ROYAL INSTITUTE OF TECHNOLOGY
KTH

MASTER THESIS PROJECT

Service Mobility in Mobile Networks

Author: Hany Assasa

Supervisors: Lars Westberg; Vinay Yadhav

Examiner: Peter Sjödin

October 27, 2014

Abstract

In the current mobile network architecture, network traffic between user equipment (UE) and services deployed on the public cloud is tromboned towards the anchor point which could lead to network congestion. Deploying services closer to the UE, for example near the eNodeB, is a potential solution. The services are deployed on small scale data centers connected to, or collocated with the eNodeB, called 'eNodeB-Cloud' (eNBC). Mobility of UEs presents a challenge for deploying services in an eNBC. When the UE is handed over from one eNodeB to another, seamless migration of UE context between the service instances running in different eNBCs needs to be ensured. In this thesis, we propose a Platform as a Service framework to enable UE context migration between eNBCs. The architecture consists of handover signaling mechanism, network session migration technology, context transfer protocol and a set of APIs towards the service. The evaluation of the prototype implementation shows that virtualization causes some extra delays to the UE context migration time. Whereas when virtualization is omitted, the time taken to migrate a UE context between two eNBCs is in the order of 12 ms on average, which is within the limit of handover interruption time between two LTE-eNodeBs.

Index Terms- eNodeB-Cloud, UE Context Migration, Network Session Mobility, Handover Signaling System.

Contents

Contents	i
List of Figures	iii
List of Tables	v
Terms and Abbreviations	vii
1 Introduction	3
1.1 Background	3
1.2 Problem Statement	7
1.3 Challenges	8
1.4 Thesis Methodology	8
1.5 Goals and Objectives	10
1.6 Thesis Outline	10
2 State of the Art	11
2.1 Session Migration Approaches	11
2.1.1 Migratory TCP (M-TCP)	11
2.1.2 MSOCKS	11
2.1.3 Reliable Network Connections	12
2.1.4 TCP Connection Passing (tcpcp)	12
2.1.5 Service Continuations (SC)	12
2.1.6 Mobile IP	12
2.1.7 SockMi	12
2.1.8 Session Migration Approaches Comparison	13
2.2 Linux Kernel Networking and Structure	14
2.2.1 Linux Kernel Module (LKM))	14
2.2.2 NetFilter	14
2.3 Media Independent Handover Protocol (MIH)	15
2.3.1 Introduction	15
2.3.2 Objectives	15
2.3.3 MIHF General Architecture	16
2.3.4 MIHF Communication Model	17
2.3.5 MIHF Services	18
2.3.6 MIH Protocol	20
2.4 Context Transfer Protocol (CXTTP)	21
2.4.1 Introduction	21
2.4.2 Protocol Overview	21
2.5 3GPP 4G-LTE Network	24

2.5.1	Network Architecture	25
2.5.2	Mobility in LTE	26
3	Design	29
3.1	Service Mobility in Deployment	29
3.2	System General Design Requirements	30
3.3	General Design Framework	30
3.3.1	Handover Signaling System Block	30
3.3.2	Network Session Migration Block	31
3.3.3	Context Transfer Block	32
3.3.4	Migration Signaling Flow Block	32
3.3.5	Service Mobility APIs (PaaS Interaction) Block	32
3.4	Proposed Framework Architecture	33
3.4.1	Service Mobility Process	34
3.4.2	Migration Signaling Flow	35
3.4.3	Service Mobility Interface APIs	39
4	Implementation	43
4.1	State Machines	43
4.1.1	Source State Machine	44
4.1.2	Destination State Machine	46
4.2	Software Modules	47
4.2.1	Session Migration Module	48
4.2.2	MIH User Module	51
4.2.3	CXTP Module	53
4.2.4	Performance Logger Module	56
4.2.5	Mobility Hook Module	56
4.3	LENA LTE Simulator	57
5	Experimental Evaluation	59
5.1	Video Server Service	59
5.2	Test Scenario	60
5.3	Test Setup 1	60
5.3.1	Evaluation and Analysis	64
5.4	Test Setup 2	71
5.4.1	Evaluation and Analysis	72
6	Conclusions	75
6.1	Future work	75
6.1.1	Potential Services	75
6.1.2	Further Design Improvements	76
	Bibliography	79

List of Figures

1.1	Anchor Points in Mobile Network	4
1.2	Video Server Service's Context	4
1.3	Illustration of Service Mobility	5
1.4	Service Mobility Deployment Scenario	6
1.5	Quality of Service Improvement	7
2.1	Migration Mechanism Design in SockMi	13
2.2	Netfilter	14
2.3	MIHF General Architecture	16
2.4	MIHF Communication Model	17
2.5	Remote MIH Command	19
2.6	MIH Command Service Flow	20
2.7	MIH Protocol Frame	20
2.8	MIH Protocol Head	21
2.9	MIH Message Parameter TLV Encoding	21
2.10	CXTP Messages General Format	22
2.11	Context Transfer Data (CTD) Message	24
2.12	Context Transfer Data Reply (CTDR) Message	24
2.13	LTE Network Architecture	25
2.14	Mobility over X2 Interface	27
3.1	Service Mobility In Deployment	29
3.2	Service Mobility General Design Framework Diagram	30
3.3	Handover System Design Block	31
3.4	Network Session Migration Block	31
3.5	Context Transfer Block	32
3.6	Service Mobility Framework Proposal	33
3.7	Service Mobility Framework Architecture	34
3.8	MIH LINK SAP as a Driver	35
3.9	Migration Signaling Flow	37
3.10	Service Mobility Interface APIs	39
4.1	Transaction Pool	44
4.2	Source State Machine	45
4.3	Destination State Machine	46
4.4	Service Mobility Software Modules	47
4.5	sk_buffs Ring	50
4.6	Saving TimeStamp values	50
4.7	Post-hook Filter for Migrated Socket	51
4.8	Pre-hook Filter for Migrated Socket	51

4.9	CXTP Message Archive	54
4.10	CTD Message Hierarchy	55
4.11	Context Transfer Data (CTD) Message Modified Version	56
5.1	HTTP Transaction for Video Server Service	59
5.2	Real Deployment Scenario	60
5.3	Test Setup 1	61
5.4	Demo Scenario	63
5.5	Histogram of UE Context Migration Time for Test Setup 1	64
5.6	Histogram of Various Activities for Test Setup 1	65
5.7	Histogram of UE Context Migration Time without Application's Time	66
5.8	Histogram of UE Context Migration Time Without Mutex	66
5.9	Transmission between two VMs	67
5.10	Histogram of CXTP Serialization and Transmission between two VMs	68
5.11	Transmission between two Physical Machines	68
5.12	Histogram of CXTP Serialization and Transmission between two Physical Machines	69
5.13	Histogram of UE Context Migration Time after Removing CXTP TX Time	69
5.14	Histogram of Context Data Length	70
5.15	Test Setup 2	71
5.16	Histogram of UE Context Migration Time for Test Setup 2	72
5.17	Histogram of Various Activities for Test Setup 2	73
6.1	CXTP Proxy Framework Architecture	77
6.2	Signaling Flow with CXTP Proxy Enabled	77
6.3	Tunnelling MIH and CXTP over X2 and S1-MME Framework Architecture	78

List of Tables

2.1	Session Migration Approaches Comparison	13
2.2	CDB Fields Description	23
4.1	State Machine Variables	44
4.2	MIH_N2N_HO_Commit Request Primitive Parameters	53
4.3	MIH_N2N_HO_Commit Response Primitive Parameters	53
4.4	Performance Values for Different Activities	58
5.1	Statistical Results for UE Context Migration Time Components in [ms] for Test Setup 1 .	64
5.2	Statistical Results for UE Context Migration Time After Eliminating the Mutex in [ms] .	67
5.3	Statistical Results for CXTP Transmission Time in [ms]	68
5.4	Statistical Results for Context Data Length in [KB]	70
5.5	Statistical Results for UE Context Migration Time Components in [ms] for Test Setup 2 .	73

List of Abbreviations

3GPP	Third Generation Partnership Project
AP	Access Point
API	Application Programming Interface
AR	Access Router
CDB	Context Data Block
CT	Context Transfer
CT-Req	Context Transfer Request
CTAA	Context Transfer Activate Acknowledge
CTAR	Context Transfer Activate Request
CTC	Context Transfer Cancel
CTD	Context Transfer Data
CTDR	Context Transfer Data Reply
CXTP	Context Transfer Protocol
E-UTRAN	Evolved UMTS Terrestrial Radio Access Network
eNBC	eNodeB-Cloud
EPC	Evolved Packet Core
EPS	Evolved Packet System
FPT	Feature Profile Type
GRE	Generic Routing Encapsulation
GSM	Global System for Mobile Communications
GTP	GPRS Tunneling Protocol
GUID	Globally Unique Identifier
HO	Handover
HSS	Home Subscriber Server

IaaS	Infrastructure as a Service
IE	Information Element
IMS	IP Multimedia Subsystem
ISP	Internet Service Provider
LBO	Local Break Out Function
LTE	Long Term Evolution
M2M	Machine to Machine
MICS	Media Independent Command Service
MIES	Media Independent Event Service
MIH	Media Independent Handover
MIHF	Media Independent Handover Function
MIIS	Media Independent Information Service
MME	Mobility Management Entity
MN	Mobile Node
nAR	New Access Router
NAT	Network Address Translation
NF	NetFilter
OS	Operating System
OVS	Open Virtual Switch
PaaS	Platform as a Service
pAR	Previous Access Router
PCRF	Policy Control and Charging Rule Function
PDCP	Packet Data Convergence Protocol
PGW	Packet Data Network
PGW	Packet Data Network Gateway
PM	Physical Machine
PoA	Point of Access
PoS	Point of Service
RCC	Radio Resource Control
RNL	Radio Network Logic

SAE	System Architecture Evolution
SAP	Service Access Point
SGW	Serving Gateway
STD	Standard Deviation
TLV	Type Length Value
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
UTRAN	UMTS Terrestrial Radio Access Network
VM	Virtual Machine
VoD	Video on Demand

Chapter 1

Introduction

1.1 Background

Mobility in wireless networks concerns the ability to change the attachment point of a wireless link in a seamless way to the end user, conditioning minimal degradation in the received signal quality. Mobility management functionality is a fundamental part in current mobile networks. It allows network operators to track the movement of mobile terminals, and provide them with permanent radio coverage as they move across different access points.

Efficient mobility management solutions focus on optimizing handover failure rate which means that the packet delivery latency, and packet dropping rate caused by user's mobility should be kept at minimum level. However, current solutions for mobility rely on the concept of anchor points [1, 2] which are network entities located at fixed locations in the network hierarchy. These anchor points maintain users' connections and relay traffic from and to them. Anchor points ease mobility management of mobile terminals, but on the other hand they have several drawbacks: they constitute a single point of failure and potential bottleneck in the network. This is because all the traffic in the network has to go through them which leads to waste of network capacity, high energy consumption, poor quality of service and traffic tromboning.

Additional to the disadvantages associated with the implementation of anchor points in a mobile network, the increasing number of mobile users has a major implication on the performance of these networks. Due to the new services available today like Video on Demand (VoD), and mobile gaming, the mobile users demand for higher data rates and better quality of services [3]. This forces operators to expand their networks to cope with their clients' demands, which results in high CAPEX and OPEX. This is because operators have to deliver high-quality data services with fast response times. During peak hours, service delivery could be delayed due to bottlenecks in the mobile core networks, which causes severe damage to the operators' businesses. Hence, operators have to perform high maintenance cost since upgrading hardware is not sufficient to handle the sudden increase in data traffic and poor quality of service leads to unsatisfied customers and eventually to lose money.

Figure 1.1 shows a general mobile network architecture with multiple access points. The mobile network gateway in this network is considered as the anchor point, where all the traffic in the network bypasses through it. Additional to that, each cell has a different number of mobile users at different times. As a result, some cells need to have flexible bandwidth to adapt with sudden changes in the number of mobile users.

To overcome the previous problems and obstacles, we propose a solution in which we deploy

Figure 1.1. Anchor Points in Mobile Network

the services closer to the mobile users and near the access points. The services can be hosted on small scale data centers connected to or collocated with the eNodeB, called 'eNodeB-Cloud' (eNBC). Though we can address the issue of traffic concentration and network congestion at the anchor points by deploying services on eNBCs, it raises the issue of seamless UE mobility without any service disruption. In the absence of a mobility system for migrating the UE contexts, when a handover of a UE occurs, the UE session associated with the service running on the eNBC is disrupted, which adversely impacts the quality of service.

Figure 1.2. Video Server Service's Context

A UE context consists of a network session between the service and the UE, as well as the context related to the UE in the service. For example, in case of HTTP video server as depicted in figure 1.2, the service context associated with a UE consists of the requested video file name and the current offset in the file. The network session in the example relates to the TCP connection between the UE and the HTTP video server. When a user sends an HTTP request message to the video server asking for a particular video file to be streamed back to his device. The video server replies back to the user with an HTTP response message containing the content of the video file. And in the same time, it creates a service context for that particular user and adds it to its internal table. This context is updated each time the video server writes bytes to the client. When the user's request has been successfully completed, the file offset will be equal to the total length of the that file.

We propose in this thesis a new framework for mobility based on the innovative concept of Service Mobility where services move together with their clients. This concept takes advantages of current developments in the area of cloud computing and networking, in which computing resources become ubiquitous, elastic, and self-configuring. This enables network service providers to realize support of mobility in a cost-effective way compared to the current mobility approaches. Figure 1.3 illustrates the concept of service mobility. Each eNodeB is connected to eNodeB-Cloud. The eNBC consists of several virtual machines running various applications and services such as Apache Web service, Video Streaming service, etc. When the User Equipment (UE) moves towards eNodeB2, all the contexts associated with that user should move with him towards eNBC2. The whole UE context migration is transparent to the end user, and there is no discontinuity in the services being delivered to the UE.

Figure 1.3. Illustration of Service Mobility

Figure 1.4 shows the proposed deployment scenario for the service mobility framework, where

cloud computing capabilities and networking are integrated inside next generation mobile networks architecture. In this deployment scenario, the cloud deployments are arranged in a tree topology structure. The tree structure has a root node which is the Primary Cloud-Site, intermediate nodes (Hub-Cloud), and leaves (eNodeB-Clouds). Primary Cloud-Site is the main data center where the services are hosted. Each node depending on its level in the tree has certain contents determined by the requests it receives from its child nodes. For the eNodeB-Clouds, requests shall be received from the mobile users. As a result, the information regarding the availability of the services and contents in each cloud-center should exist as database. Having such an architecture can help delivering services to the end user in an optimal way depending on its location.

Figure 1.4. Service Mobility Deployment Scenario

The chosen network topology forces any node to support two different types of migrations namely horizontal and vertical migrations. Horizontal migration implies the ability to move contexts between eNBCs in the same level of hierarchy conditioning that the destination cloud eNBC has the same services and contents of the source eNBC. Whereas vertical migration means the ability to move UE context to a higher level in the network hierarchy. This happens only if the horizontal migration is not possible due to the unavailability of the contents in the destination eNBC in the same level. For example, a vertical migration can be from eNBC cloud to a Hub-Cloud.

