College of Engineering, Design and Physical Sciences Electronic and Computer Engineering

Software Defined Networking Integration with Mobile Network towards Scalable and Programmable Core

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy (PhD)

Mukhald Salih
Student Number: 0835699

Supervisor:
Professor John Cosmas

Year of Submission: 2019

Abstract

The huge appreciation received by the Software Defined Networking (SDN), Network Functions Virtualization, and Cloud Computing in latest years pushed researchers, vendors, and mobile network operators to investigate the possibility of innovative design that integrates these technologies in cellular network aiming to overcome the limitations posed by currently deployed mobile networks and cope with the increasing demands of mobile customers.

This thesis describes an experimental investigation, design, implementation and evaluation of three different solutions to integrate SDN with mobile network.

The first two approaches exploit the evolution of SDN where the control plane of the current LTE entities like the Mobility Management Entity, Serving Gateway and Packet Data Network Gateway are running as packages in an SDN controller or as VMs running in a cloud environment, while the data plane entities are represented by Openflow switches and eNodeBs.

The third approach uses SDN as an add-on to the backhaul of the existing cellular network to introduce variety of new services, selective traffic offloading to cloud-based infrastructure is used to demonstrate the advantage and disadvantage of different solution to implement this approach.

Whereas all the proposed solutions have been proven to provide enhancement to the system performance. The first solution shows that utilizing SDN helps to reduce the signalling load, provides faster recovery time and better resource utilization. Extending Openflow protocol plugin to support mobile network operations reduce the initial attachment signalling loads by 66% and serving gateway failover by 40%.

In the Second solution, utilizing SDN enhanced the core network links utilization by 25% compared to the current mobile network implementation. Also, it reduced queuing time and the packet loss by up to 4% when the network is congested, which contributes to reduce the end-to-end delay by up to 27.9%.

In the last solution, SDN is utilized to move the content near to the mobile users, which contributes to reduce the end-to-end delay of the delay sensitive traffic. The packet processing time in the third approach is much less that the first two approaches which contributes to provide better performance in term of end-to-end delay.

Acknowledgments

First and foremost, I wish to express my deepest appreciation and sincere gratitude to my supervisor, Professor John Cosmas, for his valuable guidance, encouragement and great support during my PhD research. It has been an honour to work under his supervision.

Also, it has been a privilege to work with my second supervisor Dr Rajagopal Nilavalan who I sincerely grateful to him.

I would also like to express my sincere thanks to the Higher Committee for Education Development in Iraq HCED and the Ministry of Communication for the financial support that made my PhD work possible.

I would like to thank all my friends and colleagues at Brunel University who supported me during my PhD research.

Finally, I would like to acknowledge the financial, academic and technical assistance given by Brunel University with special gratitude to the staff in the Student Centre, the Post-Graduate Research Office, the Library, the Accommodation Office and the Residences Office.

Dedication

I dedicate this work to

Mum and Dad for their sacrifices and their support to help me get the best education possible;

Brothers and sister for their support and encouragement;

Beloved wife and kids for their love, support, encouragement and for giving me the motivation to work hard during my research period;

List of Contents

Abstract i
Acknowledgments ii
Dedication iii
List of Figures ix
List of Tables xiii
List of Abbreviations xiv
1 Introduction 1
1.1 Introduction 1
1.2 Motivations 1
1.3 Aims and Objectives 2
1.4 Challenges 3
1.5 Contributions 5
1.6 Methodology 6
1.7 Publications 7
1.8 Thesis Organisation 7
2 Literature Review 9
2.1 Introduction 9
2.2 Software Defined Mobile Network Architecture 10
2.2.1 SDMNA with GTP Extensions 12
2.2.2 SDMNA without GTP 16
2.3 SDN as an Add-on 20
2.3.1 Selective Traffic Offloading and Caching 20
2.3.2 Distributed Mobility Management 25
2.4 Software Defined Virtualized Mobile Network Architec-tures 27
2.5 Cloud RAN and SDMN Security 30
2.6 Summary 31
3 OpenFlow Technology and Implementation Model 33

3.1 Introduction 33
3.2 Networking Technologies 33
3.2.1 Traditional networking 34
3.2.2 Software Defined Networking 35
3.3 OpenFlow 37
3.3.1 OpenFlow Switch 39
3.3.2 Connection Establishment between Switch and Controller 48
3.3.3 OpenFlow Messages 48
3.4 Simulation Model 49
3.4.1 OMNeT++ Simulation Framework 49
3.4.2 OpenFlow 1.3 Simulation Model 50
3.5 Openflow 1.3 Model Validation 64
3.6 Summary 66
4 Evolved Packet System Technologies and Implementation Model 67
4.1 Introduction 67
4.2 Evolved Packet System 67
4.2.1 Evolved UMTS Terrestrial Radio Access Network 68
4.2.2 Evolved Packet Core 69
4.3 Interfaces and Protocol Stacks 72
4.3.1 Signalling Protocols 72
4.3.2 User Plane Protocols 73
4.3.3 GPRS Tunnelling Protocol 77
4.4 EPS Procedures 79
4.4.1 EUTRAN Initial Attachment Procedure 79
4.4.2 Access Bearer Release Procedure 83
4.4.3 Service Request Procedure 85
4.4.4 Handover 86
4.5 Evolved Packet System Model 89
4.5.1 SimuLTE Platform 90
4.5.2 Data Plane Traffic Forwarding in LTE Simulation Model 110

4.6 Summary 113
5 Software Based Mobile Core Network Architecture 114
5.1 Introduction 114
5.2 SBMCN Architecture 115
5.2.1 Network Controller 115
5.2.2 MME 116
5.2.3 Serving and PDN Gateways Control Plane 116
5.2.4 Forwarding Device 116
5.2.5 OpenFlow-Based eNodeB 116
5.3 Architecture Operations 117
5.3.1 Initial Attachment and Initial Access Bearer Setup Procedures 117
5.3.2 Resiliency 121
5.3.3 Load-Balancing 125
5.4 Network Model 127
5.4.1 NC Controller Module 129
5.4.2 GTP-Capable-OpenFlow-Switch Module 130
5.4.3 OpenFlow-Based eNodeB 133
5.4.4 Local Agent 137
5.5 Simulation Setup and Results 141
5.5.1 Applications 142
5.5.2 Signalling Experiment 144
5.5.3 SDN Agent and Load balancing Experiment 149
5.6 Summary 153
6 Software Defined Evolved Packet Core 155
6.1 Introduction 155
6.2 SDEPC Architecture 156
6.2.1 Architecture Description 156
6.2.2 Architecture Operations 159
6.3 Simulation Study of SDEPC Architecture 163
6.4 Simulation Results and Analysis 164

6.4.1 GTP Overhead Experiment 165
6.4.2 SDEPC Performance Evaluation Experiment 167
6.4.3 UE to UE Communication Experiment 169
6.5 Advantages and Disadvantages 174
6.6 Backward Compatibility Support 174
6.6.1 SGW and PGW Data Plane Forwarding Device 175
6.6.2 Simulation Network 181
6.6.3 Performance Metrics and Evaluation Results 183
6.7 Summary 189
7 Software Defined Selective Traffic Offloading 191
7.1 Introduction 191
7.2 Solution 1 Network Design 192
7.2.1 Traffic Offloading Switch Implementation in OMNeT++ 193
7.2.2 Control Plane Operations 199
7.2.3 Data Plane Traffic Forwarding Procedures 203
7.2.4 Handover 209
7.2.5 Accounting and Charging 211
7.2.6 Advantage and Drawbacks 212
7.3 Solution 2 Network Design 212
7.3.1 Control Plane Setup Procedure 213
7.3.2 Data Plane Traffic Forwarding Procedures 215
7.3.3 Handover 218
7.3.4 Accounting and Charging 218
7.3.5 Advantage and Drawbacks 219
7.4 Solution 3 Network Design 219
7.4.1 Control Plane Operations 221
7.4.2 Data Plane Traffic Forwarding Procedures 225
7.4.3 Handover 228
7.4.4 Accounting and Charging 229
7.4.5 Advantage and Drawbacks 230

7.5 Simulation Scenario 230
7.5.1 Simulated Module of the MME Entity 232
7.5.2 Simulation Parameters 235
7.5.3 Performance Metrics and Evaluation Results 237
8 Conclusion and Future Works 248
8.1 Conclusion 248
8.2 Future Works 249
References 251
Appendix 258

List of Figures

Figure 2.1: SoftCell Network Architecture proposed by [35][1] 12
Figure 2.2: On-Demand Connectivity Architecture proposed by [4][15] 13
Figure 2.3: OpenFlow-enabled mobile core network Architecture proposed by [16] 14
Figure 2.4: Disruptive integration of SDN with MME proposed by [6] 17
Figure 2.5: Mobile Cloud Offloading Architecture (MOCA) proposed by [11] 21
Figure 2.6: Software-Defined Networking Mobile Offloading Architecture (SMORE) proposed by [12] 22
Figure 2.7: Double-NAT DMM Architecture proposed by [24] 25
Figure 2.8: Cellular software defined network proposed by [19] 30
Figure 3.1: Software Defined Networking Architecture 37
Figure 3.2: OpenFlow 1.0 Switch Structure 40
Figure 3.3: OpenFlow 1.2 Switch Structure 41
Figure 3.4: Main Component of OpenFlow Switch 42
Figure 3.5: Flow Table Basic Structure 42
Figure 3.6: Group Table Basic Structure 44
Figure 3.7: Meter Table Basic Structure 45
Figure 3.8: OpenFlow 1.3 switch processing pipeline procedure 46
Figure 3.9: OMNeT++ Basic Building Blocks 50
Figure 3.10: OpenFlow 1.3 Module Classes UML Diagram 52
Figure 3.11: Actions Validation Procedure of the OFP-Flow-Mod Message 55
Figure 3.12: OFA_Switch Module handleMessage Procedure 56
Figure 3.13: executeInstructionSet method Operations 58
Figure 3.14: Flow Table content at simulation runtime 58
Figure 3.15: OMNeT++ Module of OpenFlow Switch 59
Figure 3.16: OMNeT++ Module of OpenFlow controller 61
Figure 3.17: Controller Node Operation with Switch Application 62
Figure 3.18: Messages Implemented in OpenFlow 1.3 Model 63
Figure 3.19: Openflow 1.3-based Network Topology 64
Figure 3.20: Openflow 1.3 Channel Setup Interaction 65
Figure 3.21: Openflow 1.3 First Packet Handling Procedure 65
Figure 4.1: Main Component of the EPS Architecture 68
Figure 4.2: Evolved UMTS Terrestrial Radio Access Network Main Components .. 69

Figure 4.3: EPC basic network function elements and the connection interfaces... 72
Figure 4.4: EPS communication interfaces and protocols 74
Figure 4.5: Control Plane Protocol stack for compunctions between a UE and MME 74
Figure 4.6: Data and Control Plane Protocol stacks for X2 Interface 75
Figure 4.7: User Plane Protocol Stack 75
Figure 4.8: EPS Bearers 77
Figure 4.9: Traffic Flow Template to EPS Bearer to GTP-U tunnel mapping 79
Figure 4.10: Initial Attach Procedure call flow 83
Figure 4.11: Access bearer release Procedure 84
Figure 4.12: Access Bearer Setup Procedure 86
Figure 4.13: X2-Based Handover When the SGW and MME kept the same 89
Figure 4.14: UE and eNodeB modules as implemented in simuLTE 91
Figure 4.15: UML Class Diagram for the control plane Modules 95
Figure 4.16: OMNeT++ Module of the MME 96
Figure 4.17: OMNeT++ Module of the MME Application 97
Figure 4.18: MME Application Module Operations 100
Figure 4.19: OMNeT++ Module of the SGW 101
Figure 4.20: OMNeT++ Module of SGW Application 102
Figure 4.21: SGW Application Module Operations 103
Figure 4.22: OMNeT++ Module of the PGW 104
Figure 4.23: OMNeT++ Module of PGW Application 105
Figure 4.24: PGW Application Module Operations 106
Figure 4.25: OMNeT++ Module of the eNodeB 107
Figure 4.26: OMNeT++ Module of eNodeB Application 108
Figure 4.27: eNodeB Application Module Operations 109
Figure 4.28: Downlink Data processing by LTE simulation Platform 111
Figure 4.29: Uplink Data Processing by LTE simulation platform 113
Figure 5.1: Software-Based Mobile Core Network Architecture 117
Figure 5.2: Initial Attachment Procedure as proposed by [4] 118
Figure 5.3: Initial Bearer Setup Procedure proposed by [4] 119
Figure 5.4: GTP tunnel Setup Methods 121
Figure 5.5: SGW failover procedure based on 3GPP specification 122
Figure 5.6: SBMCNA SGW failover procedure 123
Figure 5.7: SGW Failover procedure based on [4] 124
Figure 5.8: 3GPP Load-Balancing Solution 125
Figure 5.9: Load-Balancing Solution proposed by [4] 126
Figure 5.10: Network topology used in OMNeT++ simulation 128

