*, 72(1):581–596, 2013.

[42] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan. Multi-controller based software-defined networking: A survey. *IEEE Access*, 6:15980–15996, 2018.

[43] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng. Balanceflow: Controller load balancing for openflow networks. In *Proceedings of the 2nd IEEE International Conference on Cloud Computing and Intelligence Systems*, 2012.

[44] N. Jagadeesan and B. Krishnamachari. Software-defined networking paradigms in wireless networks: A survey. *ACM Computing Surveys (CSUR)*, 47(2):27, 2015.

[45] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, et al. Modeling and performance evaluation of an openflow architecture. In *Proceedings of the 23rd International Teletraffic Congress*, 2011.

[46] S. Jeon, C. Guimarães, and R. Aguiar. Sdn-based mobile networking for cellular operators. In *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*, 2014.

[47] M. Karimzadeh, A. Sperotto, and A. Pras. Software defined networking to improve mobility management performance. In *Proceedings of the IFIP International Conference on Autonomous Infrastructure, Management and Security*, 2014.

[48] M. Karimzadeh, L. Valtulina, and G. Karagiannis. Applying sdn/openflow in virtualized lte to support distributed mobility management. In *Proceedings of the 4th International Conference on Cloud Computing and Services Science*, 2014.

[49] M. Kassar, B. Kervella, , and G. Pujolle. An overview of vertical handover decision strategies in heterogeneous wireless networks. *Computer Communications*, 31(10):2607–2620, 2008.

[50] P. Kaur, J. Chahal, and A. Bhandari. Load balancing in software defined networking: A review. *Asian Journal of Computer Science and Technology*, 7(2):1–5, 2018.

[51] D. Kendall. Some problems in the theory of queues. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 151–185, 1951.

[52] A. Khiat, J. Bakkoury, M. El Khaili, and A. Bahnasse. Study and evaluation of vertical and horizontal handover's scalability using opnet modeler. *International Journal of Computer Science and Information Security*, 14(11):807, 2016.

[53] Y. Kirsal and O. Gemikonakli. Performability modelling of handoff in wireless cellular networks with channel failures and recovery. In *Proceedings of the 11th IEEE International Computer Modelling and Simulation Conference*, 2009.

[54] C. Kolias, S. Ahlawat, C. Ashton, et al. Openflow-enabled mobile and wireless networks. *White Paper*, 2013.

[55] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, 2010.

[56] D. Kreutz, F. Ramos, P. Verissimo, C. Rothenberg, et al. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[57] A. Krishnamurthy, S. Chandrabose, and A. Gember-Jacobson. Pratyaastha: An efficient elastic distributed sdn control plane. In *Proceedings of the 3rd ACM Workshop on Hot Topics in Software Defined Networking*, 2014.

[58] R. Kubo, T. Fujita, Y. Agawa, and H. Suzuki. Ryu sdn framework: Open-source sdn platform software. *NTT Technical Review*, 12(8):1–5, 2014.

[59] S. Kukliński, Y. Li, and K. Dinh. Handover management in sdn-based mobile networks. In *Proceedings of IEEE Global Communications Conference Workshops*, 2014.

[60] W. Lan, F. Li, X. Liu, and Y. Qiu. A dynamic load balancing mechanism for distributed controllers in software-defined networking. In *Proceedings of the 10th International Conference on Measuring Technology and Mechatronics Automation*, 2018.

[61] S. Lee, K. Sriram, K. Kim, Y. Kim, and N. Golmie. Vertical handoff decision algorithms for providing optimized performance in heterogeneous wireless networks. *IEEE Transactions on Vehicular Technology*, 58(2):865–881, 2009.

[62] L. Li, Z. Mao, and J. Rexford. Toward software-defined cellular networks. In *Proceedings of IEEE European Workshop on Software Defined Networking*, 2012.

[63] J. Little. A proof of the queueing formula: L=λw. *Operations Research*, 9(3):383–387, 1961.

[64] H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter. Traffic engineering with forward fault correction. In *Proceedings of the ACM SIGCOMM Computer Communication Review*, 2014.

[65] S. Liu and B. Li. On scaling software-defined networking in wide-area networks. *Tsinghua Science and Technology*, 20(3):221–232, 2015.

[66] M. Liyanage, A. Gurtov, and M. Ylianttila. *Software defined mobile networks (SDMN): Beyond LTE network architecture*. John Wiley & Sons, 2015.

[67] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarchel. Modelling of openflow-based software-defined networks: The multiple node case. *IET Networks*, 4(5):278–284, 2015.

[68] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarschel. On the modeling of openflow-based sdns: The single node case. *aArXiv preprint ArXiv:1411.4733*, 2014.

[69] T. Mahmoodi and S. Seetharaman. Traffic jam: Handling the increasing volume of mobile data traffic. *IEEE Vehicular Technology Magazine*, 9(3):56–62, 2014.