In figure 1.4, at the beginning the user is using some services hosted on the eNBC at Point (A). Then, the user moves towards point (B), but this time the requested contents do not exist in the target eNBC; they are located in the Hub-Cloud. As a result, a vertical UE context migration is performed to move the contexts associated with the user from the eNBC to the Hub-Cloud. Later on, the user continues his mobility and moves towards point (C), at this point the contexts are neither available in the eNBC nor in the Hub-Cloud. However, they exist in the Secondary site-cloud which implies carrying out a vertical migration, but this time it's from the Hub-Cloud to the Secondary site-cloud. Finally, when the user reaches point (D), the contents are available again in the eNBC

there. This ends up in performing a vertical migration from the Secondary site-cloud to the eNBC. This way of deployment requires a standardized framework to authorize the movement of contexts between eNBCs in different levels, which was designed and developed during this thesis work.

Finally, we can say that deploying service mobility could reduce energy consumption which is vital for the environment, society, and future generations. In the same time, it would allow to use network resources more efficiently and improve the quality of services. This is because, the number of network elements and computing resources involved in delivering the packets between the client and the end server is reduced dramatically as shown in figure 1.5. In addition, distributing services over a pool of servers would prevent possible network failure due to sudden traffic increases on specific points in the network. When mobile network operators upgrade their current mobile networks and integrate them with cloud computing capabilities and provide service mobility as PaaS, they would be able to cut down the budget required for CAPEX and OPEX. This is because network mobile operators would not have to upgrade their networks continuously to meet up the increasing demands of their subscribers.

Figure 1.5. Quality of Service Improvement

1.2 Problem Statement

This thesis addresses the need for service mobility in mobile network and is an effort towards integrating cloud computing capabilities in next generation mobile networks. Mentioned below are the functions defined and realized during the period of this thesis:

- **Network Session Migration:** Since services are located closer to the end user, and due to the mobility of the user, contexts associated to that user have to support mobility too. This implies re-establishing both service context and network sessions contexts at the destination eNBC, starting from the state in which they left off in the source eNBC.
- **Vertical & Horizontal Migration Capabilities:** If the user is moving towards eNBC serving the same contents as the source eNBC, a horizontal UE context migration shall take place. Whereas for the complementary case in which the contents are not available in the destination eNBC, a mechanism to move UE context to a different level in the network hierarchy is required. This results in a vertical context migration to be carried out.
- **Contexts Transfer:** Minimalistic state transfer for re-establishment of execution in the new eNBC. These states include a minimum amount of information about network sessions contexts and their corresponding services contexts.
- **An abstract API layer to support UE context migration:** Define a set of API calls to allow services to interact with the service mobility system. This set of API calls allows services to export/import contexts related to the UE and its network sessions.

1.3 Challenges

Realizing service mobility in mobile networks has multiple challenges and difficulties, the most significant challenge in the development of the framework is its design, which will affect its deployment in real mobile networks. Significant challenges in the design of the service mobility are described below:

- **Interoperability:** Interoperability is defined by EICTA [4] as "The ability of two or more networks, systems, devices, applications or components to exchange information between them and to use the information so exchanged". Network interoperability can be achieved in two ways either by defining a standard interface to which all network entities adhere to or by providing an intermediate network entity between them that translates messages exchanged between these two protocols [5]. Mobile networks use standardized interfaces defined by 3GPP to establish communication and exchange signaling among different entities supplied by various vendors. For this reason, all interfaces exposed by the service mobility framework shall depend only on the available standardized protocols to satisfy its various purposes.
- **Low complexity:** Minimum number of API calls to interact with the service mobility shall be taken into consideration. Details regarding network session migration and handover signaling mechanisms shall be hidden from service developer. In addition, supporting service mobility should require minimum modifications on the current services' implementation.
- **Fast context migration procedure:** Migrating a UE context between eNBCs should require a minimum amount of time, and it should not affect end user's experience.

1.4 Thesis Methodology

The main essence of this thesis is in the development of the service mobility framework. Developing a new system requires going through three different phases:

- **Design Phase:** The design phase is the first and essential step, since a good design is an important step to building an effective solution in terms of simplicity, performance and future

extensibility. In this phase, various design blocks characterizing the system are explored. In addition, requirements for each of these design blocks are identified, and any alternative design approaches are examined and compared among each other to pick up the best approach. There are five design blocks that constitute service mobility framework namely: handover signaling system, contexts transfer technology, network session migration, migration signaling flow, and platform as a service (PaaS) API interface. The following paragraphs enumerate the approaches taken for each design block.

The handover signaling system is the first design block. It includes a triggering mechanism which informs eNBC to start a UE context migration process. The triggering mechanism should conduct information about the mobile node for which the L2 handover is in action, and the destination to where the mobile node is handed over. In addition, depending on the information available in the internal database about the target eNBC, the handover signaling system should be able to perform two types of migration operations namely horizontal migration and vertical migration.

The requirement from the contexts transfer technology is to be able to migrate both service context and network session context, between network entities involved in service migration. It is important that the approach selected be standardized to allow interoperability between different vendors. In addition, it should allow the identification of various contexts that belong to different services in a unique way, so there is no place for ambiguity.

The design block of network session migration is an essential part in the service mobility system. It is responsible for obtaining states and information regarding network session to be migrated. In terms of evaluating various options in this design block, different network session migration solutions are considered and compared against each other based on various factors such as transparency, portability, dynamicity and implementation availability. A comparison is made between current existing solutions, and a list of pros and cons for each is documented. It is important that the mechanism selected meets most of the requirements mentioned previously.

The signaling flow design block is responsible for defining a sequence of the signals to accomplish UE context migration. In addition, it should deliver information regarding the role of each node involved in this task. The primary requirement for this block is to generate an optimized signaling flow capable of moving UE context across eNBCs in a short amount of time.

The Platform as a Service (PaaS) API Interface is the final design block. It is important from an ease of deployment perspective for the proposed framework. It should be agnostic to the underlying technologies used by the service mobility system. Besides that, it should be easy and straightforward for the service developer to interact with service mobility system.

- **Implementation Phase:** In this phase, modular programming is a software design technique chosen to implement the functions of the previous design blocks. Depending on the technologies chosen to fulfil these tasks, different open source implementations for these technologies are investigated and compared against each other based on the quality of the code and availability of support. The implementation that satisfies most of the requirements mentioned before is picked up. For technologies which have been documented but not implemented yet, a collection of modern tools and methods including state machines and object-oriented programming are used to realize their functionalities.

- **Evaluation Phase:** It is the final phase in the development process. In this phase, performance parameters characterizing the system are identified. Evaluation of the implemented system is carried out in an emulated environment similar to the real deployment scenario. Conclusions about the performance results are drawn which may lead to the calibration of the design or the implementation to obtain more optimized results.

1.5 Goals and Objectives

The goals of this master thesis can be described as follows:

- Explore current available solutions for migrating TCP sessions with emphasis on their implementation and performance.
- Investigate Media Independent Handover (MIH) IEEE 802.21 standard to assist UE context migration.
- Examine Context Transfer Protocol RFC4067 standard to transfer UE context.
- Integrate a session migration approach, the MIH framework, and the CXTTP protocol in one system to enable UE context migration.
- Build a prototype for the proposed service mobility framework and integrate it with a simple mobile network.
- Evaluate the prototype by measuring the time taken to migrate a UE context.

1.6 Thesis Outline

This work is organized as follows:

- *Chapter 2* contains the background study of the available options for migrating network sessions with an overview of both media independent handover protocol IEEE MIH 802.21 and Context Transfer Protocol (CXTTP) RFC4067.
- *Chapter 3* contains the requirements and the design of the service mobility system. The design includes the framework architecture, the handover signaling flow, and the application programming interfaces (APIs) used by various services to interact with the service mobility system.
- *Chapter 4* includes description and explanation of the implementation stage. Also explains how the design proposed in *Chapter 3* is implemented.
- *Chapter 5* evaluates and analyses the performance of the implemented service mobility prototype.
- *Chapter 6* summarises the work and draws some conclusion out of it. Moreover, some suggestions for the future work are provided.

Chapter 2

State of the Art

This chapter includes a study of various approaches to perform TCP session migration. Then, an overview of both media independent handover protocol IEEE MIH 802.21 and Context Transfer Protocol (CXTP) including services, signaling flow model and structure is introduced. Finally, a brief introduction about 4G-LTE/SAE architecture and mobility support is presented.

2.1 Session Migration Approaches

Most of the networking services today are built over TCP protocol, which provides reliable service over a non-reliable transport medium. To make a connection, a transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end. This need creates a strong binding between a service and the IP address of the server providing it, during the lifetime of a TCP connection. Due to this, a TCP client will be vulnerable to any adverse conditions that may affect the TCP endpoint of the server or the network in between, after a connection is created: network congestion, failure or server overloaded. One solution to the previous problem could be by migrating the TCP/IP connection from the point of failure to a stable point. Practically this means to substitute at least one of the peers with another process located in the same or different system. Additional to mitigating endpoint failure, migration of TCP/IP connections can be useful when there are requirements of load balancing, quality of service, and security [6, 7]. In this section we introduce a quick overview of different mechanisms proposed to perform TCP/IP session migration, with an emphasize on their implementations and performance.

2.1.1 Migratory TCP (M-TCP)

Migratory TCP (M-TCP) is a reliable connection-oriented transport layer protocol that allows the mobility of living TCP/IP connections. The protocol enables a service to be resumed seamlessly by transmitting a specific state between stateful servers. However, migration of the client endpoint is not supported by the current implementation. In addition, the IP addresses of the cooperating servers must be known in advance [6, 7].

2.1.2 MSOCKS

Typical mobility mechanisms do not enable applications to take advantage of more than one network interface at a time. Due to this MSOCKS was introduced as a new architecture for transport layer mobility which permits mobile hosts to change their attachment point on the Internet. MSOCKS also provides the capability to control data flow through different network interfaces. Achieving the transport layer mobility scheme is done by using an architecture of split-connection proxy and a technique named TCP Splice. TCP Splice technique gives the same end-to-end semantics as a normal

TCP session for split-connection proxy systems. Both TCP sequence and ACK numbers in MSOCKS are modified when they go through the proxy, which results in updating the TCP check-sum field in each TCP header. According to the claim of the authors, the operation's overhead is limited. However, it is not stated whether their mechanism scales and adapts well with a high number of TCP connections [6, 8].

2.1.3 Reliable Network Connections

Reliable sockets (ROCKS) and reliable packets (RACKS) are two techniques that provide transparent mobility of network connections and protection against network failures. ROCKS does not require kernel modifications nor be installed by privileged users and work at user-space. Both systems can detect a network failure within seconds of its occurrence, keep the failed endpoint of a connection in a consistent suspended state for a certain amount of time. Then, they automatically reconnect, even when one endpoint of the connection changes its IP address, without loss of in-flight data. ROCKS and RACKS are transparent to applications, but they must exist at both ends of the connection [6, 9].

2.1.4 TCP Connection Passing (tcpcp)

TCP Connection Passing is a mechanism that allows applications to migrate the kernel state of a TCP connection's endpoint between hosts while the connection is in the established state. tcpcp does not require the peer to cooperate in any way. However, tcpcp is more a building block rather than a complete solution for session migration. For example, it does not define a mechanism for the IP packet redirection in the network. Additional to that, the migration is invoked only by the process which means it is not possible to force that process to migrate its sockets [6, 10].

2.1.5 Service Continuations (SC)

Service Continuations is an operating system mechanism that supports the migration of Internet service sessions between cooperating servers in a seamless and dynamic way. Service Continuations provides server applications with a simplified abstraction and transparent solution to migrate both service state with the serviced connection. SC is a generic, application-independent, and does not require any application knowledge at the OS or transport level for resuming service after migration. However, no software distribution is available to be tested and evaluated [11].

2.1.6 Mobile IP

IETF proposed the Mobile IP standard to solve the host mobility problem between different networks. Mobile IP allows a host to maintain its current transport-layer connections as its mobiles across various networks. It requires mobile host to use two addresses: one home address and one care-of address. The home address is a permanent address, whereas the care-of address changes when the mobile host moves between networks. Mobile IP addresses the problem of host mobility, but it does not take into consideration connection migration between servers [12, 13].

2.1.7 SockMi

SockMi is a solution for migrating TCP/IP connections between servers running Linux systems. The migration is completely transparent to the client that can reside on a machine running any OS. With respect to other solutions, SockMi design shown in figure 2.1 achieves the following goals [6, 14]:

Figure 2.1. Migration Mechanism Design in SockMi

- **Transparency:** A connection end-point which is not involved in a migration mechanism is not affected in any way. This means that no cooperation between both ends is required to complete the migration.
- **Flexibility:** It means the ability to migrate a stream socket in any internal state.
- **Portability:** The migration mechanism should be independent as much as possible from the underlying operating system and does not require any modification to the current TCP protocol stack.
- **Symmetry:** It means the ability to migrate both connection end-points at any time during the connection lifetime.
- **Dynamicity:** A connection end-point can be migrated many times during its lifetime to any server.

2.1.8 Session Migration Approaches Comparison

Table 2.1 summarizes differences between different session migration mechanisms based on transparency, portability, dynamicity, implementation availability and whether it is a complete solution to migrate a TCP socket. Complete solution means that the session migration approach defines a mechanism for redirecting IP packets in the network. Additional to that, it determines whether an external process can force the migration.

Approach/Item	Transparency	Portability	Dynamicity	Complete Solution	Implementation Availability
M-TCP	✓	✓	X	✓	✓
MSOCKS	✓	X	✓	✓	✓
Reliable Network Connections	X	✓	✓	✓	✓
TCP Connection Passing	✓	✓	✓	X	✓
Service Continuations	✓	✓	✓	✓	X
Mobile IP	✓	✓	X	X	✓
SockMi	✓	✓	✓	✓	✓

Table 2.1. Session Migration Approaches Comparison

2.2 Linux Kernel Networking and Structure

2.2.1 Linux Kernel Module (LKM))

A kernel module is simply a standard program used to extend the core Linux Kernel functionality in a dynamic way. Usually, LKM are used to add support for new hardware devices or new files systems. Without an LKM, the base kernel had to include all possible anticipated features that will be used and not used which would result in waste of memory. Moreover, every time a new functionality has to be added, the user would have to rebuild the base kernel again and reboot the machine. The kernel module solves this issue by supporting hot-plugging, so when the system detects a new device; it loads the corresponding driver without the necessity to restart the system.

2.2.2 NetFilter

The netfilter subsystem provides a framework to mangle packets during their traversal of the TCP/IP protocol stack. Each protocol defines a set of hooks that are well-defined points in the protocol stack. The kernel registers some callbacks to listen to some hooks for different protocols. When a packet passes through the netfilter framework, it checks if anyone has registered for that protocol and hook. If so then some operations can be executed on the packets such as changing addresses, ports dropping the packets, logging and more. There are five points in the network stack where the netfilter has hooks as shown in figure 2.2, the name of the hooks are as follow [15, 16]:

Figure 2.2. Netfilter

1. **NF_IP_PRE_ROUTING**: It is the first hook point in the netfilter subsystem after the packets have passed the simple sanity checks and before performing a look-up in the routing subsystem. The routing subsystem decides whether the packet is destined for a different interface or to the local host. It may drop packets that can not be routed.
2. **NF_INET_LOCAL_IN**: All incoming packets with a destination to the local host reach this hook point after first passing via the **NF_INET_PRE_ROUTING** hook point and after performing a look-up in the routing subsystem.
3. **NF_INET_FORWARD**: All forwarded packets reach this hook point after first passing via the **NF_INET_PRE_ROUTING** hook point and after performing a look-up in the routing subsystem.
4. **NF_INET_POST_ROUTING**: Packets that are forwarded reach this hook point after passing the **NF_INET_FORWARD** hook point and before being put on the wire again. Also,

the packets that are created in the local machine and sent out arrive at `NF_INET_POST_ROUTING` after passing the `NF_INET_LOCAL_OUT` hook point.