Figure 5.11: Controller Module in OMNeT++ 129
Figure 5.12: Implemented Model of a GTP capable OpenFlow Switch 131
Figure 5.13: OpenFlow-based eNodeB Module Structure 133
Figure 5.14: onDemandSwitchApp Advanced Mode Operations 135
Figure 5.15: Internal Structure of the ProcessingUnit module of the eNodeB Node 137
Figure 5.16: Agent Module Structure 138
Figure 5.17: OFAgent class tree structure 139
Figure 5.18: Control and Control Plane Operations of the OFAgent module 141
Figure 5.19: Number of Control Messages Sent by the Controller vs. Number of UEs Running a Single Application that Sends PING Messages to the InternetHost 146
Figure 5.20: Number of Control Messages Sent by the Controller to Handle the SGW-D Failover with respect to the Number of UEs Served by the Failed SGW-D 148
Figure 5.21: Number of Control Messages Sent by the Controller when the FDs have i) Normal OpenFlow plugin, ii) Enhanced OpenFlow plugin, iii) Enhanced OpenFlow plugin with Agent 150
Figure 5.22: Load Distribution between the SGW-Ds in the Local Pool for both Weighting Factor and Network Load Selection Mechanism 153
Figure 6.1: SDEPC network 158
Figure 6.2: Simple Software Defined EPC Operations 162
Figure 6.3: SDEPC OMNET++ Simulation Model 164
Figure 6.4: Load vs Link Utilization 166
Figure 6.5: System performance comparison result 168
Figure 6.6: EPS vs SDEPC Mean RTT 171
Figure 6.7: EPS vs SDEPC RTT Vectors 171
Figure 6.8: VoIP Traffic - end-to-end Delay comparison of EPS and SDEPC 172
Figure 6.9: VoIP Traffic - end-to-end Delay Time Vectors 173
Figure 6.10: VoIP Traffic - EPS vs SDEPC Mean Opinion Score 173
Figure 6.11: SGW and PGW Data plane FD module 176
Figure 6.12: UML representation of the SGW and PGW FD Modul 180
Figure 6.13: Logical Topology of the simulated network in OMNeT++ 181
Figure 6.14: UE1(0) serving Cell Id though out the simulation 184
Figure 6.15: Packet sent by the InternetHost and the SGW-FD 185
Figure 6.16: GTP tunnel statistic When the Packet size equal to 400 Byte 186
Figure 6.17: SDEPC UE1(0) end-to-end delay 186
Figure 6.18: end-to-end delay of downlink traffic first Packet 187
Figure 6.19: First packet delay with respect to the controller location 189
Figure 7.1: Solution 1 - Traffic Offload Setup Call Flow Sequence 193

Figure 7.2: OMNeT++ module of the Traffic Offloading Switch 195
Figure 7.3: Solution 1 - TOS Configuration Procedure 200
Figure 7.4: Solution 1 - flow-table of the TOS Module 201
Figure 7.5: Logical Topology of the distributed cloud based offloading solution 1 and 2 203
Figure 7.6: Solution 1 - TOS Procedure to extract Uplink traffic from GTP tunnel and send it to the Cloud-Based infrastructure 205
Figure 7.7: Solution 1 - TOS Procedure to return the Cloud traffic back to the GTP tunnel between the SGW and the Serving eNodeB 207
Figure 7.8: Solution 1 - data plane operations 208
Figure 7.9: Handover Procedure when the source and target eNodeB connected to the same Traffic-Offloading-Switch 210
Figure 7.10: Handover Procedure when the source and target eNodeB connected to the different Traffic-Offloading-Switches 211
Figure 7.11: flow-table of the Traffic-Offloading-Switch OMNeT++ Module 213
Figure 7.12: Solution 2 TOS - Data Plane Uplink Traffic Processing Procedure 216
Figure 7.13: Solution 2 TOS - Data Plane Downlink Traffic Processing Procedure 217
Figure 7.14: Solution 3 - Logical Network Topology 221
Figure 7.15: Solution 3 - Control Plane Operations 223
Figure 7.16: Solution 3 - Cloud Offloading Application Operations 224
Figure 7.17: Solution 3 - Uplink Traffic Operations 226
Figure 7.18: Solution 3 - Downlink Traffic Operations 227
Figure 7.19: Solution 3 - UE Handover Procedure 229
Figure 7.20: Network Topology Simulated in OMNeT++ 231
Figure 7.21: Simulation platform File hierarchy 232
Figure 7.22: MME Application Internal Structure 233
Figure 7.23: Expirement1-UDPBasicApp end-to-end delay of the EPS with and without the traffic offloading Solution 239
Figure 7.24: Expirement1-PingApp RTT with and without Cloud Offloading Solution 240
Figure 7.25: TOS Virtual Port Statistics 242
Figure 7.26: TOS Processing Time 243
Figure 7.27: UE1(0) Serving Cell Id during the simulation 245
Figure 7.28: Time required to Handle Downlink Traffic Redirection 245

List of Tables

Table 5.1: Main Components of TEID Entry 132
Table 6.1: EUTRAN Configuration Parameters 165
Table 7.1: EUTRAN Configuration Parameters [88] 236

List of Abbreviations

3GPP	3rd Generation Partnership Project
AMBR	Aggregate Maximum Bit Rate
API	Application Programming Interface
APN	Access Point Name
ARP	Address Resolution Protocol
AS	Access Stratum
CAPEX	Capital Expenditure
CPU	Central Processing Unit
CUPS	Control Plane User Plane Separation
DMM	Distributed Mobility Management
EMM	Evolved Packet System Mobility Management
EPC	Evolved Packet Core
EPS	Evolved Packet System
E-RAB	Evolved Radio Access Bearer
ESM	Evolved Packet System Session Management
EUTRAN	Evolved Universal Terrestrial Radio Access Network
GBR	Guaranteed Bit Rate
GPRS	General Packet Radio Service
GTP	GPRS Tunnelling protocol
GUTI	Globally Unique Temporary Identifier
HSS	Home Subscriber Server
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
ID	IDentifier
IETF	Internet Engineering Task Force
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IT	Information Technology
LAN	Local Area Network
LCID	Logical Channel IDentifier
LTE	Long Term Evolution
MAC	Medium Access Control
MME	Mobility Management Entity
MPLS	MultiProtocol Label Switching
NAS	Non-Access Stratum
NAT	Network Address Translation
NC	Network Controller
NED	NEtwork Description
NFV	Network Function Visualization
NIC	Network Interface Card
OMNeT++	Objective Modular Network Testbed in C++

ONF	Open Networking Foundation
OPEX	Operating Expenses
OXM	Openflow eXtensible Match
PCC	Policy and Charging Control
PCEF	Policy and Charging Enforcement Function
PCRF	Policy and Charging Rule Function
PDCP	Packet Data Convergence Protocol
PDN	Packet Data Network
PDU	Packet Data Unit
PEGW	Packet Data Network Edge Gateway
PGW	Packet Data Network Gateway
PGW-C	PGW control plane
PGW-D	PGW Data plane
QoS	Quality of Service
RAN	Radio Access Network
RLC	Radio Link Control
RRC	Radio Resource Control
RRM	Radio Resource Management
RTT	Round-Trip Time
SBMCNA	Software Based Mobile Core Network Architecture
SCTP	Stream Control Transmission Protocol
SDEPC	Software Defined Evolved Packet Core
SDMA	Semi Distributed Mobile Management Architecture
SDMN	Software Defined Mobile Network
SDMNA	Software Defined Mobile Network Architecture
SDN	Software Defined Network
SDVMNA	Software Defined Virtualized Mobile Network Architecture
SGW	Serving Gateway
SGW-C	SGW control plane
SGW-D	SGW Data plane
SINR	Signal to Noise and Interference Ratio
SRB	Signaling Radio Bearer
TCP	Transmission Control Protocol
TEID	Tunnel Endpoint IDentifier
TFT	Traffic Flow Template
UDP	User Datagram Protocol
UE	User Equipment
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
VLAN	Virtual Local Area Network
VM	Virtual Machine
VoIP	Voice over IP
WAN	Wide Area Network
XML	eXtensible Mark-up Language
QCI	QoS Class Identifier
MOS	Mean Opinion Score
TOS	Traffic-Offloading-Switch (Modified Openflow switch)

CHAPTE 1

1 Introduction

1.1 Introduction

This chapter briefly describes motivation behind the research, followed by an explanation of some challenges of the Evolved Packet System (EPS) along with description of the open challenges facing the Software Defined Network-Evolved Packet Core (SDN-EPC) integration. It briefly describes the main contributions of the research and shows the used research methodology. Finally, it summarizes the other chapters of this thesis.

1.2 Motivations

In the recent years, the world has witnessed a massive growth in mobile broadband traffic, due to increasing numbers of connected devices, such as smartphones and tablets. Customers' expectations for high speed mobile broadband are on the rise as people rely more and more on real-time mobile applications, high quality video content, cloud-based services and expect to stay connected anytime, anywhere. Consider that the existing cellular network suffers from inflexible and expensive equipment, complex control plane protocols, vendor-specific configuration interfaces [1] and inefficient routing due to the centralized gateway, which has defects of long latency, data forwarding inefficiency, and user plane congestion.

The massive growth of mobile users' data traffic requires a significant re-design of the network's data and control plane infrastructures. In fact, now it is widely accepted that future cellular networks will require a greater degree of service awareness and optimum use of network resources. The motivation is to simplify the mobile network deployment

and operations by providing a programmable system that allows operators to deploy a new service or tune an existing one in a simple, flexible manner.

Mobile core network is a crucial part of the wireless network, and for this reason its virtualization/softwarization represents a challenging issue in the emerging mobile networks.

In the meantime, SDN continues to be the dominant topic of interest in networking today. It is proposed as a way to provide a centralized view and programmatically controlled networks, making it easier to deploy new applications and services, as well as tuning network policy and performance. This will make networks more adaptable and easily configurable in order to meet changing demands and operating environments. Open standards, such as OpenFlow, transform networking architecture and turn individual network elements into programmable entities. By decoupling the control and data planes, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications. As a result, highly adaptive, flexible, and scalable networks can be built [2]. SDN has been successfully implemented in data centres, Wide Area Network (WAN), and campus networks. The results have been promising but have as yet had little impact on cellular networks. Only recently,

SDN has attracted attention in the area of mobile communications networks. Extensive studies and research have been conducted in an effort to simplify the design and management of the existing cellular networks by introducing Control Plane User Plane separation (CUPS) and running the functions as applications on a logically centralized controller. This flexibility makes it possible to build network applications that are not only 3GPP compliant, but also implement innovative schemes from third party providers, such as location-based services and optimized content distribution and content caching.

1.3 Aims and Objectives

The aim of this research is provide comprehensive solutions to enhance cellular network in term of both design and performance, this includes: i) simplify the design and management of the network to reduce the deployment and operating cost; ii) increased capacity, improved data rate, decreased latency, and better quality of service. This research presents several Open-flow based solutions to simply the mobile network design while providing better performance. The adaptation of Software Defined

Networking along with the Network Function Virtualization in the proposed solutions play a major role to simplify the network design and deployment. Three Objectives are described below.