[70] C. Marquezan, X. An, Z. Despotovic, R. Khalili, and A. Hecker. Identifying latency factors in sdn-based mobile core networks. In *Proceedings of IEEE Symposium on Computers and Communication*, 2016.

[71] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, et al. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[72] J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. Curtis, and S. Banerjee. Devoflow: Cost-effective flow management for high performance enterprise networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[73] M. Moradi, W. Wu, L. Li, and Z. Mao. Softmow: Recursive and reconfigurable cellular wan architecture. In *Proceedings of the 10th ACM International Conference on emerging Networking Experiments and Technologies*, 2014.

[74] G. Mulec, R. Vasiu, and F. Frigura-Iliasa. Distributed flow controller for mobile ad-hoc networks. In *Proceedings of the 8th IEEE International Symposium on Applied Computational Intelligence and Informatics*, 2013.

[75] R. Muntz. *Poisson departure processes and queueing networks.* IBM Thomas J. Watson Research Center, 1972.

[76] S. Namal, I. Ahmad, A. Gurtov, and M. Ylianttila. Sdn based inter-technology load balancing leveraged by flow admission control. In *Proceedings of the IEEE SDN for future networks and services*, 2013.

[77] E. Navarro and V. Wong. Comparison between vertical handoff decision algorithms for heterogeneous wireless networks. In *Proceedings of the 63rd IEEE Vehicular Technology Conference*, 2006.

[78] A. Neghabi, N. Navimipour, M. Hosseinzadeh, and A. Rezaee. Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature. *IEEE Access*, 6:14159–14178, 2018.

[79] V. Nguyen, T. Do, and Y. Kim. Sdn and virtualization-based lte mobile network architectures: A comprehensive survey. *Wireless Personal Communications*, 86(3):1401–1438, 2016.

[80] D. Nowoswiat and G. Milliken. Alcatel-lucent: Managing the signaling traffic in packet core. *White Paper*, 2013.

[81] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634, 2014.

[82] OpenFlowHub. BEACON. http://www.openflowhub.org/display/beacon.

[83] Open Networking Foundation. https://www.opennetworking.org/sdn-resources/openflow.

[84] Open Networking Foundation. Openflow switch specification version 1.3.1. Technical report, September 2012.

[85] J. Palicot, C. Moy, B. Résimont, and R. Bonnefoi. Application of hierarchical and distributed cognitive architecture management for the smart grid. *Ad Hoc Networks*, 41:86–98, 2016.

[86] C. Perkins. Ip mobility support for ipv4. *RFC 3344*, 2002.

[87] P. Pupatwibul, A. Banjar, A. Sabbagh, and R. Braun. Developing an application based on openflow to enhance mobile ip networks. In *Proceedings of the 38th IEEE Conference on Local Computer Networks Workshop*, 2013.

[88] P. Wang S. Lin and M. Luo. Control traffic balancing in software defined networks. *Computer Networks*, 106:260–271, 2016.

[89] T. Saaty. How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, 48(1):9–26, 1990.

[90] S. Said, M. Sama, K. Guillouard, L. Suciu, and X. Lagrange G. Simon, and J. Bonnin. New control plane in 3gpp lte/epc architecture for on-demand connectivity service. In *Proceedings of the 2nd IEEE International Conference on Cloud Networking*, 2013.

[91] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya. Heterogeneity in mobile cloud computing: Taxonomy and open challenges. *IEEE Communications Surveys & Tutorials*, 16(1):369–392, 2014.

[92] H. Selvi, G. Gür, and F. Alagöz. Cooperative load balancing for hierarchical sdn controllers. In *Proceedings of the 17th IEEE International Conference on High Performance Switching and Routing*, 2016.

[93] Z. Shang and K. Wolter. Delay evaluation of openflow network based on queueing model. *ArXiv preprint ArXiv:1608.06491*, 2016.

[94] R. Sherwood and K.-K. Yap. Cbench controller benchmarker, 2011.

[95] Q. Song and A. Jamalipour. A network selection mechanism for next generation networks. In *Proceedings of the IEEE International Conference on Communications*, 2005.

[96] I. Stojmenovic. *Handbook of wireless networks and mobile computing*. Wiley Online Library, 2002.

[97] S. Sun, M. Kadoch, L. Gong, and B. Rong. Integrating network function virtualization with sdr and sdn for 4g/5g networks. *IEEE Network*, 29(3):54–59, 2015.

[98] J. Sztrik. *Basic queueing theory: Foundations of system performance modeling*. GlobeEdit, 2016.

[99] F. Tam and N.Correia. Fractional switch migration in multi-controller software-defined networking. *Computer Networks*, 157:1–10, 2019.

[100] K. Tantayakul, R. Dhaou, and B. Paillassa. Impact of sdn on mobility management. In *Proceedings of the 30th IEEE International Advanced Information Networking and Applications Conference*, 2016.

[101] C. Thieme. Challenges for modelling of software-based packet processing in commodity-hardware using queueing theory. *Network*, 49:1–7, 2017.