5. **`NF_INET_LOCAL_OUT`**: All outgoing packets that were created on the local machine reach this point before reaching the `NF_INET_POST_ROUTING` hook point.

2.3 Media Independent Handover Protocol (MIH)

Terminals with multi-access technologies have become available in the recent years. These multi-mode devices add new challenges to mobility management in mobile networks. As a result, the IEEE released new standard for media-independent handover services (IEEE 802.21 MIH) to address these challenges. MIH aims to improve user experience of mobile terminals by enabling handovers between heterogeneous access technologies while optimizing session continuity [17]. In the following section, an overview of IEEE 802.21 MIH is presented starting with a short introduction to the standard and its objectives. Then, the MIH architecture with its various components are briefly introduced. Finally, the MIH protocol frame and specifications are described.

2.3.1 Introduction

Current mobile devices are equipped with multiple network interfaces belonging to different access technologies. A network connection should be established using the best possible interface which might imply higher data rate, lower latency and delay. During a communication session, the characteristics of the serving network may change or a new access network is reached by Mobile Node (MN) and this network might be better in terms of signal quality and coverage; thus, already established connection should be migrated from the current interface to another. Most of the mobility management solutions address handovers within the same type of access technology, but none has ever addressed the ability to perform a vertical handover between different access technologies. For this purpose, the IEEE has proposed and introduced the 802.21 standard which provides necessary network information to higher layers to carry out an optimized handovers between heterogeneous networks and preserve session continuity without having to deal with the specifications of each link layer technology. All media types specified by Third Generation (3G) Partnership Project (3GPP), 3G Partnership Project 2 (3GPP2), and both wired and wireless media in the IEEE 802 family of standards are included in this standard [17, 18, 19].

So we can say that the main purpose of the IEEE 802.21 is to enable the optimization of handovers between heterogeneous access network technologies (cellular technologies and including IEEE 802) with minimum service interruption, hence improving mobile user experience. IEEE 802.21 provides the missing, technology-independent, abstraction layer capable of providing a common unified interface to the upper layers protocols, thus hiding media layer technology-specific primitives and details. This abstraction can be exploited by any upper layer protocols like IP stack to interact with the underlying technologies, thus leading to improved handover performance [17, 18, 19].

2.3.2 Objectives

While developing the 802.21 standard, the IEEE had the following objectives in mind:

1. Design a framework that enables seamless handover between different wireless access technologies while maintaining service during the transitions of the mobile node between these technologies. This framework requires all the devices involved in the handover to enable a new protocol stack. This protocol stack provides the mandatory interactions among devices for optimizing handover decisions.

2. Define a new link layer SAP "service access point" that is independent of the underlying link layer technology specifics called the MIH_LINK_SAP. This SAP is mapped to technology-specific primitives defined by different media types. The new SAP primitives assist the MIHF collect link information and control link behaviour during handover. The new SAPs are recommended as amendments to the standards for the respective link-layer technology.
3. Define a set of handover functions that provide the upper layers such as mobility management protocols with the required commands to perform enhanced handovers. These new functions are provided by an entity called the MIH Function (MIHF). The MIHF provides the following services:
 - a) The media independent event service (MIES) provides event classification, event filtering, and event reporting. These events correspond to the detection of changes in link-layer properties, status and quality.
 - b) The media independent command service (MICS) provides the MIH users with a list of commands to control different link layers properties. These commands are relevant to handover and mobility and perform switching between links if required.
 - c) The media independent information service (MIIS) provides the information about available service in different neighbouring networks. This information enables more effective handover decisions to be made across heterogeneous networks.

2.3.3 MIHF General Architecture

Figure 2.3 shows the location of the MIHF in the protocol stack and its interaction with other elements of the system. Messages exchange between the MIHF and other functional entities take place through service primitives, grouped in service access points (SAPs).

Figure 2.3. MIHF General Architecture

The 802.21 standard includes following three SAPs:

1. **MIH_SAP**: Media independent interface which allows communication between the MIHF layer and upper layer of the protocol stacks (MIH-Users).
2. **MIH_LINK_SAP**: Abstract media dependent interface between the lower layers of the protocol stack and the MIHF layer.

3. **MIH_NET_SAP**: Abstract media dependent interface through which remote MIHF entities exchange information between over either L2 or L3 transport services.

2.3.4 MIHF Communication Model

Figure 2.4 represents the 802.21 reference model, which includes the following network entities:

Figure 2.4. MIHF Communication Model

- **MIH point of service (MIH PoS)**: An MIHF is called MIH PoS when it communicates directly and exchanges MIH information with an MN-based MIHF.
- **MIH non-PoS**: It is a network entity that does not have direct communication with the MN; thus, it can not exchange any MIH message with it.
- **MIH point of attachment (PoA)**: An MIHF is considered to be an MIH PoA when it is the endpoint of a layer two communication link that has an MN as the peer endpoint.

To allow communication between these network entities possible, the reference model defines several reference points as follow:

- **Communication Reference Point R1 (MN ↔ Serving PoA (PoS))**: It is used by the MN to communicate with its PoA over both L2 and L3 and above. Information regarding the status of the connection can be retrieved over this reference point.
- **Communication Reference Point R2 (MN ↔ Candidate PoA (PoS))**: It is defined as an interface between a mobile node and a candidate PoA. The communication over this reference

point takes place on both L2 and L3 and above. Information about candidate PoAs is gathered before making a handover decision.

- **Communication reference point R3 (MN↔non-PoA (PoS)):** This communication reference point is used to establish communication between the mobile node and an MIH PoS located on a non-PoA network entity. Different IP configuration modes in the network is one kind of information that can be carried over this interface.
- **Communication reference point R4 (PoS↔non-PoS):** Communication over this reference point is done between an MIH PoS in a network entity and an MIH non-PoS in another network entity. This reference point encompasses communication interfaces over L3 and above, and it is typically used when an MIH server serving a mobile node (the PoS) needs to obtain some information from another MIH server (the non-PoS).
- **Communication reference point R5 (PoS↔PoS):** This communication point refers to the procedures used between two different MIH PoSs located at different network entities. The communication is established over L3 and above.

2.3.5 MIHF Services

The 802.21 architecture and reference model present a framework that defines necessary information to optimize and facilitate vertical handover across different media access technologies. There are three different types of communications, each one of them has its associated semantics. These three types of communications are called MIH services and defined by the IEEE 802.21 as following:

- Event Services (ES).
- Command Services (CS).
- Information Services (IS).

These services allow MIH users to access handover-related information, listen to different link-layer events, in addition deliver commands to the lower layers or network entities. These three services can be delivered in two ways:

- **Asynchronous Delivery:** Events generated in link layers and transmitted to the MIHF or MIH users are delivered asynchronously.
- **Synchronous Delivery:** Commands and information generated through a query and response mechanism are delivered synchronously.

These services are managed and configured through service management primitives.

2.3.5.1 Service Management

Before an MIH entity starts to receive services from different MIHF nodes, it must be configured properly with its local MIHF. This is done through the following service management functions:

- **MIH Capability Discovery:** This procedure is used by an MIH-User to discover the capabilities of local or remote MIHF in terms of MIH services.
- **MIH Registration:** It is defined as a way of requesting access to specific MIH services and declaring the presence of an MIH-User entity.
- **MIH Event Subscription:** This mechanism allows an MIH-User to subscribe and listen for a particular set of events that originates from a local or remote MIHF.

2.3.5.2 Media Independent Event Service (MIES)

MIH-Users can subscribe to receive event notifications from a particular event source. When an event is generated, it will be delivered to the subscription list. The MIHF can help in dispatching these events to multiple destinations. Events can be divided into different groups:

- **Link Events:** These events are originated from link layers and terminated at the MIHF. IEEE 802.x, 3GPP and 3GPP2 MIH LINK SAP generate and propagate these link events.
- **MIH Events:** These events are propagated from one MIHF to another MIHF, or from the MIHF to the MIH User.

2.3.5.3 Media Independent Command Service (MICS)

The media-independent command service (MICS) provides a set of commands to obtain the status of wireless links and configure the terminal to gain optimal handover performance. These commands are sent from the MIH-Users or upper layers to the lower layers. For example, the network initiates and controls handovers to balance the load of two different access networks. The commands are classified into two categories:

- **Link Commands:** These commands originate from the MIHF and are directed to the lower layers.
- **MIH Commands:** The MIH-Users generate these command and send them to the MIHF.

MIH Commands can be delivered locally or remotely. Remote MIH commands are sent by MIH-Users to the MIHF in a peer protocol stack. Remote commands allow the use of network initiated handovers and network assisted handovers in which a network force a terminal to perform handover. Figure 2.5 shows remote MIH commands.

Figure 2.5. Remote MIH Command

In the case of network initiated handover, handover initiation and selection policy function resides on the network. A set of `MIH_Net_HO_***` commands is used in conjunction with any `MIH_N2N_HO_***` commands for initiating handovers. The network can use these commands to:

- Query list of resources currently being used by the MN.
- Reserve resources at a candidate target network by a serving network.

- Command an MN to commit a handover to a specific network.

Figure 2.6 shows the service flow model for a remote command.

Figure 2.6. MIH Command Service Flow

2.3.5.4 Media Independent Information Service

Media independent information service (MIIS) provides a framework by which an MIH enabled entity discovers and obtain information about networks within a geographical area to facilitate network selection and handovers. Simply MIIS can be considered as a database that contains information elements (IEs). These elements provide information essential to perform a successful handover across heterogeneous networks and technologies. Policy engines use static information delivered by MIIS service since they do not require dynamic and updated information. Whereas dynamic information about active networks should be obtained by the use of both MIH event and command services.

2.3.6 MIH Protocol

MIH Protocol defines the format and structure of messages (The MIHF packet with header and payload) that are exchanged between MIHF enabled entities. Figure 2.7 shows the components of the MIH protocol frame.

Figure 2.7. MIH Protocol Frame

The MIH protocol header in figure 2.8 contains information essential for parsing and analysing the MIH protocol frame.

Figure 2.8. MIH Protocol Head

Figure 2.9 shows general TLV encoding used for all parameters in an MIH protocol message.

Figure 2.9. MIH Message Parameter TLV Encoding

MIH protocol does not support any congestion control mechanism. Therefore, it is recommended to run the MIH protocol over congestion aware transport layers such as TCP. Also, Acknowledgement Service shall be used depending on whether the media dependent transport layer is reliable or not. If the media dependent transport is unreliable such as UDP, then the Acknowledgement Service shall be enabled. Whereas if the media dependent transport is reliable, then the Acknowledgement Service is not necessary.

2.4 Context Transfer Protocol (CXTF)

2.4.1 Introduction

CXTF [20] protocol is an experimental protocol proposed to enable authorized context transfers between access routers. Context transfers allow better support for node mobility so that the mobile user can experience minimal disruption. This is achieved by avoiding the re-initiation of signaling to and from the mobile node.

2.4.2 Protocol Overview

This section provides an overview of the CXTF protocol. A context transfer can be either "Mobile Controlled Context Transfer" activated by a request from the mobile node or "Network Controlled Context Transfer" based on internal or network triggers in the previous or the new access router. This protocol operates mainly between a source node and a target node. A context transfer happens when an event, such as handover, takes place. Such an event we call it a Context Transfer Trigger.

2.4.2.1 CXTF Protocol Usage Cases

Performing a context transfer may have multiple advantages described as following:

- Conducting context transfer in advance may increase handover performance and reduce disruption time. To allow this to happen, certain conditions must be met such as acquiring a prior knowledge of the impending handover.
- Performing context transfer after a handover has occurred is a better option than having to re-establish all the contexts at the target node from scratch.

2.4.2.2 Context Transfer Message Format

A CXTP message contains a message-specific header and a number of data blocks. Each message's header has a 3 bit version number field in the first octet, together with the 5 bit field representing the message type. The specification mentioned here applies only to Version 1 of the protocol, and as a result version number field is set to 0x1. Future revisions of the protocol that make incompatible binary changes requires the version number to be incremented.

2.4.2.3 Context Types

Contexts types are identified by the Feature Profile Type (FPT) code, which is a 16 bit unsigned integer. Each context type has a unique meaning determined by a specification document. IANA authority maintains a registry where the context type numbers are tabulated and handled according to the message specifications in this document. Diagram 2.10 illustrates the general format of CXTP message.

Figure 2.10. CXTP Messages General Format

Each context type specification has the following information:

1. Data fields number, size (in bits), and their order in the state variable vector that makes up the context.
2. Default values (if any) for each data field in the context state vector.
3. Different procedures and requirements for recreating a context in a new access router.
4. Status codes for success or failure regarding the context transfer.

2.4.2.4 Context Data Block (CDB)

CDB is used for transferring the content of the context. Table 2.2 explains the meaning of the CDB fields and members.

Field Name	Description
Feature Profile Name	The type of data included in the Data field. It is a 16 bit integer, assigned by IANA.
Length	Message length in unit of eight octet words.
P	0: No presence vector, 1: Presence vector exists.
Reserved	Reserved for future use. Set to zero by the sender.
Data	Context type-dependent data. Data should be aligned on 64 bit; otherwise it is padded with zeros.

Table 2.2. CDB Fields Description

The size of the CDB block is indicated by the length field, including the first four octets starting from FPT.

2.4.2.5 CXTP Messages

In this section, different messages defined by the CXTP protocol are listed and described. The MN is always identified by its previous IP access address during a context transfer operation. Whether IP address is IPv4 or IPv6 is indicated by the 'v' flag.

1. **Context Transfer Activate Request (CTAR) Message:** When an MN requests a context transfer, it sends this message to the new access router (nAR). The message may contain a list of FPTs to transferred. All contexts for the MN are requested if no context types are specified.
2. **Context Transfer Activate Acknowledge (CTAA) Message:** It is used as an acknowledgement message sent by the receiver of CTAR to the MN. This message might include a list of FPTs that were not successfully transferred.
3. **Context Transfer Data (CTD) Message:** This message is used to carry actual feature data (CXTP data), and it is sent by the previous access router (pAR) to the nAR. This message can handle both predictive and normal context transfer. The acknowledgement flag 'A' indicates whether pAR requires a reply. Figure 2.11 shows the structure of the normal CTD message. Algorithm, key and key length fields are used in the case of predictive context transfer only.
4. **Context Transfer Data Reply (CTDR) Message:** When an acknowledgement is required in the CTD message, the nAR sends this message to the pAR, indicating success or failure. The content of this message is shown in figure 2.12.
5. **Context Transfer Cancel (CTC) Message:** If context transfer (CT) cannot be completed with a specific amount of time, then the nAR may send this message to the pAR to abort an ongoing CT operation.
6. **Context Transfer Request (CT-Req) Message:** Upon the reception of a CTAR message, the nAR sends this message to the pAR to request the start of CT operation.

Figure 2.11. Context Transfer Data (CTD) Message

Figure 2.12. Context Transfer Data Reply (CTDR) Message

2.5 3GPP 4G-LTE Network

LTE stands for Long-Term Evolution, commonly marketed as Fourth-Generation mobile communications system and it is the latest steps in an advancing series of mobile telecommunications systems. LTE describes the evolution in both the wireless communication part and the network architecture part. It has been designed to support packet switch services only compared to previous legacy systems that support circuit switched, and packet switched services. LTE features the following capabilities compared to the previous mobile systems like GSM & UMTS [21]:

- Higher user data rates, reduced latency, and support of quality of service.