- The first part of this research investigates the signalling loads of several mobile procedures such as Initial-Attachment, mobile network entity failover and load balancing procedures. The system performance of the current mobile network during these procedures is presented and compared with the Openflow-based approach, more consideration is given to the structure of the Openflow protocol and what kind of extensions are required to improve the performance of Openflow-based mobile network.
- The second part focus on the hierarchal design and routing in the mobile network. and the inefficient routing. This part addresses the link utilization and how to optimize the use of the available reassure by demonstrating a way to reduce the overhead coming from using protocol like GTP.
- The third part shows the new services that can be adopted on the mobile network by using Openflow-based network. Cloud-based offloading is used to demonstrate one of many possible services that can be adopted in both 4G and 5G mobile network. The part describes in detail several challenges that comes with the Cloud-based offloading such as mobility and accounting

1.4 Challenges

The continuously increasing demand and expectation of the mobile network users make network operators boost their effort to evolve and meet the market demands with a wide range of services. This section summarizes the challenges of the current EPS that make it difficult to meet the new network requirements, moreover the challenges of the SDN-EPC integration is discussed. In EPS, the EPC entities are specialized hardware that interact between each other through standardized interfaces. They require static deployment, provisioning, and configuring. As a consequence, the openness of the network architecture is extremely limited, which makes it lack in flexibility, dynamicity, or on-demand features. Therefore, it is very difficult to introduce a new service by the network operation and very expensive to upgrade the network. In EPS, the PGW is responsible for the IP address allocation of the UEs and acts as its gateway to the IP networks. The UE gateway is not changed even if a UE moves a long distance,

which introduces inefficient routing due to the centralized gateway, since this hierarchical design requires the UE traffic to traverses the core network even if some data traffic is going to/from local application servers, e.g. enterprise cloud servers [3]. Therefore, EPS has defects of long latency, data forwarding inefficiency, and user plane congestion. In EPS network entities restoration procedure is not transparent e.g. SGW failure procedure requires the release of access bearers of all the UEs served by the failed network entity and waits for the UEs to initiate service request procedure to restore the service, which causes service interruption and significant amount of signalling. Also, a weighting factor is used for traffic distribution, which makes it difficult to optimize network resource by considering network conditions and service requirements holistically. Researchers in both academia and industry consider SDN-based solutions for the next mobile network, considering that SDN is a novel approach that gained great momentum and interest. It separates the control plane from the data plane and uses open interfaces to increase system programmability. There are different points of view on where to deploy SDN in the mobile network, but the foremost attention goes towards deploying SDN-based core network. This adaptation is still at an early stage and different conceptual and technical solutions are proposed by researchers. These SDN-EPC integration proposals accommodate many challenges that include:

- Ability of the proposed solution to be scalable and handle the communication between the data and control planes without introducing a single point of failure in the system. Moreover, the ability of the centralized controller to handle a large network.
- Backward compatibility with the current LTE standard, and this point looks like it is either dismantled or intentionally ignored by the researchers.
- OpenFlow does not support GTP operation and researchers are divided into two parties, first extend OpenFlow protocol to support GTP operations and second eliminate GTP and reduce the added overhead.
- Ability to provide an efficient and optimized way to handle handover considering that UE mobility is an important character of the mobile network.
- Provide effective load balancing by distributing network load efficiently between the data plane Forwarding Devices (FDs).

It is of critical importance to address these challenges, and although the researchers were diligent in this respect, most of them have produced no performance validation of their proposed architectures or their findings have not been sufficiently strong to provide practical suggestions, whereas other works evaluate their proposals using a proprietary testbed or using a platform that is privately owned and created by the authors. Thus, there is room for an open, more generic, flexible and powerful tool to evaluate these integrated networks.

1.5 Contributions

This thesis discusses how SDN, specifically OpenFlow protocol, can play a major role in enhancing the existing LTE networks by providing programmability, flexibility and ease of network management for operators. The contributions of this research as following:

- Extend OpenFlow protocol to support GTP operations and design SDN oriented mobile core network called Software Based Mobile Core Network Architecture (SBMCNA). SBMCNA discusses the required modifications within the EPC in order to overcome some of the limitations of the current EPS. Load balancing and resiliency were used to demonstrate the capability of the proposed system to reduce the signalling load. The performance of the system using three different methods to support GTP operations is compared and evaluated. Then presents the benefit of extending OpenFlow plugin to build an intelligent FD that can handle new set of messages designed specifically for the mobile network. Finally, demonstrate the benefit of using SDN agent in the OpenFlow FD to enhance system performance in term of reducing signalling loads and offload some of the controller tasks to the FDs.
- Presents new mobile core network architecture, called Software Defined Evolved Packet Core (SDEPC), whose inherent SDN characteristics to provide an abstraction layer separating the control plane from the underlying user plane. The architecture proposes an SDN-based transport network to handle user traffic during mobility without implementing a GTP which is used by 3GPP standards. SDEPC improves system performance in terms of end-to-end delay, packet loss ratio and bandwidth utilization. Removing GTP introduce backward

compatibility issue, therefore, a hybrid network that can operate with and without GTP is built to keep the interoperability with legacy networks.

- ❖ Present three different solutions to implement selective traffic offloading to a cloud-based infrastructure assumed to be available in the near proximity from the mobile access network, then model the proposed solutions in OMNeT++ and compare the performance of these solutions.
- ❖ Build OMNeT++ model to simulate software based mobile core network. The simulation model has the capability to simulate all the aforementioned networks and it is flexible enough to be extended to test new ideas and protocols.

1.6 Methodology

The rapid growth of mobile users' demands and Capital Expenditure (CAPEX) and OPerational EXpenditure (OPEX) costs for network operators to innovations in the mobile network to cope with their users' demands represents the biggest motivation for this thesis to evaluate the advantages offered by SDN-EPC integration. The hierarchical SDN-EPC integration trend in this work starts from overcoming the high signalling overhead, load distribution and SGW failover of EPS network as the first challenge, then moves towards inefficient data forwarding and interrogability, and ends with selective traffic offloading to cloud based infrastructure. In this thesis, the aforementioned challenges are discussed in detail, and then illustrations of existing solutions are presented, followed by our own solutions to improve the performance and avoid the limitations and difficulties that may degrade the system's performance, as well as comparing the performance of our solution with other solutions if required.

Software Based Mobile Core Network Architecture is proposed to overcome the first challenge and SDEPC is used to provide better performance and more efficient routing to provide better latency. Finally, three solutions are compared to provide selective traffic offloading. Simulation-based experiments are used to evaluate and validate the performance of the proposed systems. The proposed systems are modelled in OMNeT++. The model includes: centralized SDN controller with a set of mobile network function applications. The data plane FDs are simulated by a combination of standard and modified OpenFlow switches. A number of different scenarios and case studies are designed and simulated, and the outcomes are compared and evaluated.

1.7 Publications

- OpenFlow 1.3 Extension for OMNeT++
  - **Published in:** 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing
- Simulation and Performance Analysis of Software-Based Mobile Core Network Architecture (SBMCNA) using OMNeT++
  - **Published in:** 2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)
- Software Defined Selective Traffic Offloading (SDSTO)
  - **Published in:** 2018 IEEE 23rd International Workshop on Computer Aided Modelling and Design of Communication Links and Networks (CAMAD)

1.8 Thesis Organisation

This thesis consists of a total of eight chapters, beginning with an introductory chapter to outline the reason behind the research, the challenges and the methodology of the research. Each chapter starts with an introduction and ends with a summary. The chapters are independent, and the readers are advised to follow the right order to fully understand the presented ideas in the thesis. The structure of the remainder of the thesis is organized as follows:

- **Chapter 2**: conducts an extensive survey to present the relevant literature on the adaptation of SDN in the mobile core network. These works include a proposal to keep GTP as the main data plane protocol, and another to eliminate the GTP tunnel. It also shows the differences between our work and the others.
- **Chapter 3**: briefly describes the difference between traditional network and SDN, then it describes OpenFlow protocols and its switch design structure based on different version of the specifications. It also shows the OMNeT++ model built to simulate Openflow-based network.
- **Chapter 4**: introduces EPS technology and its main building components. It briefly describes the communication interfaces, protocols, UE procedures. Then

presents and discussed in detail the design structure and functionality of the OMNeT++ model that used to simulate the control plane of EPS network.

- **Chapter 5**: Review one of the SDN-EPC integration proposals and identifies a few issues. Then discusses new network architecture with three methods to support GTP in OpenFlow switch, followed by a comparison between the systems performance handling SGW fail-over and traffic distributions.
- **Chapter 6**: represents the second contribution of our research. It shows the benefit and the advantage of eliminating GTP and using the centralized control plane of SDN with a layer 2 technology to handle UEs traffic forwarding and maintain it during handover. It also describes major issue with removing GTP and proposed a solution.
- **Chapter 7**: describes the third contribution which focus on selective traffic offloading to a distributed cloud infrastructure. Three different solutions are discussed and compared in this chapter.
- **Chapter 8**: concludes the works presented in the thesis by summarizing the research finding and points out potential future work.

CHAPTE 2

2 Literature Review

2.1 Introduction

The world is witnessing a rapid growth in mobile data broadband, where users expect to be connected anywhere and anytime. Cisco predicted [5] that global data traffic generated by mobile network will increase 8-fold between 2015 and 2020. Mobile operators are forced to handle the significant growth in data traffic in very resourceful ways due to the design of the existing mobile network. Many works proposed to use Software Defined Networking (SDN), Network Function Virtualization (NFV) and cloud computing as enabling technologies to design a new scalable and flexible cellular network, which provides fine-grain control for network management, while keeping costs to the minimal. SDN is an emerging technology and there are many opinions on how to integrate it with the cellular network. This includes whether to use it to build a new slate design like the works presented in [6][7][8][9][10] or use as extension to the existing mobile network to improve the network performance like the works presented in [11][12][13][14].

Other works proposed to replace control elements by centralized software to provide on-demand connectivity and services such as the works presented in [4][15][16], whereas the works presented in [17][18][3][19] focus on the integration of SDN and NFV to overcome the limitations of the existing mobile network to make possible upcoming 5G networks. Also, the authors of [20] focuses on the realization of on-demand cloud-based mobile core network to provide EPC as a service. The authors of [21] proposed to use SDN to provide Layer 2 based backhaul network. Meanwhile, the works presented in [22] [23] focus on enhancing the user experience by utilizing SDN. The authors of [22] introduce virtual mobile core network built by utilizing

technologies like SDN, NFV, and MobileVisor to provide control slicing which allows multiple operators to use the same core network, which helps to reduce the deployment and operation costs as well as use network resources efficiently, while the authors of [23] used multimedia services provided by the IP Multimedia Subsystem (IMS) to demonstrate the advantage of SDN-based mobile core network to the user experience. The authors of [24][25][26][27][28][29] proposed the redesign of the existing mobile network to provide Distributed Mobility Management (DMM) by utilizing SDN and NFV. The surveys conducted by [30][31] describe the development of Software Defined Mobile Networks (SDMNs) and present several research directions. The rest of the chapter is organized into 4 sections namely: i) Software-defined Mobile Network Architecture (SDMNA); ii) SDN as an add-on; iii) Software Defined Virtualized Mobile Network Architectures (SDVMNA); iv) Cloud Radio Access Network (RAN) and SDMN Security.

2.2 Software Defined Mobile Network Architecture

This section describes SDN-based network architectures, which redesign the current cellular network components for providing scalable and flexible cellular networks. These architectures use Openflow as the southbound protocol. This section is divided into two sub-sections, the first sub-section describes network architectures that assumes or defines extensions to the protocol to support GTP operations, while the second sub-section includes ”new slate” network architectures that eliminate the GTP tunnelling and network architectures that eliminate the GTP tunnelling only over the S1 interface to provide layer 2 based backhaul. The authors of [32][33] specify the control plane complexity and inflexibility of the current cellular network that makes it difficult to evolve the system to address the future requirements of the users. The authors argue that SDN-based cellular network provides simplified cellular data network design with ease of management, while enabling new services.

In [32] the authors describe the challenges of the pure SDN-based mobile network to support a huge number of subscribers, handle their mobility and provide fine-grain measurement. Then they proposed to extend the SDN controller, switch and the eNodeB to include more features and functionality to address the aforementioned challenges. These extensions include i) policy-based forwarding that utilizes subscribers’ attributes; ii) local agent in each switch; ii) Deep Packet Inspection (DPI).