[102] S. Tomovic, M. Pejanovic-Djurisic, and I. Radusinovic. Sdn based mobile networks: Concepts and benefits. *Wireless Personal Communications*, 78(3):1629–1644, 2014.

[103] A. Tootoonchian and Y. Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the internet network management conference on Research on enterprise networking*, 2010.

[104] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2012.

[105] L. Valtulina, M. Karimzadeh, G. Karagiannis, G. Heijenk, and A. Pras. Performance evaluation of a sdn/openflow-based distributed mobility management approach in virtualized lte systems. In *Proceedings of IEEE Global Communications Conference Workshops*, 2014.

[106] H. Wang, H. Xu, L. Huang, J. Wang, and X. Yang. Load-balancing routing in software defined networks with multiple controllers. *Computer Networks*, 141:82–91, 2018.

[107] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, and Y. Sun. Minimizing controller response time through flow redirecting in sdns. *IEEE/ACM Transactions on Networking*, 26(1):562–575, 2018.

[108] T. Wang, F. Liu, J. Guo, and H. Xu. Dynamic sdn controller assignment in data center networks: Stable matching with transfers. In *Proceedings of the 35th IEEE International Conference on Computer Communications*, 2016.

[109] Y. Wang and J. Bi. A solution for ip mobility support in software defined networks. In *Proceedings of 23rd International conference on Computer Communication and Networks*, 2014.

[110] Y. Wang, J. Bi, and K. Zhang. Design and implementation of a software-defined mobility architecture for ip networks. *Mobile Networks and Applications*, 20(1):40–52, 2015.

[111] Q. Wei, K. Farkas, C. Prehofer, P. Mendes, and B. Plattner. Context-aware handover using active network technology. *Computer Networks*, 50(15):2855–2872, 2006.

[112] F. Wibowo, M. Gregory, K. Ahmed, and K. Gomez. Multi-domain software defined networking: Research status and challenges. *Journal of Network and Computer Applications*, 87:32–45, 2017.

[113] R. Wolff. Poisson arrivals see time averages. *Operations Research*, 30(2):223–231, 1982.

[114] P. Wu, L. Yao, and G. Wu C. Lin, and M. Obaidat. Fmd: A dos mitigation scheme based on flow migration in software-defined networking. *International Journal of Communication Systems*, 31(9):e3543, 2018.

[115] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li. Performance evaluation of openflow-based software-defined networks based on queueing model. *Computer Networks*, 102:172–185, 2016.

[116] H. Yan-Nan, W. Wen-Dong, G. Xiang-Yang, Q. Xi-Rong, and C. Shi-Duan. On the placement of controllers in software-defined networks. *The Journal of China Universities of Posts and Telecommunications*, 19:92–171, 2012.

[117] M. Yang, Y. Li, D. Jin, L. Zeng, X. Wu, and A. Vasilakos. Software-defined and virtualized future mobile and wireless networks: A survey. *Mobile Networks and Applications*, 20(1):4–18, 2015.

[118] H. Yao, C. Qiu, C. Zhao, and L. Shi. A multicontroller load balancing approach in software-defined wireless networks. *International Journal of Distributed Sensor Networks*, 11(10), 2015.

[119] L. Yao, P. Hong, and W. Zhou. Evaluating the controller capacity in software defined networking. In *Proceedings of the 23rd IEEE International Computer Communication and Networks Conference*, 2014.

[120] K. Yap, R. Sherwood, M. Kobayashi, T. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar. Blueprint for introducing innovation into wireless mobile networks. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, 2010.

[121] G. Yi and S. Lee. Fully distributed handover based on sdn in heterogeneous wireless networks. In *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, 2014.

[122] J. Yu, Y. Wang, K. Pei, S. Zhang, and J. Li. A load balancing mechanism for multiple sdn controllers based on load informing strategy. In *Proceedings of the 18th Asia-Pacific Network Operations and Management Symposium*, 2016.

[123] J. Zahorjan and E. Wong. The solution of separable queueing network models using mean value analysis. In *Proceedings of the ACM SIGMETRICS Performance Evaluation Review*, 1981.

[124] M. Zekri, B. Jouaber, and D. Zeghlache. A review on mobility management and vertical handover solutions over heterogeneous wireless networks. *Computer Communications*, 35(17):2055–2068, 2012.

[125] S. Zhang, J. Lan, P. Sun, and Y. Jiang. Online load balancing for distributed control plane in software-defined data center network. *IEEE Access*, 6:18184–18191, 2018.

[126] Y. Zhou, M. Zhu, L. Xiao, L. Ruan, W. Duan, D. Li, R. Liu, and M. Zhu. A load balancing strategy of sdncontroller based on distributed decision. In *Proceedings of the 13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2014.

[127] J. Zúniga, C. Bernardos, A. de la Oliva, T. Melia, et al. Distributed mobility management: A standards landscape. *IEEE Communications Magazine*, 51(3):80–87, 2013.