- Lower cost per transmitted bit thus improving spectrum utilization.
- More flexible mobile terminals with reduced energy consumption.
- Seamless connections to different networks with different access technologies.
- Simplified network architecture, reduced system cost, flexible spectrum allocation and interoperability with legacy systems.

2.5.1 Network Architecture

LTE refers to the evolution of the radio access part through the Evolved-UTRAN, and it is accompanied by the evolution of the non-radio aspects under the term 'System Architecture Evolution' (SAE) which contains the Evolved Packet Core (EPC). Together LTE and SAE form the Evolved Packet System (EPS). EPS comprises multiple functional, logical entities without particular implementation about the physical entities. The figure 2.13 shows the overall network architecture including the network entities and the standardized interfaces.

Figure 2.13. LTE Network Architecture

The overall network architecture can be split into two parts the radio access part (E-UTRAN) and the core network part (EPC). The EPC is made up of multiple logical nodes, whereas the E-UTRAN contains only one node that is the evolved NodeB (eNodeB). These nodes are interconnected with each other through standardized interfaces for interoperability purposes that allow operators to deploy multi-vendor products in their networks 2.13.

2.5.1.1 The Core Network (CN)

The core network in the EPS is fully IP-Based network that provides packet switched services only. Circuit switched services (CS) shall be provided by the IP-Multimedia Subsystem (IMS). Due to the reduced number of elements, the EPC provides lower latency, and delay compared to the previous systems. The main logical nodes of the core network:

- **Packet Data Network Gateway (P-GW):** It provides UE with connectivity to external data networks by serving as an exit point for the traffic generated by the UE and an entry point for the traffic coming from the external networks. It also serves as a mobility anchor for inter-working with non-3GPP technologies such as CDMA2000 and WiMax. P-GW servers and

policy enforcement rule (PCEF) for the policies originated by PCRF entity, it also provides packet filtering, content charging, deep packet inspection (DPI), IP address assignment and lawful interception.

- **Serving Gateway (S-GW):** It is considered as an entry point to the CN for packet routing and forwarding. It serves as local mobility anchor for data bearers when UE moves between different eNodeBs and as mobility anchor for inter-working with other 3GPP technologies such as GPRS and UMTS.
- **Mobility Management Entity (MME):** MME is the control node in the CN, which is responsible for mobility management in the control plane. Mobility management includes management of user contexts and mobile status including dedicated bearer establishment, tracking area list management, PDN Gateway (P-GW) and Serving Gateway (S-GW) selection, lawful interception of signaling traffic, assignment of temporary identifiers, and Non-access stratum (NAS) signaling.
- **Policy Control and Charging Rule Function (PCRF):** PCRF is a new network core entity introduced for next-generation networks to determine and deliver policy rules to PCEF node. These rules allow mobile operators to optimize and control their networks in real-time by offering multiple QoS levels, and charging rules to the end users in a dynamic fashion based on subscription information and usage statistics.
- **Home Subscriber Server (HSS):** The HSS serves as a database for the LTE network. It contains users' subscription data such as QoS profile and roaming access restrictions. Also, it holds information about the PDNs to which the user can connect in the form of an Access Point Name (APN). In addition, the HSS obtains dynamic data like the id of the serving. The Authentication Center (AuC) may be integrated as part of HSS; the AuC is responsible for generating the vectors for authentication and security keys.

2.5.1.2 The Access Network

The access network of the LTE (E-UTRAN) contains a network of eNodeBs interconnected with each other through a standardized interface named "X2 Interface", and to the CN by the means of "S1 Interface". There is no central entity in the access network so that the architecture can be considered as flat. Access Stratum (AS) protocols are the protocols which run between the eNodeBs and the UE. The E-UTRAN is an entirely new air interface system based on Orthogonal Frequency Division Multiple Access (OFDMA) in the downlink and single carrier FDMA (SC-FDMA) in the uplink. This interface provides higher data rates and lower latency and is optimized for packet data. Also, it is responsible for all radio-related functions such as radio resource management, header compression, security, and connectivity to the EPC.

2.5.2 Mobility in LTE

LTE/SAE supports user mobility within LTE/SAE, and also mobility to other legacy systems using both 3GPP specified and non-3GPP technologies. Compared to previous mobile generations, LTE does support hard handover because of this a centralized data-combining function is not required anymore. As a result, data buffering is done in the eNodeB itself. Handover in LTE can be characterized as following:

- **Network-Controlled:** Selection of the target cell is done by the network in the E-UTRAN, not by the UE.

- **UE-Assisted:** Radio signal measurements are made and frequently reported by the UE to the network.

Figure 2.14 shows signaling flow for mobility over X2 interface.

Figure 2.14. Mobility over X2 Interface

The handover procedure can be subdivided into three phases [22, 23]:

- **Handover Preparation Phase:** In this phase, the thresholds for transmitting measurement reports are set by the eNodeB. The Radio Resource Control (RRC) in the Source eNodeB uses the latest measurement to decide whether a handover is required to another cell. This is done by checking if the neighbouring cell signal quality is better than the current serving cell. If the handover is required, the target cell is selected, and the eNodeB for the target

cell is identified. Then, the Source eNodeB initiates the handover with the Handover Request message. Later, the target eNodeB sends a Handover Request Acknowledge message back to the source eNodeB. This message contains the Handover Command message (RRC Connection Reconfiguration Request) in a transparent container. Finally, an X2 GTP-U tunnel is created between the Source and the Target eNodeBs. This tunnel carries the user data during the handover.

- **Handover Execution Phase:** This phase starts upon the reception of the handover command sent by the UE sent by the Source eNodeB. Then, the target eNodeB sends the Packet Data Convergence Protocol (PDCP) sequence numbers to the source eNodeB. Additionally, all data being buffered at the source eNodeB is forwarded to the target eNodeB. At this point, the UE is capable of receiving and transmitting data. Packets destined towards the UE is still being routed across the source eNodeB. In the next phase, the data transmission path will be updated to omit the source eNodeB from it.
- **Handover Completion Phase:** During this final phase, the target eNodeB sends a request to the MME to switch the data path from the source eNodeB to the target eNodeB. Then, the MME requests the SGW to change the path to the target eNodeB. Finally, target eNodeB receives the end marker and requests the source eNodeB to release all the resources related to the UE.

Chapter 3

Design

This chapter contains the requirements and elements involved in the design of the service mobility system. It starts with a general design block for implementing a service mobility system. Then, an overview about the requirements and the selection criteria for each block is presented. Later, a detailed framework architecture for implementing a prototype of the service mobility is proposed. Finally, details regarding the migration signaling flow and the application programming interfaces (APIs) are presented.

3.1 Service Mobility in Deployment

Figure 3.1 displays the position of the service mobility system in a cloud platform. Service mobility system is designed to be a part of cloud deployment's Platform as a Service (PaaS) offering. Services that are deployed on cloud platforms that offer service mobility system can utilize its functionality through a set of API calls.

Figure 3.1. Service Mobility In Deployment

3.2 System General Design Requirements

1. To facilitate the real deployment of the service mobility system; the system should make use of the current existing set of standardized protocols. In this way, we do not have to go through a standardization process due to the design and implementation of proprietary protocols, thus producing a vendor neutral solution.
2. The migration time calculated as the time difference between the moment a Layer 2 handover trigger is received and a confirmation message of a successful context migration operation is obtained, should be as small as possible. This requires optimization in terms of signaling messages used to initiate the context migration process.

3.3 General Design Framework

Figure 3.2 shows different blocks involved in the development of service mobility system. The system is made up of five blocks. Each block has a specific set of functions to perform. A common controller is required to allow interaction and communication between these blocks.

Figure 3.2. Service Mobility General Design Framework Diagram

3.3.1 Handover Signaling System Block

This block is mainly responsible for initiating UE context migration signaling. The requirements for building this block can be summarized in the following points:

- A topology database to maintain information about cloud deployments and contents availability.
- A handover protocol to deliver information regarding the migration procedure in a standardised way. The information should include the identity of the mobile node, the type of the handover action, and the destination to where the mobile node will be handed over.

- A handover-decision making mechanism which allows eNodeB-Cloud to trigger a UE context migration process.

The previous requirements are grouped together in one block as shown in figure 3.3.

Figure 3.3. Handover System Design Block

3.3.2 Network Session Migration Block

The block of network session migration shown in 3.4 is an essential part of service mobility system. It is responsible for obtaining various states and data related to the network session (Socket) to be migrated during the export phase. Whereas in the import phase, it recreates all states and contents of different queues of the migrated network sessions in the importing server. The network session migration mechanism should be carried out in a transparent and seamless way from a UE's perspective. In details this means:

Figure 3.4. Network Session Migration Block

- The client's TCP/IP protocol stack should remain the same without requiring any modifications to its functionality.
- The modification of the connection ownership should have a minimum impact on the perceived user's quality.

3.3.3 Context Transfer Block

This block is responsible for migrating UE context between eNBCs involved in a context migration. The block has three main tasks shown in 3.5 and summarized as following:

- Migration data message Construction: Contexts identifying user's active services and their network sessions are collected to construct a migration data message. Each context in the message should be identified in a unique way, so there is no place for ambiguity in the receiver side.
- Migration data extraction: Contexts encapsulated in the received migration message should be extracted and delivered to their corresponding services.
- Migration data message serialization and deserialization using a standardized frame format.

Figure 3.5. Context Transfer Block

3.3.4 Migration Signaling Flow Block

The migration signaling flow block is responsible for defining the sequence of signals required to migrate contexts between eNodeB-Clouds. While defining the sequence of the signals, the following requirements should be kept in mind:

- The signaling flow should deliver information regarding the role of each eNBC involved in a service mobility.
- Information regarding the status of the migration operation should be delivered to all parties.

3.3.5 Service Mobility APIs (PaaS Interaction) Block

This block provides an abstracted interface of the service mobility system through a set of API calls. Following are the requirements from the API interface:

- This layer should be agnostic to the underlying technologies used by the different blocks.
- It should be easy and straightforward for the service developer to interact with service mobility system.

3.4 Proposed Framework Architecture

Figure 3.6 shows the chosen protocols and systems to build up the service mobility system.

Figure 3.6. Service Mobility Framework Proposal

For the handover signaling system block, the IEEE 802.21 MIH is chosen to assist in context migration signaling. The MIH is selected because it has the following capabilities:

- It provides a framework to carry out a network to network handover. So it can be used by an eNBC to trigger a peer eNBC to prepare for incoming context migration. This task is provided by the Command Service (MICS) in the MIHF.
- It contains a database with information regarding various nodes in the network topology. In this way, the source eNBC will be capable of determining to which eNBC the UE context should be migrated to. The Information Service (MIIS) provides this capability.
- It defines a standardized frame format for handover signaling.

The CXTP protocol is used to transmit UE context data. The CXTP provides a set of standardized messages to exchange contexts between nodes. In addition, it allows service developers to define the meaning and specifications of their service context through Feature Profile Type (FPT) codes. Finally, for the signaling flow block a combination of CXTP and MIH protocols message are selected to carry out a UE context migration.

To start building up the system, different blocks should be encapsulated inside user-level processes. Figure 3.7 shows the framework architecture for implementing the proposed design. In this architecture, we can distinguish different types of interfaces as follow:

1. Standard interface between MIHF and MIH-User as defined in the IEEE 802.21 specification.
2. Interface between local and remote MIHF entities as defined in the IEEE 802.21 specification.

Figure 3.7. Service Mobility Framework Architecture

3. API calls for PaaS Interaction.
4. Interface to interact with service mobility process.
5. CXTP Protocol interface defined by RFC4076.

3.4.1 Service Mobility Process

In the proposed architecture, the whole migration process is driven by the MIH-User. It works as a hook for the handover signals generated by the LTE-eNodeB. Once it catches a handover imminent signal in the layer 2 of the radio interface, it starts up a UE context migration process. An alternative triggering mechanism could be by using technology agnostic interface between LTE/SAE (Link Layer) and MIHF as defined in the IEEE 802.21 specification and shown in figure 3.8. In this design, the MIHF entity includes the MIH_LINK_SAP, that works as a driver between the MIHF and the eNodeB. MIH_LINK_SAP performs various messages mapping and translation between LTE-eNodeB handover signals and MIHF handover signals. In addition, it drives and initiates the migration process.

In the network initiated handover procedure defined in the IEEE 802.21, in which the network initiates the handover process by indicating to the mobile node that a handover is necessary. Only

Figure 3.8. MIH LINK SAP as a Driver

the MIH-User and the MIHF entities are engaged and involved in the handover operation. This exactly matches our use case, since the handover decision is made and initiated by the LTE eNodeB. As a result, we do not have to include the MIH_LINK_SAP part in the framework. By excluding the MIH_LINK_SAP part, we optimize the number of internal calls. Additional to that the time required to carry migration process is reduced, since we omit two periods; the period required to perform translation between 4G-LTE handover messages and MIH_LINK_SAP handover indications, and the period spent by the local MIHF to map the MIH_LINK_SAP primitives into the corresponding MIH_SAP primitives. Because of this, we extend the original scope of IEEE 802.21 MIH framework from assisting in handover across heterogeneous wireless access networks, to support handover signaling between eNodeB-Clouds to migrate a UE context.

To transfer contexts related to the services with their corresponding network sessions, the Context Transfer protocol (CXTF) is employed. As mentioned before in the *Chapter 2*, CXTF is an experimental protocol proposed by the IETF for transmitting applications' contexts. To our knowledge, the protocol has not been adopted or tested in any system.

3.4.2 Migration Signaling Flow

To start up and drive a UE context migration process, we need to define a signaling flow. This flow informs different elements in the network about their roles and how to react upon the reception of a particular signaling message. Figure 3.9 displays the proposed migration signaling flow. In the later paragraphs, the following two terms will frequently be used:

- **Serialization:** It refers to the process of sending the data as a sequence of bits using particular encoding scheme.
- **Deserialization:** It refers to the process of extracting the data as a sequence of bits using particular decoding scheme.

1. The migration process starts when the source eNodeB receives a Handover Request ACK on the X2/S1-C interface. This message is used to trigger the MIH-User part in the eNBC to create a new source state machine.
2. Then, the MIH-User serializes an MIH_N2N_HO_COMMIT Request message and sends it to its local MIHF. The MIHF in turn sends this message to the destination eNBC where the user will be handed over and resume his services.
3. In the target eNBC, upon the reception of the MIH_N2N_HO_COMMIT Request message, the local MIHF sends an MIH_N2N_HO_COMMIT Indication message to its local MIH-User. As a result, a new destination state machine is created to this particular context migration.
4. Later, the MIH-User serializes an MIH_N2N_HO_COMMIT Response message and sends it to its local MIHF. The MIHF in turn sends this message to the source eNBC indicating whether the user can process with context migration or not.
5. Upon the reception of MIH_N2N_HO_COMMIT Response message in the source eNBC, the MIHF entity generates an MIH_N2N_HO_COMMIT Confirm primitives and transmits it to its local MIH-User. Depending on the value of the Status TLV field in the confirm message, the MIH-User decides whether to continue with the context migration or aborts it. If the value is Success, the source state machine continues its transaction; otherwise it proceeds to the Failure State and stops the migration process.
6. At this point the migration is considered to be possible. So the source eNBC starts serializing a Context Transfer Data (CTD) message containing all services contexts and network sessions contexts, to allow the user to resume his services at the destination eNBC.
7. Finally, the destination eNBC uses the data in the received CTD message to import contexts related to different services and import their network sessions. If everything goes smooth, a Context Data Reply (CTDR) message is constructed and sent back to the source eNBC confirming a successful migration process.