In [33] the authors focus on inter-operation with other wireless network technologies and other operator networks. The authors describe the benefit of the SDN-based cellular network to support flexible resource slicing based on the subscriber's attributes rather than the header fields of the data traffic. The authors describe how the centralized radio-slicing can provide isolation between the Radio Resource Management (RRM), admission control, and mobility management of each slice. The work presented in [34] is built upon the works presented in [32] and [33].

The authors present their envisioned SDN-based cellular network design that reshapes the current mobile network and provides a backward-compatible modular cellular architecture. The authors argue that their architecture will reduce the operating cost because the centralized control plane helps to reduce the network maintenance and upgrade costs. The authors used UE mobility management as a case study to demonstrate the benefit of the proposed architecture. The authors claim that their architecture: i) enhance the power consumption of the UEs, which increase the increase the battery life; ii) provides better mobility management which reduces the signalling loads; iii) improve the handover delay. The authors think that the proposed architecture will reduce the signalling loads by 50% considering the number of mobile subscribers worldwide has exceed 6 billion in 2012 based on Infonetics Research on the Total Fixed and Mobile Subscribers, April 2102. However, no actual validation is provided.

Meanwhile the work presented in [35] and [1] describes scalable SDN-based cellular core network architecture known as softCell. The authors emphasize on the improvement offered by the softCell architecture to the cellular system scalability and flexibility. In [35] and [1] the authors present the softCell architecture and describe how to connect the unmodified UE to the Internet without the need for specialized complex networking elements or complicated protocols to handle the setup and management of users' traffic forwarding.

SoftCell consists of SDN controller, Access switch attached to each eNodeB, core switches and commodity middleboxes such as dedicated appliances, virtual machines, or packet- processing rules on switches as shown in Figure 2.1. The authors describe the challenges of the SDN-Based Cellular Core Network that includes: i) support fine-grained service policies with small switch tables; ii) support fine-grained packet classification in asymmetric topology; iii) scalable handling of network dynamics. The authors proposed to use multi-dimensional aggregation, smart access edge, dumb gateway edge and smart local agent at base stations to address these challenges.

The authors used floodlight controller [36] to implement SoftCell prototype and benchmarks the performance of the SDN controller and the SDN agent in small and large-scale simulations. The obtained result shows that SoftCell achieves scalability in both control and data plane by offloading packet classifiers and policy tags at local agents of the FDs and pushing packet classification to low-bandwidth access switches respectively.

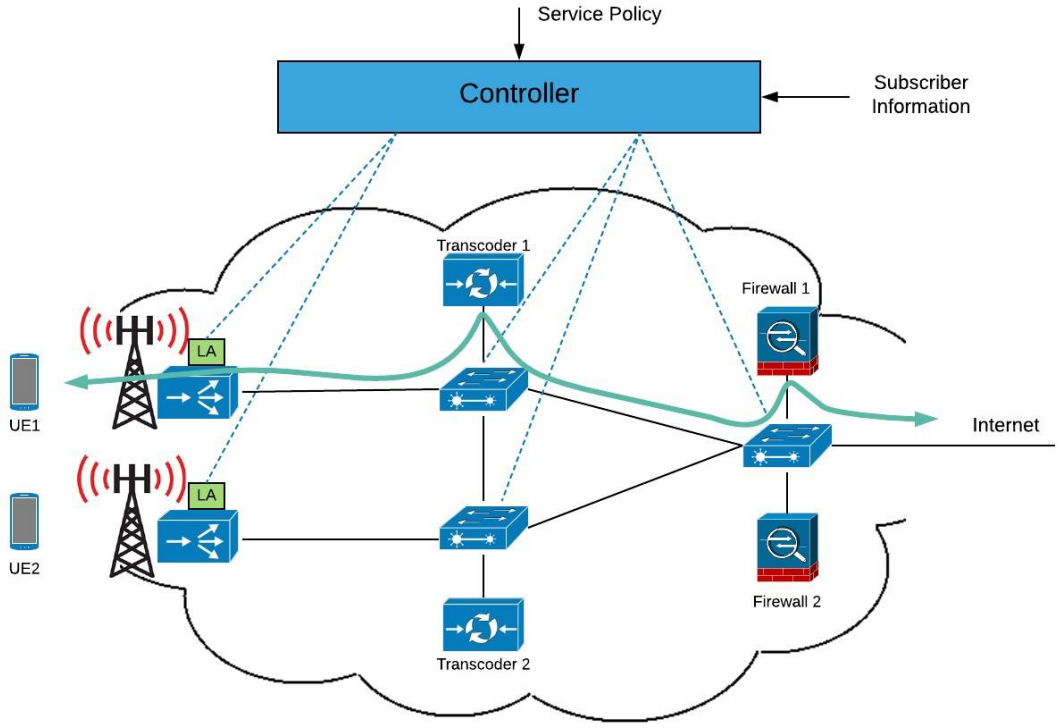


Figure 2.1: SoftCell Network Architecture proposed by [35][1]

2.2.1 SDMNA with GTP Extensions

The authors of [4] think mobile networks can cope with the increasing demands of their customers and ever-changing network conditions that include sudden congestion, network device failure by using on-demand connectivity service. The authors described how LTE handles several procedures that include: i) service request procedure; ii) resiliency; ii) load balancing and specified the drawbacks of these procedures, that include: i) systematic establishment of the EPS bearer even when there is no data to use the bearer; ii) TEID value of the GTP tunnels is locally allocated by each node; iii) complicated SGW failover procedure that involves releasing the bearers of all UEs and waiting for the UEs to re-initial the service request procedure, which produces more signalling loads in the system; iv) load distribution based on a pre-configured weighting

factor and unaware of the current network load. Therefore, authors argue that LTE/EPC lacks the network visibility and control elasticity that enables the on-demand connectivity service and proposed to use SDN based network to provide the tools and possibility to overcome these drawbacks. The authors proposed an OpenFlow-based control plane for LTE/EPC architectures as shown in Figure 2.2.

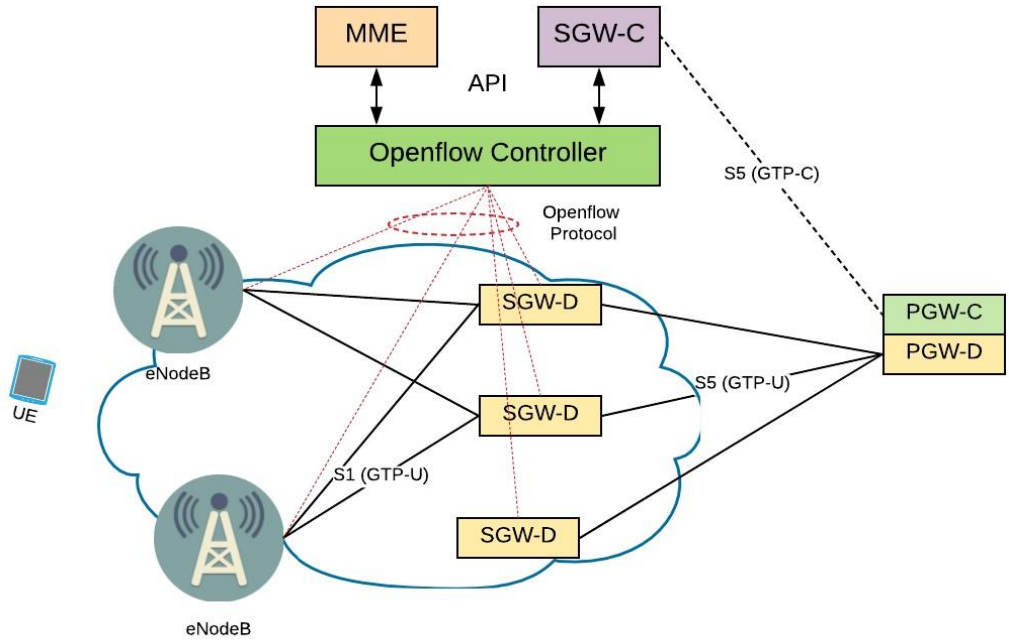


Figure 2.2: On-Demand Connectivity Architecture proposed by [4][15]

It basically consists of OpenFlow-enabled eNodeBs, set of OpenFlow switches, SDN controller, and PGW in which the MME and the control plane of the serving gateway (SGW) are running as an application on top of the controller. The SGW application communicates with PGW using the same primitives specified by the 3GPP. The authors proposed to substitute the eNodeB S1-MME interface with OpenFlow switch to send the UE control message using OpenFlow Packet-In to the SDN controller. In the proposed architecture the SGW control plane is centralized and uses the OpenFlow protocol to remotely manage the SGW data forwarding plane. This separation is built into the system to ensure that the on-demand connectivity service operates even in critical situations such as network equipment failure and overload situations. The authors used the service request procedure, resiliency, and load balancing to demonstrate the advantage of the proposed architecture and show how it can guarantee the on-demand connectivity service. In this architecture the first packet of each flow is sent to the controller and based on the session type (derived from the first packet headers), network loads and statistics, forwarding rules are installed to forward the

session utilizing the best available routing path to guarantees the on-demand connectivity service.

Also, work proposed by [15] uses the same network architecture presented in [4], the authors think that the programmability offered by this architecture will help network operators to cope with the new demands and overcome most of the new network usage. Signalling cost analysis derived from mathematical calculation is used by the authors to show and evaluate the boost in system performance to the standard LTE/EPC implementation. Several use cases are utilized in this work. This includes i) Initial-Attachment; ii) Access-Bearer-Setup; iii) Access-Bearer-Release procedures. The preliminary result shows that using OpenFlow can reduce the signalling load for UE S1 and S5 and data bearer parameters are kept in the network equipment during the application IDLE period because the controller is responsible for setting the optimal release timer for each flow entry to help reduce the signalling load and memory space usage in network equipment.

While the architecture presented in [16] used an approach that is similar to the works of [4][15]. The authors proposed an OpenFlow-enabled mobile core etwork called OFEPC. In this approach, the control plane of all EPC entities is centralized and running as applications on top of OpenFlow controller. The only difference between this approach and the work proposed by [4][15] is the control plane and data plane separation of the PGW as shown in Figure 2.3.

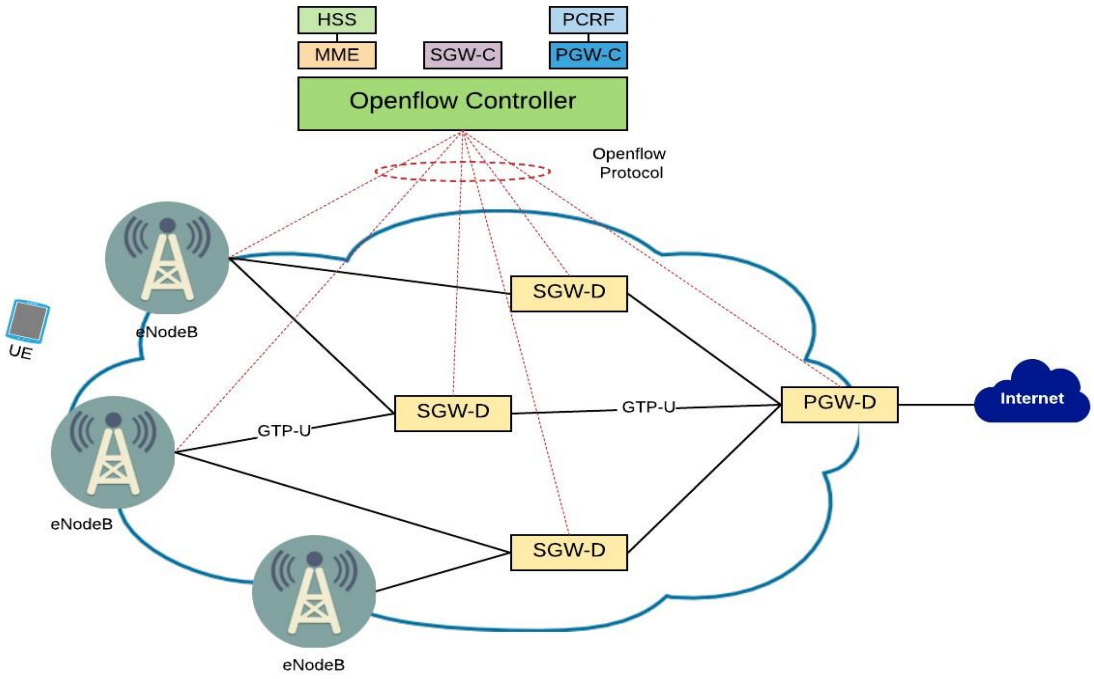


Figure 2.3: OpenFlow-enabled mobile core network Architecture proposed by [16]

The authors think that the centralization of the PGW control plane will feather enhance the system performance compared to [4][15]. The authors used initial attachment, UE-triggered service request, network-triggered service request, handover, and the tracking area update procedure to demonstrate the boost in performance offered by the new architecture. The authors think that their proposal increases the flexibility of the mobile core network and offers a wide range of innovations for the operator without the need for the vendor intervention whilst also the proposed architecture eases configuration and management. Numerical results derived from mathematical equation is used to show the reduction in signalling load in comparison with the traditional LTE/EPC architecture and the work proposed by [15].

The authors of [37] present an evolution to the mobile EPC to simplify the configuration and maintenance and eliminate the distributed IP routing control plane. The authors proposed to pull out the control plane of EPC entities and run it on virtual machines in a data centre separated from the data plane implemented by a mesh of OpenFlow switches that enhanced with GTP routing extensions. The authors explained in detail the required extension to OpenFlow 1.2 switch design to allow EPC gateway nodes to support GTP encapsulation and decapsulation along with the OpenFlow protocol modification required to allow flow routing based on the GTP header field and Tunnel Endpoint Identifier (TEID). The authors used Selective Flow Routing for In-Line Services, Multihomed Terminals, and Security Isolation of Mobile Terminals use cases to demonstrate the advantage of the proposed architecture.

The authors of [38] present another method to support GTP operation in SDN-based core network design. The proposed solution does not require an extension to OpenFlow protocol or upgrading the switch design. The authors explain the operation of the GTP in term of packet structure, the processing mechanism, and processing delay. Then they proposed to carry TEID value in the source IP source field of the network layer header. This way the key field (TEID) of the GTP layer is visible in network layer header and can be parsed by current protocol and routing devices. An experiment is conducted by the authors to measure the feasibility of the proposed architecture and verify the possibility of parsing higher-layer field in a lower-layer. The obtained processing delay of the conducted experiment is high at approximately 75ms, but the authors argue that the delay is scale independent.

Meanwhile the work presented in [39] discusses the potentials and limitations of a possible 5G network that integrates SDN technology, specifically OpenFlow protocol,

with the existing mobile networks. The authors describe the benefits of this integration in terms of increasing network flexibility by providing network-wide programmability. Also, the authors highlighted some open challenges that include: GTP extension, scalability, interoperability, mobility, and routing. The authors used the NS3 simulator to build software realizing of the SDN-LTE integration. The authors used a combination of OpenFlow 1.3 module with NS3 LTE module to build a simulation environment for the SDN-LTE integration. The authors extend OpenFlow protocol to support TEID matching and introduce a specialized EPC controller to build a scenario that consists of OpenFlow based backhaul network to communicate with the EPC MME to evaluate how the SDN-LTE can provide better source utilization.

Meanwhile authors of [40] focus on providing better load distribution while reducing the signalling loads. The authors propose to use flow-based load balancing by utilizing the same SDN-Based architecture proposed by [4][15]. The authors wrote load balancing algorithms that utilize the network stats gathered from the network data plane FD every 1s. An experiment is conducted by the authors using mininet to demonstrate how the combination of SDN-based core architecture with their proposed load balancing algorithm can provide better traffic distribution between data plane nodes, which preserved the high throughput and decreased latency among the new network node.

2.2.2 SDMNA without GTP

2.2.2.1 Disruptive cellular network architecture

The works presented in [6][7] represent an attempt to boost the performance and increase the elastic scaling of the mobile network to improve the network throughput to cope with the increasing demand in mobile data traffic. The authors think that disruptive redesign of the current mobile network is the solution to fulfil the future throughput requirements, therefore they proposed to revamp the mobile backhaul to introduce flatter software defined mobile architecture that includes MME functionality as part of the SDN controller as shown in Figure 2.4.

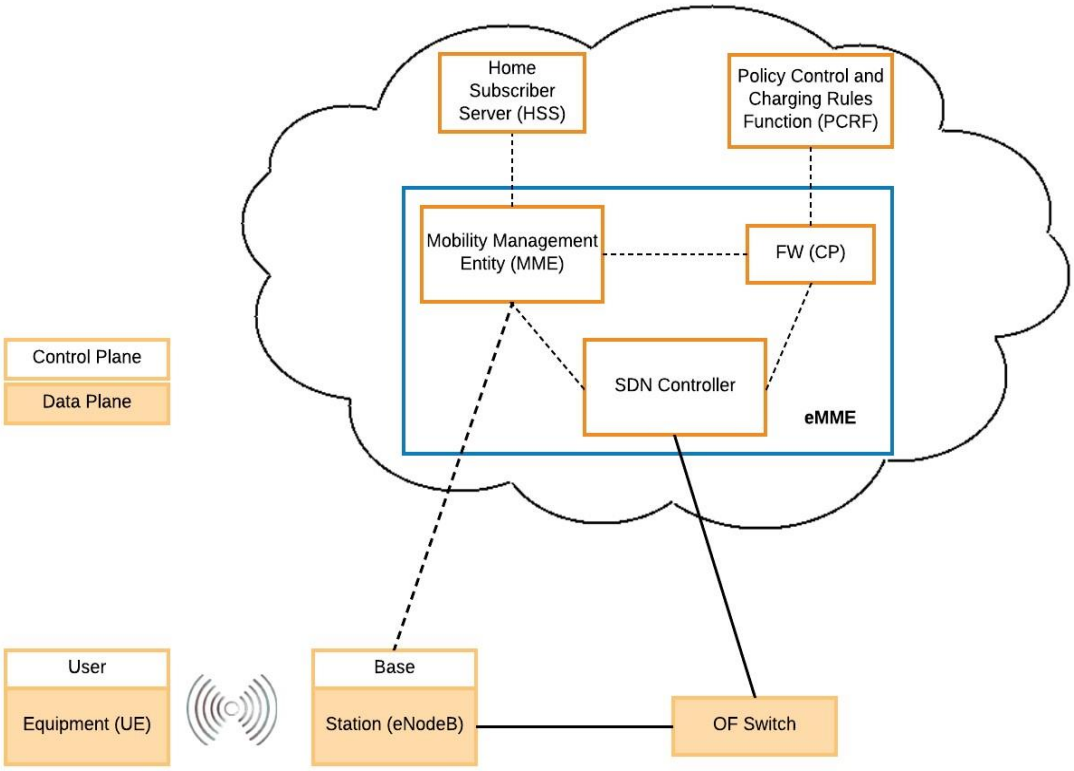


Figure 2.4: Disruptive integration of SDN with MME proposed by [6]

The authors describe multiple designs to illustrate the proper migration path and ensure reasonable transition phase and concluded that simplified access network formed by base stations i.e. eNodeBs, interconnected through a backhaul network composed by SDN switches managed by the network controller is the best solution. Moreover, this architecture allows the virtualization of mobile network elements, which increase the network flexibly and availability considering that multiple instances of these control functions can be launched in data centres to manage the simplified data plane. The authors think that their architecture simplifies both the control and data planes considering that the transport network is based on SDN switches, while the control planes of the LTE entities such as MME and S/PGW are merged with SDN controller. Moreover, the author proposed to remove GTP and uses 802.1ad in the backhaul to simplify the data plane protocol stack. The obtained results show that removing GTP reduced overhead and prevent fragmentation which helps to improve the network throughput.

However the work presented in [8] and [41] can be considered as a complementary to the work proposed by [6] and [7], where the authors of [8] explained in detail the building blocks of the proposed architecture, this includes: i) the SDN controller

applications e.g., virtual eNodeBs, backhaul transport, Mobility management, Access, Caching, Monitoring, and Services delivery; ii) data plane transport network that consists of simplified access devices and Carrier Ethernet Switches. The authors discuss the best way to migrate to their proposal without unnecessary replacement investments. Experiments are used to show how it is possible to realize the migration of 3GPP to SDN-based mobile network and demonstrate the effectiveness of their proposal. In the experiments, OpenFlow protocol is used as the southbound protocol between the controller and switching devices. Moreover, WIFI offloading scenario is presented by the authors to show the feasibility of SDN in LTE networks.

Whereas the authors of [41] paid more attention to the integration of technologies like Software-defined Networking and Network Functions Virtualization as a basis for the 5G network architecture. The authors emphasise on the special requirement of the Internet of Things (IoT) and video services in terms of efficient network resource operability and dynamic scalability. In the proposed architecture network elements are virtualized and running in the cloud, which may cause a reliability and robustness that needs to be addressed. The migration of control plane VM due to hardware failure or the need to extend the available resources can increase the delivery time and affect the overall latency of the system. A testbed of the proposed architecture is built by the authors and submitted as European Telecommunications Standards Institute Proof of Concept (ETSI PoC). This work demonstrates how SDN can simplify the data plane of the mobile network and provide more efficient routing and data forwarding while reducing the cost.

Another SDN based mobile core network solution is proposed by [42], the authors proposed to use the centralized controller to handle the management of user traffic and mobility and thereby eliminate the need for the GTP. The authors focus on the user mobility and used intra-LTE and intra-RAT scenarios to describe the advantage offered by the proposed solution in terms of signalling cost, tunnelling cost, handover latency and scalability. The authors used mathematical equations to calculate the evaluate the performance of their solution, like [4][15][16], they also assumed that the data plane device can be configured by the single packet-out message.

The authors of [43] also think the LTE network is excessively complex to manage because it uses GTP, which is unduly complex and requires an important amount of signalling to manage. Also, deploying a new technology in LTE is very difficult, therefore integrating SDN with the LTE network has the advantages of control and data

planes separation and eliminates the need for tunnelling protocols like GTP, therefore it enables network programmability and reduces the time required for the deployment of new services. The authors used three different handover scenarios to demonstrate the enhancement offered by their solution, this includes i) X2 Handover without SGW relocation; ii) Intra-Domain Handover with SGW relocation; iii) Inter Domain Handover with EPC controller (EPC-CTR) relocation. The authors used mathematical equations to calculate and compare system's performance and the preliminary results show that the proposed solution has the potential to improve the efficiency of the core network.

2.2.2.2 Layer 2-Based Backhaul

Other approaches targeting localized mobility management are proposed in [44][21]. These approaches utilize layer 2 based forwarding in the backhaul network between the core and access network and eliminate the use of GTP-U over the S1 interface.

The authors of [44] proposed to use a carrier Ethernet network with a modified version of TRansparent Interconnection of Lots of Links (TRILL) to hide the UE mobility from the core network to reduce the signalling load between the access and core network. The authors assume that the backhaul network consists of two sub-networks, namely: access-subnetwork and aggregation sub-network. A modified version of TRILL is used in the backhaul access sub-network and Carrier Ethernet is used in the backhaul aggregation sub-network. The authors think that their approach requires minimal change to the existing LTE entities such as the SGW and the eNodeBs and at the same time will be the enabler for the deployment of small cells without affecting the signalling loads. Also, the proposed architecture provides better resource utilization since it enables efficient multipath switching and eliminates the GTP-U overhead.

Another approach is proposed by [21]. The authors think that the deployment of the SDN-based in the backhaul access sub-network will offer programmability and increase the network flexibility. The authors proposed Semi-Distributed Mobile Management Architecture (SDMA), which has a localized mobility application running on top of the SDN controller of the backhaul access sub-network. The application is used to handle some of the UE handovers without the need to contact the EPC network. The proposed architecture eliminates GTP and uses the centralized mobility control plane application to improve the data transportation efficiency and offer distributed mobility anchor points. Numerical calculation is used to demonstrate the enhancement

offered by the SDMA in terms of Handover signalling load and latency. The outcome shows that SDMA reduces the signalling loads sent to the EPC by 33% and enhances the handover latency by 86%. This solution will increase the processing latency of the Initial Attachment procedure because the control messages exchanged between the eNodeB and the EPC are processed by the SDN controller of the backhaul access sub-network to make the disturbed mobility anchors possible. Also, new signalling load is generated between the SDN switches and controller in the backhaul network.