Figure 3.9. Migration Signaling Flow

3.4.2.1 MIH Signaling Messages

The MIH_N2N_HO_Commit message with its different operation codes is chosen to support the migration process. This is because N2N stands for the Network to Network communication and Commit implies that a handover operation is going to take place. Below is a list and description of different operations codes used in our handover signaling flow:

1. **MIH_N2N_HO_Commit Request:** This primitive is used by an MIH-User on the serving network to inform a selected target network that the mobile node (MN) is about to move to the target network. In our case, it is used by the source eNBC to inform the destination eNBC about incoming context migration.
2. **MIH_N2N_HO_Commit Indication:** This primitive is used by an MIHF to indicate that an MIH_N2N_HO_Commit Request message has been received from a peer MIHF on the serving network. For our case, this message is considered as an internal call in the destination eNBC to inform the MIH-User module about incoming context migration.
3. **MIH_N2N_HO_Commit Response:** This primitive is used by an MIH user to respond to an MIH_N2N_HO_Commit Indication primitive. In our case, it is used to inform the source eNBC if it can proceed with its migration process or not.
4. **MIH_N2N_HO_Commit Confirm:** This primitive is used by an MIHF to confirm that an MIH_N2N_HO_Commit Response message is received from a peer MIHF on the selected target network. This message is used as an internal call in the source eNBC to inform the MIH-User entity about the answer of the target eNBC regarding the ability to proceed with the context migration.

Most of the TLV fields of the previous primitives are irrelevant to our use case of the MIH protocol. However, they have to be included in the MIH message to adhere to the MIH protocol standard; hence, their values are set to zero. We use the IP Address as a unique identifier to identify a mobile node (UE) when a context migration process takes place. For this purpose, we need to find a TLV field within the MIH protocol that allows us to save the IP address. In the specifications of the MIH protocol, we have the "Mobile node MIHF ID" TLV, which is derived from an OCTET_STRING data type. This data type permits us to store any string value conditioning that the maximum length of the string is less than 253 bytes. Therefore, we will use this field to store the IP address of the client in a text mode which will result in extra bytes instead of using the binary mode.

3.4.2.2 CXTP Messages

In the original scope of the CXTP protocol, the MN is aware and involved in the context transfer procedure. However for our case, the migration procedure should be completely transparent to the end user and does not require any modifications to the client device. Because of this, only two types of CXTP messages will be used to carry the context migration:

1. **Context Transfer Data (CTD) Message:** This message is sent by the source eNBC upon the reception of N2N_HO_COMMIT_Response with a success status. It carries contexts related to the services and network sessions. As the IEEE 802.21 MIH Protocol, we set zero values for the fields that are irrelevant to our use case. Specifically for this message, the "Elapsed time" field will not be used at all and shall be set to zero.
2. **Context Transfer Data Reply (CTDR) Message:** When a CTD message is received by the target eNBC, and after extracting various contexts encapsulated in it, the target eNBC

constructs a CTDR message and sends back to the source eNBC. This message confirms a successful migration of all the contexts sent by the source eNBC. In other words, the reception of this message concludes a context migration process.

3.4.3 Service Mobility Interface APIs

To allow various services to interact with the service mobility framework, a set of API calls are defined and encapsulated as a separate standalone library. Figure 3.10 shows the proposed APIs which assists cloud services to UE context migration.

Figure 3.10. Service Mobility Interface APIs

1. **Register Service API:** During the start-up or the initiation phase of the service, this API must be called to register a callback with the Service Mobility system. This API takes two input parameters; the first one is the Service ID, and the second one is a pointer to the callback function to be registered with the Service Mobility system. Service ID should uniquely identify a service across different systems, for this purpose we use GUID (Globally Unique Identifier) which is a 128 bit reference number used in software construction to globally identify application. On the other hand, the Callback function should take three types of parameters:

- a) The type of the event for which the callback function has been called. We define six events as follow:
 - i. **PREPARE_TO_EXPORT Event:** This event is signaled to a particular set of services, when a context migration is in action. It is used to inform these services to prepare their contexts and suspend read/writ operations related to the network sessions associated to the UE. At the end, each service should call Export Session API.
 - ii. **PREPARE_TO_IMPORT Event:** Upon the complete reception of all migration data from the source eNBC, the service mobility system triggers this event and sends it to each service listed in the migration data. This event carries the corresponding context data for each service. When this event is received by the service, it installs the context data associated with this event and creates a number of empty network sessions to be used for importing the migrated network sessions. At the end, each service should call Import Session API.

- iii. **EXPORT_COMPLETED Event:** When the UE context migration process has been completed successfully, the service mobility system signals this event to all services used by the UE to inform them that migration process has been completed. As a result, each service can remove the user's context data from its internal table and deregister him from the service mobility system.
 - iv. **IMPORT_COMPLETED Event:** This event is sent individually to each service to inform it that network sessions have been imported successfully by the Kernel Module, and the service can resume its execution.
 - v. **EXPORT_FAILED Event:** In the case of migration failure, this event is sent to all user's services in the source eNBC carrying the cause of failure. Depending on the reason, it is the responsibility of the services to take appropriate actions.
 - vi. **IMPORT_FAILED Event:** This event has similar meaning to the previous one, except it is used to propagate the reason of failure to the services in the destination eNBC.
- b) Transaction Identifier (Transaction ID) is a unique identifier that is used to match an event generated by the underlying service mobility system with any operation to be executed for that particular event. Two transactions IDs are generated for the each context migration process, one for the source eNBC and the other for the destination eNBC.
- c) The data associated with the event, the content of this data depends directly on the event's type. The following list illustrates the information associated with each event:
- **PREPARE_TO_EXPORT Event Data:** It contains only one variable, which is the IP address of the mobile user.
 - **PREPARE_TO_IMPORT Event Data:** The data structure here includes three data members namely; The IP address of the mobile user, the number of connections associated with that user, and the corresponding service context data.
 - **EXPORT_FAILED Event & IMPORT_FAILED Event Data:** It indicates the reason of the failure.
- Regarding both EXPORT_COMPLETED and IMPORT_COMPLETED Events, no information shall be associated with them, since they serve as an indication to a successful operation.
2. **Register User API:** Whenever a user requests a service hosted on the eNodeB-Cloud, the service registers this user with the service mobility system. This will ease the task of the service mobility process since it creates a table and stores in it a list of all services being used by each user. So when a user wants to move to a different eNodeB, the service mobility triggers only the active services for this user and informs them to prepare their contexts. In this case, we mitigate the extra processing required by each service to figure out if the migrating UE is being served by it or not. This API takes only one parameter that is the IP address of the client.
 3. **Deregister User API:** In contrast to Register User API, this API tells the service mobility to remove a user from its internal table. It is called, whenever a user closes all the network sessions with this service.
 4. **Export Context API:** This API is called by a service when it receives PREPARE_TO_EXPORT event. By calling this API, a service informs the service mobility system about the network sessions being in use by the mobile user. This API takes four parameters as follow:
 - a) TransactionID: Its value should be similar to the one associated with the PREPARE_TO_EXPORT event.

- b) **ProcessID**: A unique process ID identifying the calling service in the system.
 - c) **FDList**: A list of socket file descriptor (network session ID) to be exported.
 - d) **ContextData**: Service context data related to the migrated user. It contains necessary and minimum amount of information required to re-establish execution at the peer eNBC.
5. **Import Context API**: Upon the reception of `PREPARE_TO_IMPORT` event, this API is called by the service. Calling this API should be done after the service has created a certain number of empty network sessions. The following four parameters are required to call this API:
- a) **TransactionID**: Its value should be similar to the one associated with the `PREPARE_TO_IMPORT` event.
 - b) **ProcessID**: The process ID of the calling service.
 - c) **FDList**: A list of the newly created socket file descriptor.

Chapter 4

Implementation

This chapter includes description and explanation of the implementation stage for the design proposed in *Chapter 3*.

4.1 State Machines

State machine is used to represent a set of complex rules and conditions in a very compact way, and to process different inputs. A state machine can be described in a tiny amount of code compared to its equivalent procedure and runs in an efficient way. Moreover, both the rules and the conditions that govern changes between states can often be stored in a table, providing a compact way to represent a machine.

A UE context migration process can be considered as a sequence of states, where each state defines sets of actions to be performed during that state. We use a state machine to represent a migration process which can exist in a limited number of conditions "states". Transition from one state to the next according happens according to a fixed set of rules.

In the service mobility framework, each migration process is represented by two transactions that correspond to two instances of the source and the destination state machines. An eNodeB-Cloud that has to start the migration process is called source transaction, and it starts the source state machine. In the same manner, an eNodeB-Cloud that receives a handover signaling message related to a new migration is called destination transaction, and it initiates the destination state machine. Each transaction has a set of variables, constants, and procedures shown in table 4.1 that can only be accessed within a single state machine for that given transaction.

Once the transaction is created, it will be added directly to the transaction pool as shown in figure 4.1. This pool has a watchdog timer with a resolution of one second, this means that the initial values used to set the timers inside the state machines are integer values. The pool timer is responsible for decrementing the timer variable for each transaction in its list each second, in response to an external system clock function. After decrementing the timer variable, if the value of the lifetime variable is equal to zero, the transaction will be added to the delete pool. For each transaction in the delete pool, some values of the internal variables will be written to a log file. And eventually the transaction will be removed from the delete pool.

Figure 4.1. Transaction Pool

Name	Description
TransactionStatus	Indicates the status of the transaction. This variable is written by the state machine and read by the MIHF. The following values are valid: <ul style="list-style-type: none"> • ONGOING. • FAILURE. • SUCCESS.
TID	Transaction Identifier.
TransactionLifeTime	The remaining lifetime of the transaction.

Table 4.1. State Machine Variables

4.1.1 Source State Machine

This state machine is created and started, when an X2/S1-C Handover Request ACK sent by the target eNodeB is captured by the Mobility Hook Module. An instance of source state machine can cease to exist once the value of TransactionStatus is set to either SUCCESS or FAILURE. Figure 4.2 illustrates different states this state machine can exist in:

- **Initial State:** One of the states must be defined as the initial state in which the state machine starts in when it is turned on. In this state, the state machine initializes its internal variables and starts a timer to measure the amount of time spent in each of the upcoming states.
- **Handover Preparation State:** The next state after initializing the state machine is the Handover Preparation State. In this state, the MIH software module serializes MIH_N2N_HO_Commit request message and sends it to the peer eNodeB-Cloud.
- **Wait HO Commit Confirm State:** After transmitting the MIH_N2N_HO_Commit Request message, the MIH module expects a confirmation for this message. For this reason, the state machine enters a new state waiting for the MIH_N2N_HO_Commit Confirm message with a wait timer set for a specific value. If the confirm message is received before the timer expires, the state machine proceeds to the "Context Migration State"; otherwise it proceeds to the "Failure State" and aborts the migration.

Figure 4.2. Source State Machine

- **Context Migration State:** In this state, the source eNBC signals all the services used by the UE to prepare their contexts and provides a list of file descriptor IDs for the network sessions in use by that particular UE. Then, the session migration module starts retrieving states and information associated to each network session in the list. Later, a CXTDP CTD message with contexts related to the services and network sessions is serialized and sent to the destination eNBC. If the transmission of the data succeeds, the state machine moves to the "Wait CTDR State", otherwise it continues to the "Failure State".
- **Wait Context Transfer Data Reply State:** During this state, the state machine starts a timer waiting for a CTDR message to be received. If the message is received before the timer expires with 'S' flag set to one, the migration has succeeded, and the state machine moves to the "Success State"; otherwise it moves the "Failure State".
- **Failure State:** When the state machine enters this state, we get an indication that the migration process has failed for some reason. As a result, the TransactionStatus variable is set to FAILURE, the state machine's timer is stopped, and all services are signaled with EXPORT_FAILED to take appropriate actions.
- **Success State:** Eventually, if the CTDR message is received with 'S' flag set to one, the transaction enters this state as proof of a successful context migration process. A sequence of operations is performed including signaling user's services with EXPORT_COMPLETED, setting the TransactionStatus variable to SUCCESS, and stopping the state machine's timer.

4.1.2 Destination State Machine

The destination state machine is started when a handover signaling message related to a new context migration is received. The state machine terminates when it transits to the FAILURE state or SUCCESS state. An instance of destination state machine can cease to exist once the value of TransactionStatus is set to either SUCCESS or FAILURE. Figure 4.3 displays different states a in which destination state machine can exist in.

Figure 4.3. Destination State Machine

- **Initial State:** Upon the reception of MIH_N2N_HO_Commit request message, a new state machine is created in the destination eNBC. The state machine starts with this state in which it prepares and initials all its internal variables, like setting the transaction status variable to ON_GOING. In addition, it starts a timer to measure the amount of time spent in each of the following states.
- **HO Preparation State:** During this state, the MIH-User begins serializing an MIH_N2N_HO_Commit Response message with a status TLV indicating whether the context migration is allowed or not. Then, the MIH-User sends this message to its local MIHF which in turns sends it back to the source eNBC. If the migration is allowed, the state machine moves to "Wait CTD State". Otherwise, it proceeds to "Failure State" and the ongoing migration process is halted.
- **Wait Context Transfer Data State:** After transmitting the MIH_N2N_HO_Commit Response message, the state machine waits for the Context Data Transfer (CTD) message from

the source eNBC with a wait timer set for a specific value. If the timer expires before the reception of the CTD message, the state machine moves to the "Failure State"; otherwise the received CTD message is deserialized, and migrated UE contexts are extracted.

- **Import Sockets State:** In this state, different number of services are signaled to import the migrated contexts. Each service while importing its context, it creates a certain number of empty TCP sockets without binding them to any specific ports or IP addresses. Then, a list of the corresponding file descriptors of these sockets is passed to the state machine. Finally, the state machine passes this list to the session migration module to import the migrated sockets and copy their states and queues contents to these empty sockets.
- **Prepare Context Data Transfer Reply State:** If everything goes smoothly in the "Import Sockets State", a Context Data Transfer Reply (CTDR) message is serialized with the 'S' flag set to one. This flag indicates whether all feature contexts sent in the CXTP CTD were received successfully or not. Finally, the constructed CTDR message is sent back to the source eNBC.
- **Failure State:** When the state machine enters this state, we get an indication that the migration process has failed for some reason. As a result, the TransactionStatus variable is set to FAILURE, the state machine's timer is stopped, and all service are signaled with IMPORT_FAILED to take appropriate actions.
- **Success State:** When the CXTP CTDR message is sent successfully, the state machine enters this state as proof of a successful migration process. In this state, the TransactionStatus variable is set to SUCCESS, and the state machine's timer is stopped too.

4.2 Software Modules

To ease the development of the service mobility framework, we subdivide it into multiple software where each module contains necessary functions to execute only one aspect of the desired operation. The figure 4.4 shows various software modules required in the implementation.

Figure 4.4. Service Mobility Software Modules

4.2.1 Session Migration Module

Based on the background study provided in Chapter 2 regarding different session migration solutions, we propose the use of SockMi implementation due to the properties it provides compared to the other methods. The original implementation of the SockMi contained three different parts as follow:

- **The SockMi Kernel Module:** It is the engine of the TCP/IP socket migration mechanism. The module is responsible for the following tasks:
 1. Save/restore the state of the migrating TCP sockets during the export/import phase.
 2. Transfer information regarding migrating sockets with the SockMi daemon.
 3. Provide low-level primitives to activate and control the socket migration facility.