2.3 SDN as an Add-on

This describes network architectures that adds SDN based solution to current cellular network components with or without any change to the structure and functionality of the cellular network components. This section is further divided into two sub-sections, selective traffic offloading and caching and DMM. The first sub-section describes solutions that aim to enhance network performance by moving content near to the users or selectively redirect user's traffic to the cloud, while the second sub-section describes works that utilize SDN to offer DMM and overcome the centralized data plane forwarding of the current cellular network.

2.3.1 Selective Traffic Offloading and Caching

The authors of [11] proposed lightweight Mobile Cloud Offloading Architecture (MOCA) that utilizes Software-defined networking with an in-network cloud platform to realize traffic offloading that can be easily adopted by mobile network operators without requiring significant changes to their networks. The authors think that SDN, cloud technologies, and minimum change to the current LTE architecture are the best mixture that can improve the mobile network architecture to increase its flexibility and provides a solution to overcome the sub-optimal routing used in LTE that causes a problematic issue for delay sensitive applications like online gaming. The main component of the proposed architecture is shown in Figure 2.5.

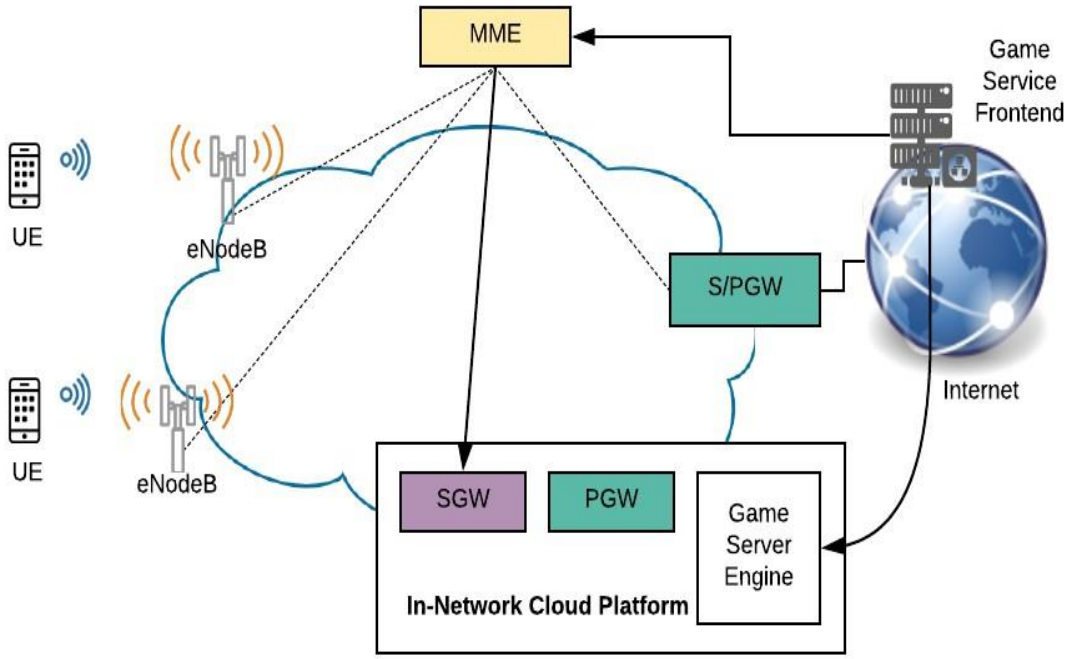


Figure 2.5: Mobile Cloud Offloading Architecture (MOCA) proposed by [11]

The authors assumed that commercial in-network cloud platform is distributed in near proximity to the mobile access network to provide services like online gaming etc. This platform introduces extensions to the Mobility Management Entity (MME) in terms of functionality and control messages structure. The authors assumed that the mobile network can initiate on-demand S/PGW virtualized instance on the cloud platform. In order to perform the traffic redirection, the authors change the operation of the virtualized SGW to enable the on-demand redirection of the default bearer. The authors used commercial testbed to build a proof-of-concept of the proposed architecture and describe several aspects that are related to handling the mobility nature of the UEs.

Similar to [11], the work proposed by [12] used cloud-based traffic offloading to overcome the sub-optimal routing of the LTE architecture. The authors provide another angle for the cloud offloading realization without the need to make any modification to the current LTE architecture. The proposed Software-Defined Networking Mobile Offloading Architecture (SMORE) is shown in Figure 2.6. The authors added SMORE controller, SMORE monitor, Database, SDN switch, and in-network cloud platform to the LTE architecture to realize the traffic offloading solution and reduce the end-to-end delay of the user's traffic. Online gaming traffic is used as an example, but the solution is applicable for more types of traffic.

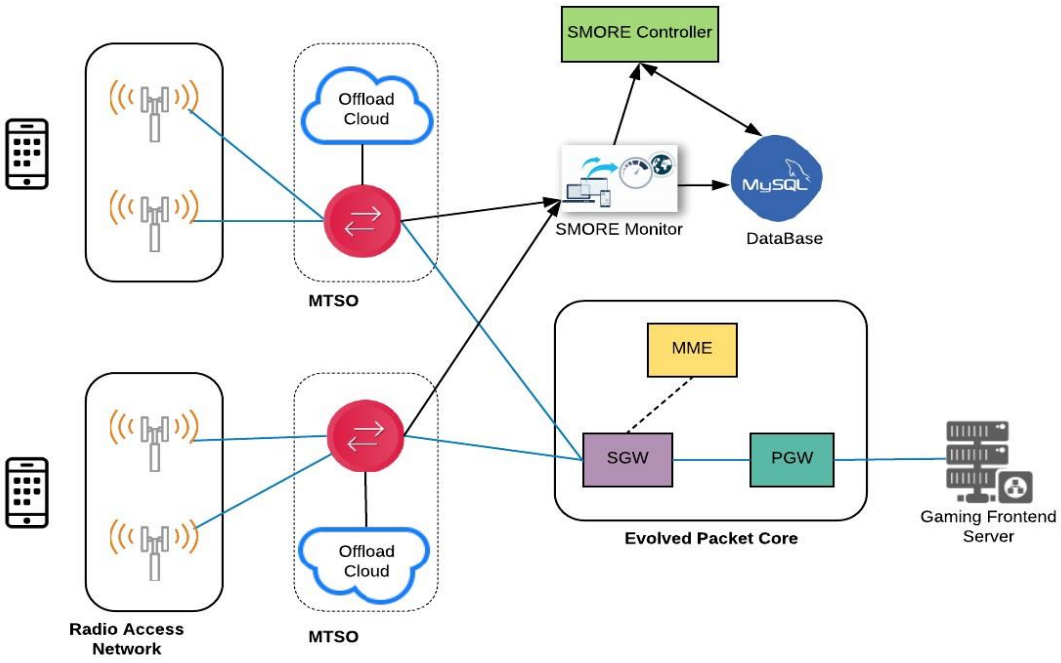


Figure 2.6: Software-Defined Networking Mobile Offloading Architecture (SMORE) proposed by [12]

In SMORE, the SDN switches are deployed in regional aggregation points, and the SMORE monitor is responsible for monitoring the LTE/EPC control plane signalling, extract the required information to be stored in the database and provides triggers for the SMORE controller about the UEs activities like the attachment and handover procedures. The SMORE controller configures the SDN switches to route the traffic between the UE and the cloud, and the controller also reconfigures the SDN switch during handover. The author shows the design of the individual entities of the proposed architecture and explains in detail the interception and rerouting of the traffic to offloading servers located inside the cellular core. The authors build the proposed architecture on a testbed that utilizes OpenEPC LTE/EPC architecture implementation [45] and Open vSwitch (OVS) 2.0 which supports the OpenFlow 1.3 standard. The authors used processing time and Round-Trip Time (RTT) to show the enhancement offered by their proposal and at the same time illustrates the extra processing time required to handle the offloading operations.

MobiScud [13] is an evolutionary architecture to the SMORE proposed by [12]. MobiScud integrates cloud and SDN technologies with the standard mobile network. It assumes that user-specific private VM is instantiated in a cloud-based platform distributed in the RAN, and the VM is maintained as users move around by utilizing

live virtual machine migration. In MobiScud, SDN capabilities are utilized to offload low-latency, compute-intensive applications to the private VM instances. MobiScud provides the tools and techniques to monitor the control plane message exchanges between the UEs and the mobile core network. During handover, MobiScud utilizes this information to migrate the private VM to the best cloud location that is near to the UE new location, at the same time, the flow forwarding rules in the SDN data plane devices are updated to ensure seamless handover without causing disruption to the UE's ongoing connections. A prototype of the MobiScud architecture is implemented in PhantomNet testbed and the RTT is used to evaluate the system performance with and without the MobiScud.

Another traffic offloading solution is presented in [46], where the authors used a mixture of Network virtualization, Software-defined Networking, and cloud computing technologies to pave the way for a virtualized 5G cellular network. The authors used virtualized-based EPC gateway with dedicated packet processing hardware to dynamically redirect the UEs active session between a cloud environment and a fast path. The gateway utilizes dynamic GTP termination to relocate the UE mobility anchor, which enables elastic usage of the user plane resources and provides user plane processing resources on-demand for each User Equipment (UE). The authors used OpenStack, SDN controller that supports GTP tunnel switching, and a dedicated fast path device to build a prototype for the proposed solution to realize the differences between dedicated fast path and cloud-based processing and the outcome result shows that their solution can mobile user's operations without incurring significant increases in latency or reducing throughput.

The work presented in [14] represent an attempt to increase the network flexibility and provide the tools to effectively deal with the fast-increasing amount of data traffic by incorporating SDN-based mobile network. The authors proposed a new architecture that is compatible with the current 3GPP architecture. The authors proposed transparent in-network caching by utilizing SDN in the mobile network backhaul, and the operation of the proposed solution is explained in detail by the authors to demonstrate how this scalable solution is compatible with the existing backhaul architecture. Basically, the authors used MPEG Dynamic Adaptive Streaming over HTTP (DASH) content's transparent caching to demonstrate the functionality of the proposed architecture. The proposed architecture includes a mechanism to intercept cacheable contents by inspecting the HTTP GET request sent in a TCP segment over GTP tunnel, and then it

extracts the request from the GTP tunnel and sends it to the cache server. The data received from cache server is inserted back to the GTP tunnel so that the caching remains transparent to the UE. A prototype implementation is used to demonstrate the overhead of the proposed solution in-network caching solution and the result shows minor impact on the streaming clients' behaviour.

Meanwhile the authors of [47][48] think that caching is the most evident solution for the network operators to meet future demands at a manageable cost. The authors proposed to bring the contents closer to the users by utilizing an in-network caching solution. Software Defined Mobile Network (SDMN) architecture proposed by [6] is used to build the proposed in-network cashing at the backhaul of the mobile network. The authors present an efficient and dynamic content delivery solution and present the benefit of the proposed solution for both network operators and the end users. The authors proposed an ultimate multi-stage cache, in which cache are collocated in every base station and other caches spread over different points of presence in the backhaul network to reduce the traffic passing through the core network. SDN is utilized to optimize the caching location by dynamically relocate the content from one server to another, which minimizes the bandwidth consumption, improve the Quality of Experience (QoE), saves power and reduces costs. The disruptive design that removed GTP, which is offered by SDMN [6] is utilized by the authors to place the cache in any part of the network. Simulation is used to validate and analyse the performance of the proposed solution.