Additional to the state of the socket, SockMi migrates in-flight data. The content of these data is the accumulation of the following queues [24]:

1. **Receive queue:** If the system receives packets and the application does not read these packets, they will be stored in this queue.
 2. **Transmit queue:** Packets to be sent over the wire, or packets already sent but not yet acknowledged are stored in this queue.
 3. **Out of order queue:** It is used as temporary storage for segments arriving out of order.
- **The SockMi Daemon (SockMiD):** It is a user-space process which works in combination with the SockMi Kernel module to support the socket migration mechanism. The daemon tasks can be summarized as follow:
 1. In the export phase, it reads exporting sockets states from the internal buffers of the SockMi Kernel module.
 2. During the negotiation phase, it searches for other hosts running SockMi daemons to choose where to migrate the sockets.
 3. Finally in the import phase, it writes the states and queues contents of the imported sockets to the internal buffers of the SockMi Kernel module.
 - **The SockMi Application Programming Interface:** It allows applications to activate and enable the socket migration mechanism.

The following changes have been made to the original implementation of SockMi to match the design requirements for the service mobility framework:

1. Both SockMi Daemon and SockMi API library are integrated together as one single software module inside the service mobility system named as "Session Migration Module". This module provides the necessary calls to start exporting/importing sockets, which is used by the state machines during a migration process.
2. The SockMi Daemon part will not be in charge of transmitting of the socket data state and communicating with other SockMi daemons. This task will be handled by a different software module called "CXTP Module".
3. The original implementation of the SockMi Kernel module considered the following two cases for IP packet redirection:

- a) The host that exports the socket can give up to its IP address in favour of the host that imports the socket.
- b) Both the exporting and the importing hosts maintain their original IP addresses because both keep exchanging data on the network after the socket migration. This results in usage of a special combination of Network Address Translation (NAT) operations.

In the proposed deployment scenario for the service mobility framework, each eNodeB-Cloud runs and manages its local service network. As a result, the mobile network will be made up of isolated service networks, resulting in the usage of the same IP addresses prefixes and ranges. So since both the importing and the exporting servers poses the same IP address, and when a mobile user moves towards a new eNodeB, it loses the physical connection with the source eNodeB. The framework implementation will make use of the first case of the IP packet redirection provided by the SockMi Kernel module and avoid using NAT functionality.

4. Negotiation phase is not necessary anymore, since the target eNodeB-Cloud is already known.
5. The service mobility framework will be responsible for the actual export/import of the TCP/IP sessions on the behalf of the running application and services.

Due to the rapid developments and quick changes in the Linux Kernel, the current version of the SockMi Kernel module which targeted Linux systems with Kernel's version of 2.6.11 cannot be compiled or run anymore on the new Linux Kernel versions. Some changes to the procedures and data members used to access and manipulate internal TCP socket states in the Kernel space, had to be done to compile with the latest data structure definitions in the Linux Kernel TCP implementation. In addition to that, some changes and error corrections had to be completed to let the SockMi Kernel module works perfectly and flawlessly. These modifications can be summarized as follow:

- While migrating a socket, the exporting server should preserve different socket's queues in a consistent state. This implies that no more segments should be removed or added to these queues during the migration phase. In the original implementation of SockMi, the contents of these queues were copied and re-created at the importing side inside the Linux Kernel by using `sk_buff` data structure and functions. Linux Kernel has gone through many changes, and the way a packet is created and stored in a `sk_buff` has changed dramatically. It was hard to copy and re-install the `sk_buffs` located at the exporting side in the importing side. For this reason, the payload part of the segments located in the write queue "The data part of the `sk_buffs`" were copied as one data block. Whereas the TCP segments stored in both receive queue and out-of-order queue were ignored, since the client will resend them again if he does not receive any acknowledgement frame in a certain amount of time. In the import side, the service mobility passes the write data block to the service through Shared memory mechanism when it signals the service with `IMPOR_COMPLETED` event. The Service Mobility Interface layer inside the service writes this data to the TCP socket in a transparent way to the service. This is done before the service starts resuming writing its data from the place it left off. Figure 4.5 illustrates the conversion of `sk_buffs` ring into one data block.
- Linux uses a global counter named `tcp_time_stamp` to generate local time stamps. If a migrated connection uses the timestamp value at the new host, the local round-trip-time (RTT) calculation may be confused when receiving timestamp replies from the previous connection. As a result, the peer's Protect Against Wrapped Sequence numbers (PAWS) algorithm will discard segments if their timestamps appear to have jumped back in time [10]. For this issue, SockMi solution tries to synchronize the local timestamp generator with the one used in the exporting side. This is completed by introducing a per-connection timestamp offset that is added

Figure 4.5. sk_buffs Ring

to the value of `tcp_time_stamp` for each of the outgoing packets only. This solves only half of the problem, because when a client receives a segment from the importing host, it will echo back the value of the timestamp it received. This echoed timestamp value should be modified and restored to the original local timestamp value without any offset. The complete solution for the timestamp problem will be solved using the netfilter framework in the following way:

1. When we import the TCP options of a migrating connection as shown in figure 4.6, we perform the following steps:

Figure 4.6. Saving TimeStamp values

- a) The last sent TCP segment timestamp value is saved in a variable named *imported_ts*.
 - b) The current local timestamp value in the importing side is saved in a variable called *base_ts*.
2. Then, we install a post-hook filter for the outgoing packets as shown in figure 4.7. This hook modifies the timestamp value for each packet that belongs to the migrated TCP connection and leaving the server. The new timestamp value is calculated as following:

$$\text{Correct TimeStamp Value} = \text{Segment TimeStamp} - \text{Base TimeStamp (base_ts)} + \text{Exporting Side Last TimeStamp (imported_ts)}$$

Figure 4.7. Post-hook Filter for Migrated Socket

3. Finally, we install a pre-hook filter for the incoming packets as shown in figure 4.8. this hook modifies the echoed timestamp value for each packet entering the server and belongs to the migrated TCP session. The echoed timestamp value is calculated as following:

Figure 4.8. Pre-hook Filter for Migrated Socket

$$\text{Correct Echoed TimeStamp Value} = \text{Segment Echoed TimeStamp} - \text{Exporting Side Last TimeStamp} (imported_ts) + \text{Base TimeStamp} (base_ts)$$

4.2.2 MIH User Module

This module represents the MIH-User part in the IEEE 802.21 standard. It is responsible for triggering UE context migration based on the handover signals intercepted on eNodeB X2/S1-C interface. Two open sources implementations of the IEEE MIH 802.21 have been identified:

Open-Source MIH: It is an open source implementation of IEEE 802.21 standard. It provides an extensible implementation for experimenting MIH. It is written in C language, and only one version has been published and it is not maintained any-more.

Open Dot Twenty-One (ODTONE): This open source implementation of the IEEE 802.21 standard offers a novel approach to interfacing different link layer technologies [25]. It is written in C++, and multiple versions have been released. ODTONE is OS independent implementation of IEEE 802.21 because:

1. Service access points (SAPs) are not implemented as a software API. Instead, the MIH protocol is cloned and reused in the local communications between the MIH-User and the MIHF, as well as between the link management modules and the MIHF. In this way, all the necessary primitives and data-types defined in the standard can be accessed by both MIH-users and link management modules.
2. The usage of the Boost libraries allows the definition of data-types and network-level operations that are system-independent.

Because of the previous features, the second implementation was chosen to be deployed in our service mobility prototype. In ODTONE, each entity from the IEEE MIH architecture runs as a separate and a standalone process. Communication between MIH-User and LINK-SAP processes with MIHF process is done through the MIH protocol over UDP. Whereas for the communication between remote MIHFs, the MIH protocol is transported over TCP.

For the prototype implementation, the MIHF process will be running as an independent process, whereas the MIH-User part will be deployed as a software module inside the service mobility framework. During the start-up phase, the MIHF will be waiting for the MIH-User to register itself. The MIH-User serializes an MIH Indication message "ODTONE specific message (user_register)" and sends it to the local MIHF. If the MIH-User has not registered itself, the MIHF discards any MIH message directed to that MIH-User.

This MIH-User module is considered to be the triggering part for the whole migration process, mainly it has two responsibilities:

- Captures Layer 2 request for imminent handover and starts a UE context migration process.
- Generates different MIH_SAP primitives and sends them to the local MIHF process. The MIHF in turns uses these primitives to encapsulate the correct MIH message and transmit it to the peer MIHF.

The following primitives will be originated by the MIH-User module:

- **MIH_N2N_HO_Commit.Request:** This primitive is used by the source eNodeB-Cloud to check the capability of performing handover to the peer side. The following parameters are used to construct this primitive:
- **MIH_N2N_HO_Commit.Response:** It is used to reply to the source eNodeB-Cloud whether it can proceed with the UE context migration. The following parameters are part of this primitive:

Parameters marked with asterisk in table 4.2 and 4.3 are irrelevant for the service mobility framework implementation. However, they should exist since they are part of the IEEE MIH standard.

Name	Description
Destination Identifier	Remote MIHF that is the destination of this request.
MN Identifier	The MIHF identifier of the MN that commits to perform handover action.
Target MN Link Identifier*	The identifier of the target link for which resources are requested.
Target PoA*	Target point of attachment's link address.
Requested Resource Set*	Parameters required by the target network to perform MN admission control and resource reservation.

Table 4.2. MIH_N2N_HO_Commit Request Primitive Parameters

Name	Description
Destination Identifier	Remote MIHF that will be the destination of this request.
Status	Operation's status.
MN Identifier	The MIHF identifier of the MN that commits to perform handover action.
Target Link Identifier*	Identifier of the target point of attachment for the MN.
Assigned Resource Set*	Set of resource parameters to be assigned to the MN by the target network.

Table 4.3. MIH_N2N_HO_Commit Response Primitive Parameters

4.2.3 CXTP Module

Transmission and reception of contexts related to services and network sessions is done through this module. This module encapsulates and decapsulates these contexts using CXTP protocol. A separate and standalone library is developed to allow the usage of this protocol beyond the scope of this framework. While developing this library, both clear and ease of usage have been taken into consideration.

All CXTP messages have a fixed and common header part of four bytes length, but they differ in their fields members following that header. For this purpose, we create a structure named "Frame". Using this structure, we create a CXTP header that follows the big Endian notation [26]. Primary, this structure contains:

- **Private data member:** This member is an array of characters with a fixed size of 4 bytes. Having this private member allows us to dereference any memory stream of data using Frame data structure and considers it as a CXTP header. All the bytes stream beyond that header will be considered as a part of the payload.
- **Public data members:** These members provide access to various fields of the CXTP header like version, type, length, IP address type, in addition we get a reference to the payload part of the message.

Since CXTP protocol has six different type of messages, and each message has unique data field members. It is necessary to build an archive class which is capable of accommodating various data members and provides serialization/deserialization capabilities for the message payload. This archive should support basic data types including unsigned integer family (16, 32, and 64). By doing this, the archive class can be used to implement and develop any protocol later. The archive class is made up of:

- Buffer that is a vector of bytes (Unsigned Integer 8). This buffer expands its size depending on the size of the associated payload data. Writing data to this vector follows the Big Endian notation, in which the lowest memory address contains the MSB of the data member.
- Pointer to the current position in the buffer.
- Pointer to the beginning of the buffer.
- Length or size of the buffer. When the current position pointer is equal to the size of the buffer, we know that we have read all the data in the buffer.

From this archive class, we derive the following two subclasses shown in figure 4.9:

Figure 4.9. CXTM Message Archive

- **Input Archive (iArchive):** This class is used to deserialize different data types, which implies obtaining the values of different fields in a received message. Depending on the supplied data member type, the archive copies a fixed number of bytes from its buffer to that data member and advances the current position pointer.
- **Output Archive (oArchive):** Serialization of the message payload is done through this class. According to the type of the data member, this archive expands the size of its internal buffer and copies the content of the data member to it.

Finally, to construct a CXTM message, we build a class named "message" to provide a higher level of abstraction. This class will be used by the developer to assign values to the CXTM header and add different data members according to the type of the CXTM message being constructed.

To transmit a constructed CXTM message, the CXTM library derives a CXTM frame structure from that CXTM message. Then, it attaches to its end, the content of the output archive. Whereas to receive a CXTM message, the CXTM frame is used to dereference the first four bytes of the received byte stream as they constitute the header. This is necessary since the header contains the length of the whole message. As a result, the frame structure will be able to initiate its internal buffer to accommodate the entire message.

Figure 4.10 illustrates the hierarchy of the CXTM CTD message used to migrate UE context. From a high level, the CTD message contains a different number of blocks corresponding to different services. Each service migration data block is made up of two main blocks:

1. The service's context block.
2. The service's sockets blocks.

Each one of these two main blocks will carry different type of data, the first block can be split into the following two sub-blocks:

Figure 4.10. CTD Message Hierarchy

1. **The service's context information block:** This block includes information related to the context of the service to be migrated such as Service Identifier (GUID), service's contexts data length, and the number of network sessions being used by the mobile user.
2. **The service's context data block:** Actual service's context data is stored inside this block.

The second main block is responsible for carrying information related to different sockets linked to the same service. It can be divided into two sub-blocks as follows:

1. **The Socket Information Block CDB:** This block includes information about the socket being migrated such as socket's data length and a number of CXTP context data blocks (CDBs) used to carry its information. Since TCP socket consists of a number of queues and these queues may contain packets while migrating the socket, so a single CDB block is not enough to accommodate this socket. Multiple blocks are used to transfer all the states and data related to this socket. Migration data including contexts related to different services and sockets are carried using CTD message. The maximum length of a CXTP message is $2^{16} - 1 = 65$ KBytes. The CTD message contains a series of context data blocks (CDBs) which carry the actual payload. In the specifications of the CXTP CDB frame, the length field's size is 8 bits and each bit represents 8 bytes word. So, as a result, the CDB can carry up to $(2^8 - 1) * 8 = 2040$ bytes including the CDB header and the optional presence vector. Due to this limitation on the size of the CXTP payload and CDB data part, we do the following:

- We propose an extension to the current version of the CXTP protocol. Since the CXTP header contains 6 bits reserved for the future use, we expand the length field from 16 bits to 22 bits. By doing this, a CTD message can accommodate up to $2^{22} - 1 = 4$ MBytes which is enough to carry migration data. Figure 4.11 shows the structure of the modified CXTP CTD message.
- We subdivide migration data into multiple CDB blocks, and each CDB block can hold up to 2036 Bytes of pure data, without the CDB header which is 4 Bytes.

In the importing side, the system initializes bytes container with a size equal to the one defined by the socket's data length in the Socket Information Block. Then, it checks for the number of the CDB blocks required to re-construct the migrated socket again. All the Socket Migration Data Blocks accommodate a payload equal to 2036 Bytes, except the last CDB which will have an amount of socket's payload data equal to $Socket'sdatalength - 2036 * (NumberofCDBs - 1)$. The content of these CDBs is copied into the bytes container, which will eventually constitute the migrated network session.

Figure 4.11. Context Transfer Data (CTD) Message Modified Version

2. **The Socket Data Block:** Actual socket's data including states and queues' contents is stored inside this block.

Each of the previous sub-blocks is considered as standalone CDB Block. As a result, each block should have a unique Feature Profile Type (FPT) to distinguish it and remove any ambiguity in the receiver side.