Finally, the authors of [49] proposed a resourceful technique to reduce the traffic handled by the mobile core network by utilizing OpenFlow-based solution. The authors think that their solution efficiently uses the network resources and balances the traffic over the core network, which may save the mobile operators millions of dollars per year. The authors focus on the HTTP traffic of the UEs Internet consumption considering that 47.7% of the traffic handled by the mobile core network is for HTTP applications [50]. Therefore, the authors used OpenFlow-based solution to redirect the HTTP traffic directly to the Internet, before reaching to the core network. The authors describe three different methods to allow OpenFlow switch to identify HTTP traffic sent over GTP-U tunnel and used OPNET simulator to demonstrate the redirection in the traffic passed through the mobile core network. The proposed solution does not cover how the offload downlink traffic is redirected to the correct UE location upon handover.

2.3.2 Distributed Mobility Management

The authors of [24] and [26] proposed a novel solution to implement DMM. The proposed solution consists of two parts: First, modification to the current LTE architecture; Second, SDN based architecture in the operator transport network outside the EPC network. The authors proposed several modifications to the current LTE architecture, that include: i) merge the functionality of both the SGW and PGW in a single entity entitled as S/PGW, which means eliminate the needs for the S5/S8 interface; ii) modify the hierarchy of the control and data planes of the current LTE architecture by using multiple S/PGW entities distributed near to the mobile access network; iii) several messages such as Create-Session-Request/Response, and Modify-Bearer-Request/Response are modified to allow the S/PGW to perform traffic redirection without the need for new IP address allocation.

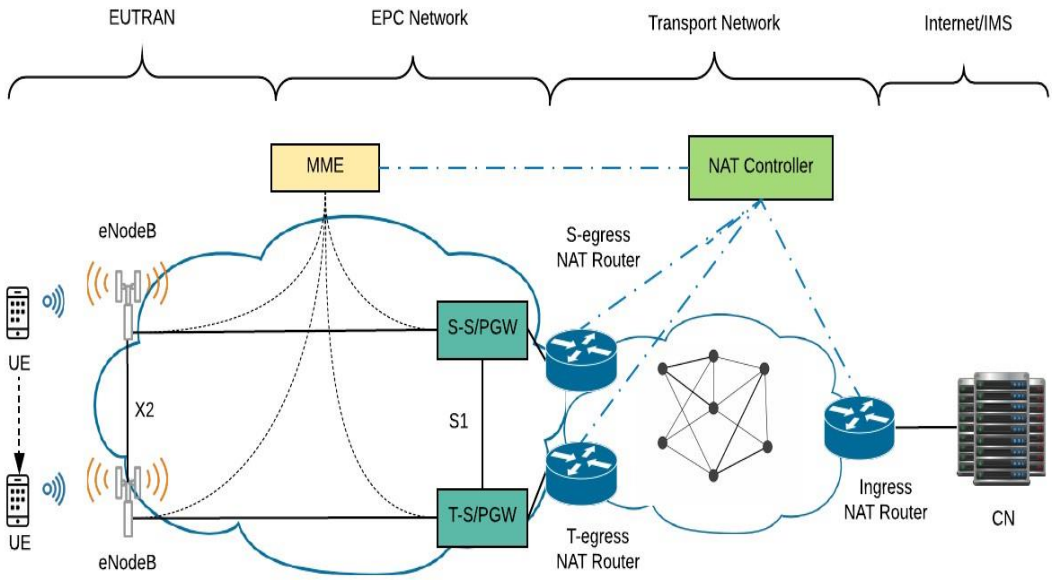


Figure 2.7: Double-NAT DMM Architecture proposed by [24]

As shown in Figure 2.7, the authors proposed that the distrusted S/PGWs are connected to the operator transport network outside the EPC. SDN solution is added to the transport network to support the traffic redirection between the S/PGWs and provides the UEs with service continuity during handover in an efficient manner. The SDN solution consists of NAT-controller, egress and ingress routers. The NAT-controller is connected with EPC centralized MME and configures the NAT routers when necessary to perform traffic redirection. Considering that NAT is already widely supported by most of the networking devices nowadays, the authors think that the adaptation of these

functionalities is easily feasible with a trivial overhead and complexity. Considering that NAT routers need to be added only at the edges of the operator transport network, the author thinks that the deployment of the double NAT solution is possible even for a very large network. The authors proposed several control messages to handle the communication between i) MME and the NAT-controller; ii) NAT controller and the NAT routers. The proposed solution is modelled in NS3 [51] and the authors provide a detailed description of the required the modification to the NS3 LTE model known as (LENA) to implement the Double NAT solution. The evaluation results presented in the authors work, showed that the proposed solution is fast enough in setting up the traffic redirection path and readily meets the latency requirement, considering the maximum allowed delay threshold for real-time applications (e.g., VoIP).

Meanwhile the authors of [26] argue that OpenFlow is a valid solution for cloud-based DMM. In this work the authors described and compared several technologies that can be used for the DMM, this includes: i) IETF Based DMM Enabling Technologies such as Double NAT [24], Distributed Mobility Anchoring (DMA), and Inter-domain DMM; ii) 3GPP DMM Enabling Technologies such as Local IP Access (LIPA) / Selected IP Traffic Offload (SIPTO). The authors come up with the conclusion that OpenFlow/SDN technology is the best candidate for the DMM solution. The authors think OpenFlow can efficiently be used for the support of DMM in virtualized LTE systems in order to provide the required flexibility and scalability to support seamless handover between two different PGWs.

Partial OpenFlow-based DMM proposed by [27] is another approach to overcome the centralized data forwarding, sub-optimal routing, potential single point of failure and low scalability of LTE network. The proposed architecture is like the works presented in [24] and [25] but applied in virtualized LTE systems. To support the continuity of a session of UE handing over from one PGW to another, new entities are added to the operator network; this includes: i) OpenFlow controller that has traffic redirection control application and co-located with the MME to reduce the signalling messages and use direct call instead; ii) Ingress and Egress FDs, the Egress FD can be either co-located PGW or in a standalone entity near the PGW, while the Ingress FD is deployed at the boundary of the mobile network operator. The authors used the NS3 simulator to build the proposed architecture, then measure and evaluate the handover delay from one PGW to another. The obtained result showed that 150ms is the best handover delay

obtained when the controller is positioned in the core network and the distance between Ingress and Egress FDs was lower than or equal to 4 hops.

Another DMM architecture is presented in [28] and [29]. The architecture utilizes a set of distributed PGWs and MMEs to increase network scalability and boost the system performance to cope with the explosive growth in the mobile Internet traffic, where Packet Data Network Edge Gateway (PEGW) is a new term used to describe the distributed PGW that is deployed near to the RAN. Several interfaces are proposed by the authors to handle the data and control plane operation between the new network entities. This includes the following interfaces: S1a-U, S2d, and S11a that are used to handle the data delivery between eNodeBs and the PEGW, data delivery between PEGWs, and bearer control between the MME and PEGW respectively. The authors validate the performance of the proposed architecture by comparing the PGW's data processing volume per unit time and the number of valid data sessions with the EPC network.

2.4 Software Defined Virtualized Mobile Network Architectures

This section describes virtualized SDN-based cellular network architectures. These architectures utilize SDN and NFV to provide data and control plane separation and on-demand dynamic provisioning of network resource. Software-defined control of virtualized mobile packet core network architecture proposed by [17] presents a solution that considers both SDN and NFV to offer programmability, boost network flexibility, provide on-demand dynamic provisioning, promising interoperability and reduction of both CAPEX and OPEX. The proposed architecture is based on the assumption that Open Networking Foundation (ONF) Wireless and Mobile Working Groups are working on hybrid OpenFlow switches that can perform complex functionalities that are required in the mobile network. The proposed architecture is based on a centralized EPC where the control plane entities are deployed on Virtualized Network Functions (VNFs) running in a data centre. Two SDN controllers are used namely, NFV domain SDN controller and Edge to Edge (E2E) SDN controller. The former is used to provide connectivity to the data centre compute and storage resources while the latter is responsible for the data plane network. The authors proposed to use GTP tunnel only over the S1 interface, and after that the UE IP address and VLAN tags

are used to route the traffic and differentiate between different bearers. VLAN tags can also be used for overlapping UE addresses going to different Access Point Names (APNs).

MobileFlow [18] is an SDN-based approach that defined a blueprint for reshaping the current and future mobile network architectures. The author's emphasis is on how difficult it is to extend, innovate and incorporate new features in the current mobile network once it has been deployed. MobileFlow is a solution that provides maximum flexibility, openness, and programmability to the mobile network without mandating any changes in UE. MobileFlow offers open interfaces and APIs, which fosters a rich environment for innovation to significantly increase the operator potential to roll out new network features, while reducing time to market for new services.

MobileFlow architecture mainly consists of: i) MobileFlow controller (MFC) with mobile network applications; ii) MobileFlow Forwarding Engine (MFFE). MFC consists of several functional blocks that include: mobile network function and mobile network abstraction. Northbound, Southbound and Horizontal interface are used to communicate with Mobile network applications, MFFEs, and Other MobileFlow controllers respectively. MFFEs are advanced FDs that include the required tools and functionality to support GTP operation. In another words, it is more complex than an OpenFlow switch but much simpler than a router or a PGW. Consequently, MobileFlow network can be integrated with legacy EPC equipment to provide backward compatibility. On-Demand Mobile Network (ODMN) prototype is used by the authors to demonstrate the programmability and flexibility of the MobileFlow architecture and provide implementation and experimentation details. No actual results are presented by the authors in this work.

Another Software-Defined 5G architecture that aims to provide the flexibility and scalability required to handle the new emerging communication paradigms such as mobile cloud computing, mobile social networking, and Internet-of-Things is proposed by [3], SoftNet is the name used by the authors to refer to their proposed architecture. SoftNet is a new architecture that re-designs the current mobile network architecture instead of enhancing it. SoftNet aims to provide Adaptability, Efficiency, Scalability, and Simplicity by utilizing Virtualized SDN-based core network and access server to provide Unified RAN. It utilizes decentralized mobility management, distributed data forwarding, and multi-RATs coordination to i) reduce signalling overhead; ii) improve the efficiency of data forwarding and iii) enhance system capacity respectively. SoftNet

has a dynamically defined design which adaptively changes it working structure to accommodate different communication scenarios. Network configuration, operator's policies, and network stats are used to specify the working network architecture. The authors used simulation to evaluate the performance of the proposed architecture to demonstrate the reduction in signalling cost offered by decentralized mobility management compared to the standard LTE mobile network. The signalling cost is measured when eNodeBs are connected to the same access server and more optimization is required with inter-access server handover.

While Cellular SDN (CSDN) architecture proposed by [19] is another 5G network that offers dynamic resource orchestration by leveraging SDN and NFV. The former enables data plane and control plane separation while the latter effectively decouples the logic of network functions from the underlying hardware. The proposed architecture consists of 4 planes that include: knowledge plane, network applications plane, control plane, and forwarding plane as shown in Figure 2.8. These planes exchange information through open interfaces.

In CSDN the knowledge plane gathers and maintains comprehensive information about the UEs, network and usage data which can be used to optimize network utilization and to enhance the user experience. The network applications plane is implemented in a centralized cloud-based infrastructure and holds the majority of the LTE control functions which includes virtualized SGW, virtualized PGW, virtualized MME, Routing, and RRM. The centralization of the RRM and the separation of the control plane from the radio equipment make it easier to perform radio resource allocation as well as backhauling. The proposed architecture provides the flexibility to dynamically implement and instantiate new applications which reduce the cost and time-to-market for new adapted and personalized services. The control plane consists of Network Operating System (NOS), a network virtualization block and uses OpenFlow protocol to communicate with the forwarding plane. The latter consists of a set of OpenFlow switches. The authors think that OpenFlow switch with the current specifications is insufficient due to the increasing number of UEs, roaming nature of the mobile devices and the required fine grain access control and proposed to extend it but without specifying the extensions.