4.2.4 Performance Logger Module

Performance evaluation is an essential part in any system to measure its capabilities under different conditions. For the service mobility system, we are interested in the measurements related to the amount of time required to migrate a UE context from one eNBC to another. To accomplish this task, we use *Boost.Timer* library that provides a class object named *cpu_timer*. This object measures wall clock elapsed time charged to the user and system. When a *cpu_timer* object is created, it begins timing, and it keeps timing until the stop function is called. Static linking with this library should be considered in the implementation. This is because shared libraries add extra time delays due to the expensive disk accesses.

Every time a new transaction is generated, a *cpu_timer* is created, and it starts counting the elapsed time. It keeps running until either the migrations process has succeeded or failed. Later, the amount of time taken to perform different tasks and whether the migration procedure has succeeded or failed are saved into a log file. Two types of log files are generated correspondingly to the type of the state machine being created either source or destination. Table 4.4 shows the names of the activities for which the time is being captured.

4.2.5 Mobility Hook Module

This module has a single and simple task compared to the previous modules. It is mainly responsible for capturing and validating layer two signals related to the initiation of handover execution phase. Once the relevant signal is detected, this module performs the following steps:

- It identifies the mobile node for which the handover execution is imminent.

- It obtains the IP address of the target eNodeB-Cloud.
- It starts a source state machine which corresponds to a new UE context migration.

4.3 LENA LTE Simulator

The service mobility system is proposed to serve as part of the future releases of the 4G-LTE network architecture. So to evaluate the performance of the prototype implementation, we need to emulate a simplified mobile network environment and combine the service mobility system with it. To achieve this, we use LENA [27], which is an open source LTE/EPC Network Simulator that allows LTE researchers to design, test and verify Self-Organized Network (SON) algorithms and solutions. LENA is based on ns-3 network simulator for internet systems. By using this simulator, we build a simplified mobile network consisting of two eNodeBs: a source eNB1 and destination eNB2 with X2 interface configured between them and one UE connected to eNB1. The X2 interface provided by the simulator provides the following procedures of the Mobility Management functionality:

- Handover Request procedure.
- Handover Request Acknowledgement procedure.
- SN Status Transfer procedure.
- UE Context Release procedure.

The RRC model provided by the simulator allows user-defined functions to be hooked to some useful traces. These functions are called by the RRC state machine upon start and end of the handover execution phase at both the UE and eNodeB sides. To start the X2-based handover procedure, we trigger a manual handover within the simulation by scheduling an explicit handover event. When the Handover Request Acknowledgement procedure is executed, the simulator calls a user-defined function. This function triggers the service mobility system to initiate a UE context migration process.

State Machine Type	Task Name	Description
<i>Source</i>	MIH HO Commit Transaction	The amount of time taken to receive MIH_N2N_HO_Commit Confirm message from the destination eNBC since the source eNBC has sent MIH_N2N_HO_Commit Request message.
	Retrieving Service Context	The time required to obtain service context.
	Retrieving Network Session Context	The time required to obtain the context related to the network session in the Kernel module.
	Prepare Migration Data	The time needed to serialize migration data and send it over the wire.
	Context Transfer Data Reply Message Received	The amount of time taken to receive CTDR message from the destination eNBC since the source eNBC has sent the CTD message.
	Elapsed Time	The time required to carry the whole UE context migration.
<i>Destination</i>	Prepare HO Commit Response Message	The amount of time taken to serialize MIH_N2N_HO_Commit Response and send it over the wire as a response to MIH_N2N_HO_Commit Request.
	Received Migration Data	The amount of time that elapsed since the destination eNBC has sent MIH_N2N_HO_Commit Response and received CTDR message and deserialized it.
	Applications Signaled to Import	The time taken by the Service Mobility to dispatch context data to the service.
	Network Session Imported	The time needed by the Kernel module to import network session context.
	Prepare Context Transfer Data Reply	The amount of time taken to serialize CTDR message and send it over the wire as a response to the CTD message.
	Elapsed Time	The time required to carry the UE context migration from the point of view of the destination eNBC.

Table 4.4. Performance Values for Different Activities

Chapter 5

Experimental Evaluation

This chapter includes evaluation for the behaviour and performance of the implemented service mobility prototype. Also, some conclusions about the results of the tests are drawn.

5.1 Video Server Service

To start evaluating the performance of the service mobility prototype, we start with a simple video server service. In an ordinary case, the user requests a video file to be streamed to his device by establishing a single TCP connection towards the server. Then, the user sends an HTTP GET Request over this connection with the URL of the file to be streamed. In response to this request, the video server starts streaming the content of the video file to the client in the body part of HTTP Response message. Figure 5.1 shows the HTTP transaction with a video server.

Figure 5.1. HTTP Transaction for Video Server Service

When the service starts up, it registers a callback to the service mobility system by calling *register_application API*. So whenever an event is signaled by the service mobility system, an appropriate procedure defined by the service is executed. When a user sends a request for a video file, the video server service performs the following:

1. It registers the user with the service mobility system by calling *register_user API*.
2. A thread is dispatched to handle the user's request, and appropriate context is created including the requested video file name, and the current offset in the video file. Each user's context is uniquely identified through the user's IP address.
3. When the streamed video reaches the end, the video server calls *deregister_user API* and any context data related to the user is removed from the server's contexts table.

5.2 Test Scenario

Figure 5.2 shows the real deployment scenario of the service mobility system. It consists of two eNodeBs, where each eNodeB is equipped with eNodeB-Cloud running service mobility system with video streaming service. We ensure that both video servers have the same video contents. In the beginning, the mobile user (UE) is connected to eNodeB1 and streaming video file from eNBC1. Due to the mobility of the user, it loses its connection with eNodeB1 and performs a handover to eNodeB2. The handover is carried over X2 Interface. As a result, eNodeB1 triggers the service mobility system in eNBC1 to perform UE context migration to eNBC2.

Figure 5.2. Real Deployment Scenario

5.3 Test Setup 1

In the test setup shown in figure 5.3, we emulate the real world deployment scenario. The test setup consists of a single physical machine. The physical machine runs Ubuntu 12.04 operating system with the following applications:

1. Two virtual machines (VMs) running Ubuntu 12.04 (Linux kernel version 3.8.0-42) with service mobility system and video server installed on both of them. Each machine represents the eNodeB with the eNBC connected to it.

Figure 5.3. Test Setup 1

2. LTE LENA Simulator based on ns3. The simulator is responsible for generating LTE X2-based handover signals flow.
3. Open Virtual Switch (OVS). The OVS provides the two VMs with layer 2 bridging and VLAN tagging capabilities. Also, it emulates the physical connectivity between the mobile client and the LTE eNodeB in the real deployment.
4. Video client application to stream video file from an eNBC. This client application emulates the UE.

The OVS acts as a bridge and provides three ports as following:

1. vmnet1 Port: This port is used to connect VM1 to the OVS bridge. It tags all the incoming traffic with VLAN-ID 3.
2. vmnet2 Port: This port is used to connect VM2 to the OVS bridge. It tags all the incoming traffic with VLAN-ID 5.
3. bveth Port: The operating system on the physical machine uses this port to connect to the OVS and, as a result have a connection to either VM1 or VM2. This port uses either VLAN-ID 3 or 5 depending on which VM the video client should stream the video file.

The OS in the physical machine should contain a virtual Ethernet (veth) interface. This veth is used to attach the OS to the OVS bridge it has an IP address of 10.0.0.1/24. Additional to that, the two VMs should possess the same IP address which is 10.0.0.2/24 and the same MAC address. If the two machines have different MAC addresses, the video client will not be able to communicate directly with the target VM upon migration until it obtains the MAC address of the target VM. This is done by sending new ARP broadcast request. When a client wants to connect to VM1, it should use VLAN-ID 3. However, for communicating with VM2, VLAN-ID 5 is used. For emulating a real 4G-LTE mobile network, the ns3 with LENA simulator is deployed. In this simulator, the handover signaling over X2 interface is simulated between two eNodeBs. Upon the execution of the Handover Request Acknowledgement procedure, the ns3 changes the VLAN-ID of the client from 3 to 5. In addition, a predefined function is executed to trigger a migration process. Figure 5.4 illustrates the chronological order of the execution of various tasks during a UE context migration process.

1. The user starts streaming a video file from the HTTP Video Service running on the VM1 (eNodeB-Cloud 1). In the same time, the ns3 schedules for the execution of a manual handover at a particular point of time.
2. At this point, the ns3 starts the execution of X2-based handover signaling scenario.
3. During the simulation of X2 handover signaling, the eNodeB1 inside the simulation scenario receives a Handover Request ACK. This entitles the ns3 to execute the function hooked with the Handover Request Acknowledgement procedure. In this function, the ns3 makes a TCP connection to the MIH-User part in the Service Mobility System located in eNBC2. Then, the ns3 triggers the MIH-User part to start a UE context migration to the peer eNBC.
4. Here, the HTTP Video Service suspends the streaming session and delivers the user's context to the service mobility system. The service mobility in turn extracts the states and queues' contents of the streaming TCP socket. Then, it serializes these data using CXTTP protocol and sends it to eNBC2 over the transport network.
5. The service mobility system in eNBC2 starts deserializing the received message. It obtains different information related to the migrated UE context. Then, it imports contexts related to the HTTP video service and its network session. Finally, the ns3 switches the L2 connection by changing the VLAD-ID of the client from 3 to 5.
6. Now the user can resume its HTTP streaming session from the new eNBC from the place it left off.

Figure 5.4. Demo Scenario

5.3.1 Evaluation and Analysis

This section contains an analysis for different performance values reported in Table 4.4 for the previous test setup. The analysis reports statistical results for each performance item in forms of min, max, mean, standard deviation and histogram. Figure 5.5 shows the histogram of UE context migration time after repeating the previous demo for 150 times. From the figure, we can see that the histogram for the UE context migration time is spread over a broad range of values.

Figure 5.5. Histogram of UE Context Migration Time for Test Setup 1

We try to analyse different components contributing to the total migration time. Figure 5.6 shows the histogram of various performance values taken during the evaluation of the service mobility prototype. We can see that both MIH HO Commit Transaction Time and Retrieving Sockets Data Time do not have high influence on the total migration time since their values are small and do not vary too much. Whereas for Retrieving Application Data Time and CXTP Serialization & Transmission Time components, they form the biggest parts of the total migration time. The following subsections illustrate more about these two components. Table 5.1 reports the statistical results for the obtained experimental data.

Performance Value/Statistics	Min	Max	Mean	STD
MIH HO Commit Transaction Time	3.0248	61.7937	13.6266	7.6606
Retrieving Applications Data Time	0.1091	244.3538	86.3961	96.4123
Retrieving Sockets Data Time	0.0440	51.0340	1.5560	4.4071
Prepare Migration Data Time	1.1256	331.3293	32.0479	70.5198
Total Migration Time	12.2732	1275.7	185.2639	186.8156

Table 5.1. Statistical Results for UE Context Migration Time Components in [ms] for Test Setup 1

Figure 5.6. Histogram of Various Activities for Test Setup 1

5.3.1.1 Application Analysis

After further analysis to the video server application, it turned out that the most time-consuming task in the application is the locking of the *mutex*. The usage of the *mutex* is necessary since we have multiple threads running concurrently, so it is necessary to synchronize their execution to avoid any race condition. Figure 5.7 shows the histogram of the context migration time after subtracting the time required by the application to provide the user's context to the service mobility system. In this figure, we can notice that the shape of the histogram of the UE context migration time has changed significantly. It has changed from being spread over a significant range of values to contain sharp edges around certain values. These new sharp edges at high values are due to the time taken to transport migration data from the exporting VM to the importing one.

The last test is carried out again, but this time by eliminating the usage of *mutex* for threads synchronization in the Video Server application. In this case, we use a simple boolean variable to lock the thread handling the user's request. This method does not guarantee the consistency of the received data and results in race condition. By eliminating the usage of *mutex*, we remove the overhead caused to lock the thread. Figure 5.8 shows histogram of the UE context migration time after modifying the video server application. The histogram is not spread over large values, and it matches the previous case where we removed the time required by the application in a statistical way. However, we still have small peaks causing high migration time values. These small peaks are due to the following two reasons:

1. Serialization and transmission of the CXTP CTD message over TCP connection from the

Figure 5.7. Histogram of UE Context Migration Time without Application's Time

exporting VM (VM1) to the importing VM (VM2).

2. Reception and deserialization of the CXTP CTD message in the importing VM (VM2).

So good implementation of the application should be taken into consideration when interacting with service mobility system. Different mechanisms to synchronize multi-threading applications should be studied and compared against each other to select the one with the lowest overhead.

Figure 5.8. Histogram of UE Context Migration Time Without Mutex

Table 5.2 reports statistical values for the obtained experimental data when the *mutex* is eliminated in the video server application. In this table, we can notice a dramatic reduction in the values of the statistical functions compared to the results reported in Table 5.1.

Performance Value/Statistics	Min	Max	Mean	STD
Retrieving Applications Data Time	0.0990	6.7281	1.1857	1.3784
Total Migration Time	14.5940	552.7835	76.0942	99.6555

Table 5.2. Statistical Results for UE Context Migration Time After Eliminating the Mutex in [ms]

5.3.1.2 CXTP Analysis

To analyse the high time caused by CXTP serialization and transmission, we perform two experiments. In these experiments, the CXTP module used in the service mobility prototype is isolated and implemented as a separate application. This application uses the CXTP protocol to encapsulate a big chunk of data defined by the user and transmits it over a single TCP session.

- **Transmission between two Virtual Machines (VMs):** In this experiment, we have two VMs running on the same host and connected to a virtual bridge. These VMs exchange data with each other using CXTP protocol as shown in figure 5.9.

Figure 5.9. Transmission between two VMs

Figure 5.10 shows the histogram of the amount of time required to transmit a file of size 1.2 MBytes between these two VMs. In this experiment, 95% of the values are below 5.244 s. In addition, the distribution has high mean and standard deviation values.

- **Transmission between two Physical Machines (PMs):** The last experiment is repeated again using two physical machines running Linux OS with no virtualization as shown in figure 5.11.

Figure 5.12 displays the histogram of the time required to transmit the previous file. In this case, 95% of the values are below 20.2 ms. Additional to that, both the mean and standard deviation values are so small compared to the previous experiment.

Figure 5.10. Histogram of CXTP Serialization and Transmission between two VMs

Figure 5.11. Transmission between two Physical Machines

From the results reported in table 5.3, we can explain the high amount of time caused by the CXTP module in the service mobility system for test setup 1. It is due to the overhead caused by the emulation of the networking resources in the host operating system. This is because the virtual machines use a hypervisor installed above the operating system to get access to the networking resources in it. This additional layer causes some overhead in terms of transmission delays, since multiple virtual machines will be using these resources concurrently. Additional to that, the scheduling of the VMs to access the share the same CPU and RAM has some effects too. So it is recommended to deploy the service mobility system directly on the physical machine to achieve efficient and optimal results.

Test/Statistics	Min	Max	Mean	STD
<i>Transmission Between VMs</i>	18.4634	14012	1726.9	1733.1
<i>Transmission Between PMs</i>	17.7966	205.5691	25.7251	6.4283

Table 5.3. Statistical Results for CXTP Transmission Time in [ms]

Figure 5.12. Histogram of CXTP Serialization and Transmission between two Physical Machines

Figure 5.13 shows the histogram of the UE context migration time after subtracting the time required to transmit a CXTP CTD frame. In this figure, we can notice that the shape of the histogram has changed significantly. We still have some peaks at high values. This is because the UE context migration time depends on the importing machine (VM2) to receive a CXTP frame, deserialize it and send a CXTP CTDR message back to the exporting machine (VM1) over a new TCP connection.

Figure 5.13. Histogram of UE Context Migration Time after Removing CXTP TX Time

5.3.1.3 Context Data Length Analysis

Figure 5.14 displays the histogram of the context data size when the application uses the *mutex* for threads synchronization. The context data contains the HTTP video server's context and its network session's context. We found that there is no correlation between the migration time and the length of the context data. The size of the UE context varies a lot depending on the status of the network between the video server and the client. The average value of the buffer's size is around 115 KBytes. This explains the necessity to the proposed extension for the length of the CXTP message to accommodate more data. Table 5.4 shows the statistical results for the context data length.

Figure 5.14. Histogram of Context Data Length

Test/Statistics	Min	Max	Mean	STD
Export Buffer Length	72.3281	204.3281	114.9255	23.1371

Table 5.4. Statistical Results for Context Data Length in [KB]

5.4 Test Setup 2

To test the service mobility system directly on the physical machine without virtualization, we propose the test setup shown in figure 5.15. The test setup consists of two physical machines representing eNBCs connected to eNodeBs. The physical machines run Ubuntu 12.04 (Linux kernel version 3.8.0-42) operating system, and the implemented prototype of the service mobility system. In addition, they run an instance of open virtual switch (OVS) inside them. We use a simple video server to represent a service running on the eNBC, and we also ensure that both video servers have the same video contents. Additionally, the two servers should use simple boolean variable for threads synchronization instead of the *mutex*. A virtual machine (VMClient) running on physical machine 1 is used to emulate the UE. This virtual machine is connected to a virtual switch on the physical machine 1.

Figure 5.15. Test Setup 2

The OS in both physical machines should contain a virtual Ethernet (veth) interface. This veth is used to attach the OS to the OVS bridge. The veth interface on the two physical machines should possess the same IP address which is 10.0.0.2/24 and the same MAC address. Whereas, the client uses IP address 10.0.0.1/24. The direct connection between the two physical machines emulates the connectivity link between eNBCs used for handover signaling and context data transfer over CXTP. A GRE tunnel (reflected as a port on the virtual switches) is established between the two virtual switches over the wireless interface of the physical machines. The purpose of the GRE tunnel is to carry the traffic between the UE and the video server running on physical machine 2 after the handover of the UE. When the client wants to connect to Physical Machine 1, it should use VLAN-ID 3. The path of the user traffic will be as follow:

VMClient → OVS Bridge1 → Physical Machine 1

However, for communicating with Physical Machine 2, VLAN-ID 5 is used. In this case, the path followed by the user traffic will be as follow:

VMClient $\rightarrow$ OVS Bridge1 $\rightarrow$ GRE Tunnel $\rightarrow$ OVS Bridge2 $\rightarrow$ Physical Machine 2

5.4.1 Evaluation and Analysis

We start testing the system by setting the VLAN-ID for vmnet Port on Physical Machine 1 to be 3. In this case, the VMClient streams the video file from the video service running on Physical Machine 1. During the streaming session, we send a command to the service mobility system on Physical Machine 1 to migrate the context of the VMClient to Physical Machine 2. At the same time, we change the VLAN-ID of vmnet Port on Physical Machine 1 to be 5. In this case, the VMClient continues streaming the video file from the video service running on Physical Machine 2. We repeat the previous experiment for 50 times and we plot the results. Figure 5.16 shows the histogram of the UE context migration time.

Figure 5.16. Histogram of UE Context Migration Time for Test Setup 2

Figure 5.17 shows the histogram of different activities contributing to the total context migration time. Table 5.5 summarizes statistics results for different components. From the table, we can notice a massive reduction in the mean values for the UE context migration time, and its various activities compared to the case where the service mobility system was deployed on a virtual machine. In addition, we see that the overall time taken to migrate a UE context is around 12 ms on average. Requirements according to ITU-R state that the UE handover interruption time to be between 27.5 ms to 60 ms [28]. According to 3GPP requirements for support of radio resource management, a maximum interruption time of 130 ms during UE handover is suggested [29]. When eNodeB hands over a UE to another eNodeB, the context migration times obtained in this test setup, are within the limits of UE handover interruption times in LTE networks. This illustrates the practical feasibility of deploying services over eNBCs without significant interruption in the service during UE handover.

This allows a make-before-break migration of UE context.

Figure 5.17. Histogram of Various Activities for Test Setup 2

Performance Value/Statistics	Min	Max	Mean	STD
MIH HO Commit Transaction Time	2.8045	7.7197	3.4935	0.6552
Retrieving Applications Data Time	0.2485	0.6255	0.4157	0.0840
Retrieving Sockets Data Time	0.4647	2.6449	1.1765	0.3094
Prepare Migration Data Time	0.8778	8.7015	1.8883	1.0719
Total Migration Time	7.9069	18.9256	11.4726	2.6538

Table 5.5. Statistical Results for UE Context Migration Time Components in [ms] for Test Setup 2

Chapter 6

Conclusions

In this thesis, we have presented the concept of Service Mobility, which enables seamless migration of UE context from one service instance to another. Service Mobility System is designed to be offered as a part of the cloud platforms. Service providers can leverage on the system to deploy services on eNodeB-Clouds in order to bring the services closer to the UE. We have discussed the architecture of the Service Mobility System and evaluated a prototype implementation of the same. The design of the system uses standard interfaces such as MIH and CXTP, which allows interoperability between different vendors and, as a result, eases its adoption and proliferation. The results obtained from the evaluation indicate that using service mobility with virtualization imposes high time for migrating UE context. The high time is due to the emulation of the networking interfaces. However, this does not imply that service mobility cannot be used with virtualization. For example, if the eNodeB-Cloud hosts a video streaming service on a virtual machine, the client will not notice the large amount of time taken to migrate its context if a buffer of 1 seconds is used. But for highly interactive services, a client might experience some jitter but not service discontinuity. Additionally, the results show that using the service mobility system without virtualization produces small UE context migration times. These times are within the limits of UE handover interruption times. For this reason, network service providers willing to integrate cloud computing capabilities into their network architecture, are engaged to deploy service mobility system without virtualization.

6.1 Future work

During the evaluation stage of the service mobility prototype, the test setup had one client using one service over a single network session. So it is necessary to test the prototype with different services and generate an optimized solution, capable of handling high number of users with different service requirements in a short period. The following sections introduce a brief introduction for three potential which could interact with the service mobility system. Finally, two design improvements are suggested to be investigated and tested in the future work.

6.1.1 Potential Services

6.1.1.1 Web Server Service

Most of the Internet services today like web services use multiple TCP connections to improve end user's throughput. So when a user is surfing the web, multiple TCP connections are established to retrieve different objects in the requested web page. So there is a necessity to evaluate the service migration prototype with multiple network sessions.

6.1.1.2 HTTP Proxy Service

A proxy server works as an intermediary point between clients seeking requests from web servers on the Internet. The client establishes a connection to the proxy server requesting a web page or other resources available from a different server. The proxy server in returns interprets and processes client's request, and send a new request to the real web server on the behalf of the client. With respect to a Web Server service, an HTTP proxy service has a special requirement; it should be configured with two different IP addresses:

- A private IP address for communicating and relaying HTTP messages with the client.
- A public IP address for communicating with the actual server located on the Internet.

This particular requirement requires some investigations regarding the modifications to be done on the design of the HTTP proxy service, for interacting with service mobility system.

6.1.1.3 Routing Information Migration Service

With the growth in revenues achieved by machine-to-machine (M2M) communications, most mobile network operators are investing a large amount of resources in their network infrastructure, marketing and sales to grow their M2M business. When multiple machines need to communicate with each other through LTE mobile network, each node has to establish a GTP tunnel with the LTE-PGW node to exchange messages with other nodes.

If these machines are located in the same proximity, it will be a waste of resources to traverse their messages across the transport network which results in network congestion, waste of capacity and high delays. A better solution would be by using a Local Break Out (LBO) functionality in the access points (APs). LBO means if two machines are within the same proximity they can exchange message directly through the serving AP only, without having to create a GTP tunnel and go through the mobile core network. As a result, the AP will act as a router between these machines and takes the responsibility for forwarding messages between them.

Due to this new connection model a new problem arises. Since these sensors are mobile, so they might move and be served by different APs. This breaks down connections with other sensors in the same proximity. One solution to this problem can be by migrating routing information related to the mobile sensor and install it in the new AP. To fulfil this task, a modification in the Service Mobility framework is proposed. In this new modified framework, the network session migration module is removed, and the routing information base (RIB) is considered to be as a service for which the context should be migrated.

6.1.2 Further Design Improvements

6.1.2.1 CXTP Proxy

The CXTP proxy behaves exactly the same as a normal proxy service in a communication network. It breaks the communication between the MIHF entities and encapsulates the MIH protocol message inside a CXTP message. This implies having the MIH protocol frame as a standalone CDB block within the CXTP frame, with a unique Feature Profile Type (FPT) assigned to this block. Due to this change, a context migration process can be accomplished much faster. The CXTP proxy framework architecture is shown in figure 6.1.

Figure 6.1. CXTP Proxy Framework Architecture

Figure 6.2 displays signaling flow model to migrate a UE context using this new framework. The MIH_N2N_HO_Commit Request is encapsulated inside the CXTP CTD frame, whereas MIH_N2N_HO_Commit Response is delivered inside the CXTP CTDR frame.

Figure 6.2. Signaling Flow with CXTP Proxy Enabled

6.1.2.2 Tunnelling MIH and CXTP

Current design of the service mobility framework separates handover signaling messages related to the radio mobility, from the messages corresponding to the UE context migration. The main drawback of this issue is having two handover operations taking place concurrently and separately. In addition to that, the service migration procedure acquires four signaling messages to be completed. One possible solution for this issue is by tunneling both MIH and CXTP messages inside an X2/S1-C Handover frame. The figure 6.3 illustrates the new modifications required for this new architecture:

Figure 6.3. Tunnelling MIH and CXTP over X2 and S1-MME Framework Architecture

The interfaces of this new framework are defined as follow:

1. An API towards the services for freezing their executions and retrieving contexts.
2. Interface used to retrieve network sessions contexts.
3. Interface between service mobility system and Session Handover Service.
4. IEEE 802.21 MIH specification standard interface.
5. CXTP protocol that can be routed through the X2 or S1-MME interface.
6. The MIH protocol that can be routed through the X2 or S1-MME interface.
7. Interface for eNB (Radio Network Logic RNL)-PaaS interaction. It is used for triggering context migration (based on imminent handover).
8. Possible local breakout (LBO) state retrieval.

Bibliography

- [1] 3GPP. Network architecture. TS 23.002, 3rd Generation Partnership Project (3GPP), September 2014.
- [2] 3GPP. General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access. TS 23.401, 3rd Generation Partnership Project (3GPP), September 2014.
- [3] Ericsson. Ericsson Mobility Report On The Pulse Of The Networked Society, 2014. <http://www.ericsson.com/res/docs/2014/ericsson-mobility-report-june-2014.pdf>.
- [4] Recommendations to governments and industry on ensuring interoperability by eicta, the voice of the ict and consumer electronics industry in europe. 2004.
- [5] Peter Stuckmann Paulo Teixeira de Sousa. Telecommunication network interoperability. European Commission, Belgium.
- [6] Massimo Bernaschi, Francesco Casadei, and Paolo Tassotti. Sockmi: a solution for migrating tcp/ip connections. In *PDP*, pages 221–228. IEEE Computer Society, 2007.
- [7] Florin Sultan, Kiran Srinivasan, Deepa Iyer, and Liviu Iftode. Migratory tcp: Connection migration for service continuity in the internet. In *ICDCS*, pages 469–470, 2002.
- [8] Pravin Bhagwat, David A. Maltz, and Adrian Segall. Msocks+: an architecture for transport layer mobility. *Computer Networks*, 39(4):385–403, 2002.
- [9] Victor C. Zandy and Barton P. Miller. Reliable network connections. In Ian F. Akyildiz, Jason Yi-Bing Lin, Ravi Jain, Vaduvur Bharghavan, and Andrew T. Campbell, editors, *MOBICOM*, pages 95–106. ACM, 2002.
- [10] Werner Almesberger. Tcp connection passing.
- [11] Florin Sultan, Aniruddha Bohra, and Liviu Iftode. Service continuations: An operating system mechanism for dynamic migration of internet service sessions. In *SRDS*, pages 177–186. IEEE Computer Society, 2003.
- [12] C. Perkins. IP Mobility Support for IPv4. RFC 3220 (Proposed Standard), January 2002. Obsoleted by RFC 3344.
- [13] Behrouz A. Forouzan. *TCP/IP Protocol Suite Fourth Edition*. McGraw-Hill, 2010.
- [14] Massimo Bernaschi, Francesco Casadei, and Paolo Tassotti. Migration of secure connections using sockmi. *SYSADMIN*, 32(4), August 2007.
- [15] Rami Rosen. *Linux Kernel Networking: Implementation and Theory*. Apress, 2014.

- [16] Rusty Russell and Harald Welte. Linux netfilter hacking howto. <http://www.netfilter.org/>.
- [17] Antonio de la Oliva, Albert Banchs, Ignacio Soto, Telemaco Melia, and Albert Vidal. An overview of iee 802.21: media-independent handover services. *IEEE Wireless Commun.*, 15(4):96–103, 2008.
- [18] Ieee standard for local and metropolitan area networks - media independent handover services. *IEEE Std 802.21-2008*, pages 1–0, Jan 2009.
- [19] P.A Shah and M. Yousaf. End-to-end mobility management solutions for tcp: An analysis. In *Computer and Information Technology, 2008. ICCIT 2008. 11th International Conference on*, pages 696–701, Dec 2008.
- [20] J. Loughney, M. Nakhjiri, C. Perkins, and R. Koodli. Context transfer protocol (cxtcp). RFC 4067 (Proposed Standard), July 2005.
- [21] Stefania Sesia, Issam Toufik, and Matthew Baker. *LTE, The UMTS Long Term Evolution: From Theory to Practice*. Wiley Publishing, 2009.
- [22] 3GPP. Evolved Universal Terrestrial Radio Access Network (E-UTRAN); X2 Application Protocol (X2AP). TS 36.423, 3rd Generation Partnership Project (3GPP), September 2014.
- [23] EventHelix.com. Lte x2 handover sequence diagram. <http://www.eventhelix.com/lte/handover/x2/lte-x2-handover-sequence-diagram.htm>.
- [24] Antti Jaakkola. Implementation of transmission control protocol in linux.
- [25] Daniel Corujo, Carlos Guimraes, Bruno Santos, and Rui L. Aguiar. Using an open-source iee 802.21 implementation for network-based localized mobility management. *IEEE Communications Magazine*, 49(9):114–123, 2011.
- [26] J. Reynolds and J. Postel. Assigned Numbers. RFC 1700 (Historic), October 1994. Obsoleted by RFC 3232.
- [27] CTTC. Lena: Lte-epc network simulator. <http://networks.cttc.es/mobile-networks/software-tools/lena/>.
- [28] ITU-R. Requirements related to technical performance for IMT-Advanced radio interface(s). Technical Report M.2134, International Telecommunication Union (ITU), 2008.
- [29] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Requirements for support of radio resource management. TS 36.133, 3rd Generation Partnership Project (3GPP), October 2014.