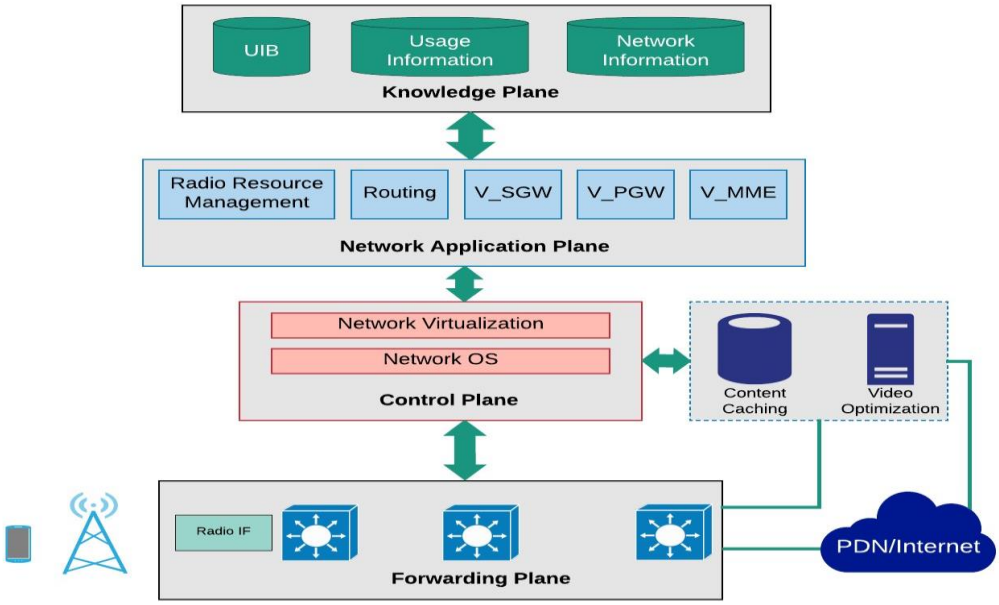


Figure 2.8: Cellular software defined network proposed by [19]

2.5 Cloud RAN and SDMN Security

This section describes a few works that handle security threats for the SDN-Based cellular network. Also work that describes Cloud RAN solution is presented in this section. The works presented in [52][53] focus on the security aspect of the SDN, NFV and mobile network integration, where the authors discuss the security challenges of the SDMN presented in [6]. The authors presented a comprehensive multi-tier component-based security architecture to address the vulnerabilities exposed due to the adaptation of new telecommunication paradigm in the SDMN. The proposed security architecture includes important components such as i) secure the control and data planes communications; ii) Denial of Service (DoS) attacks mitigations by utilizing policy-based communications; iii) Implements Deep Packet Inspection (DPI) and SDN-Based traffic monitoring techniques to improve security threat detection and orchestrate the monitoring activities receptively. The authors used testbeds to validate the feasibility to implement the proposed security architecture in the real world. The authors used experiments to demonstrate the ability of the proposed architecture to prevent IP based attacks and provides an automated countermeasures reaction to mitigate network threats.

Meanwhile, many other types of research focus on reshaping the access network of the mobile network by integrating of SDN and NFV with the RAN, such as the works presented in [54][55][56][57]. The authors of [54][55][56] discusses how

heterogeneous RANs increased the complexity of the mobile network and the emphasis on the data plane control plane separation offered by SDN as a potential solution to address the control plane problems in these networks.

In [54], the authors describe an architecture with Cloud-based RAN, in which the control plane functionality is virtualized in NFV platform that allows the orchestration of resources using virtualization techniques. On the other hand, the authors of [57] present a novel approach that extends the current Third Generation Partnership Project (3GPP)'s 5G architecture. The authors propose to reduce the signalling costs between the RAN and the core network through the centralization of the control plane functionality of the 5G RAN. The proposed architecture provides end-to-end separation between the data and control planes. In their proposal, the base station known as the gNB is converted into a pure data plane node along with the other FDs. The authors think that the running the Radio Resource Control (RRC) functionality and RRM into the core network to centralize the control of radio resources will reduce the signalling cost, provides better radio resources management due to the network-wide view, and reduce the processing time of header encoding and decoding. NS3-simulator is used to validate the proposed architecture. The authors used the mobility management, attachment time and system throughput to demonstrate the improvement offered by the centralized algorithms used by the proposed architecture.

2.6 Summary

This chapter presents a comprehensive survey covering a wide range of researches that demonstrate different visions regarding the integration of SDN, NFV, and Cloud computing with the mobile core network as a step toward the 5G mobile network. Some of the works focus on the RAN network, and others focus on the core network. Despite that the integration of the aforementioned approaches with the RAN had been receiving great attention, here the EPC network is the main focus and a few of the works that focused on the RAN are briefly mentioned. These works characterized as following: First, SDMNA is described, which sub-divided to SDMNA with GTP and SDMNA without GTP. The SDMNA without GTP itself is further divided to two sub-sections that include: i) disruptive design the remove GTP from the whole network; ii) Layer 2 based backhaul that only removes GTP from the S1 interface. Second, the works that used SDN as an add-on to perform selective traffic offloading or distributed mobile

management are described. The third category describes the Software Defined Virtualized Mobile Network Architectures (SDVMNA), which present some of the works that adopt both SDN and NFV in their proposed network architectures. Then, finished by presenting a few works that related to the integration of the SDN and NFV with the RAN. Also, the possible security threats that can be introduced by using SDN, NFV in the mobile network is described in this section as well. These works tried to address several issues with the current mobile network that prevent it from coping with the growing demands of their users. This includes: i) improve the system performance by reducing signalling loads; ii) address the sub-optimal routing caused by the hierarchical design of the current 4G network; iii) better resource utilization and load distributions; iv) enhance failure recovery procedures; v) reduce the CAPEX and OPEX; vi) increase the network flexibility and scalability.

CHAPTE 3

3 OpenFlow Technology and Implementation Model

3.1 Introduction

This chapter explains the similarities and the differences between the traditional network and SDN. Also, this chapter discusses the specification of OpenFlow protocol in a way that shows the development and the evolution process of the multiple versions of the protocol. Finally, it describes the simulation model of the OpenFlow protocol in OMNeT++. This model explains in detail the design and the structure of the OpenFlow controller, and switch nodes. Also, it describes the OpenFlow messages supported in the simulation model.

3.2 Networking Technologies

Networking devices consist of three planes. First, the data plane which represents the device hardware and processing silicon; second, the control plane which represent the logic and the intelligence of the device, which normally is performed by software applications that runs standard or proprietary protocols; Typically, the framework and the control plane applications are developed and maintained by vendors only [58]. The third plane is the management plane, which is part of the control plane and used by operators and IT administrators to monitor and configure the networking devices. These planes reside inside almost all the networking devices nowadays. A fairly new and emerging network architecture known as SDN has proposed the separation of the data and control planes. In SDN the control plane is independent and centralized, which increases the programmability of networking devices. The migration of the control

plane enables abstraction of network infrastructure and treats the network as a virtual or logical entity. The next sub-sections describe both the traditional and SDN based networking.

3.2.1 Traditional networking

Traditional networks are a combination of Local Area Network (LAN) and WAN networks connected to each other to form a bigger network and so on. These networks include various networking device such as switches, routers and firewalls. LAN represents a group of networking and end user devices interconnected to each other in a relatively small geographic area. This can be a home, small building or even a university. On the other hand, WAN interconnects devices across different cities as a nationwide network in a country and it connects many LANs together [59]. In these networks data traffic is transmitted over different connections utilizing multiple technologies. LAN Ethernet is the most commonly used connection because it is very flexible and mature and can offer a variety of connection speeds like 100Mbps, 1Gbps, even 10Gbps. WAN uses different connection technologies, but optical fibre is the most promising technology as it offers a very high-speed data rate.

3.2.1.1 Switch

A switch is a network device, works at Layer 2 of the Open Systems Interconnection reference model. In networking nowadays, switches are used to forward data traffic between two nodes in the network. It is possible that both nodes are connected to the same switch or connected to two different witches in the network. The switch utilizes layer 2 information such as: destination MAC address and the VLAN id to specify the output direction for the received frames. The forwarding process consists of two steps. In the first step the switch uses the source MAC address to learn about the location of the connected device, then in the second setup the destination MAC address is used to find the correct output port. Upon receiving a new frame, the destination MAC address is lookup against the switch forwarding table, and if a match is found, then the frame is sent out the port specified by the match entry. Otherwise, the frame is flooded to all ports except the port on which it was received [59]. Considering that a switch only works based on layer 2 information, any packet which is destined to another IP sub-net cannot be processed by it and is sent to a router for further processing.

3.2.1.2 Router

A router is a networking device that works and operates at the layer 3 of the Open Systems Interconnection reference model. It is normally used as LAN gateway. It connects several LANs and WANs together. The control plane of the routers runs routing protocols that perform a computational task to build the routing behaviour, which includes finding the optimal path between any two points in the network. The best path calculated by the routing protocol may depend on the number of hops, the link bandwidth and other parameters. Upon calculating the best path, the routing protocol configures the routing table with this information, which helps the router to forward the traffic from one IP subnet to another. Traffic forwarding is performed by the router based on the destination IP address of the received packet in contrast to the MAC address used by the switch. The process starts after receiving the packet, where the routing checks the destination IP address, and if it directly connected then the packet is sent though that interface, alternatively it forwards a packet to another router, if the destination node lies further away [58].

3.2.2 Software Defined Networking

SDN is a modern approach to networking that is considered in the eyes of some experts as a revolution in the networking industry as it aims to eliminate the complexity and the static nature of traditional distributed network architectures. SDN offers control and data planes separation to centralize the network intelligence and abstract it from underlying network infrastructure, as shown in Figure 3.1. SDN architecture converges the management of network and application services into a centralized, extensible orchestration platform that automates the provisioning and configuration of the entire infrastructure. [60] The control plane is centralized on a software-based controller that provides performance and fault management, which typically handles the configuration and the management of the SDN complaint devices and understands the network topology. Loaded with these details, the controller can process requests based on the desired requirements such as Quality of Service (QoS) level and also perform link management between devices. The data plane is responsible for data traffic forwarding. The centralization of the control plane helps accelerate application deployment and delivery, which improves network agility and automation dramatically, while substantially reducing the cost of network operations via policy-enabled work-flow

automation. SDN has gained a massive interest is both academia and the industry because many of the big players of networking around the world support it. This includes Google, Microsoft, Yahoo and Facebook because of the characteristics offered by SDN. SDN benefits can lead to:

- **Centralized control of multi-vendor environments**: in traditional networking, vendor specific interfaces and configuration is a real problem that is always a concern for the IT team of the network because it requires a specialized team to manage and configure a small part of the network from an individual vender. Differently, in SDN based networks, the service and management applications can configure any OpenFlow-enabled network device (switches, routers, virtual switches) from any vendor. The centralized control plane helps the IT administrators to deploy and update network services of the entire network in a quick and efficient manner [60].
- **Reduced complexity through automation**: SDN controllers are normally equipped with tools that increase the flexibility of network management and offers automation framework. These tools help solve the problem of manual management and human error that lead to network instability. At the same time these tools reduce operator overhead and support emerging IT-as-a-Service and self-service provisioning models. The integration of SDN in a cloud-based solution that provides self-provisioning and intelligent orchestration helps reduce the operation overhead and increase the business agility.
- **Higher rate of innovation**: the centralized control plane offered by SDN allows network administrator to dynamically (re)program the entire network in real time to meet user expectations and achieve faster business innovation. Network administrators can define network behaviour or introduce a new service in a matter of hours by leveraging network virtualization and services abstraction [61].
- **Increased network reliability and security**: in traditional networking that is used nowadays, network configuration and policy that are specified by IT administrators or network operators need to be configured on all the network devices one by one. These devices need to be re-configured each time a new services or application is introduced or removed from the network. The story is different with the SDN-based networks because southbound protocols like OpenFlow help to deliver the network configuration and policy to the network

infrastructure and when a policy is changed, or service or application needs to be modified, the controller re-configures the network devices using OpenFlow, which eliminates the need to individually (re)configure network devices used in networking these days [61].

- **More granular network control**: southbound protocols like OpenFlow offer a very granular level of policy control. This includes session, user, device, and application levels, and with the network abstraction and automation offered by SDN, cloud operators can maintain and support multiple tenants while keeping them isolated in terms of security and resource management, if the customers share the same infrastructure.
- **Better user experience**: The Centralized control plane offers a unified interface to manage the network of different equipment manufacturers. In addition, the centralization of the network state is leveraged by the high-level applications to provide dynamic and more efficient adaptation to the business needs.

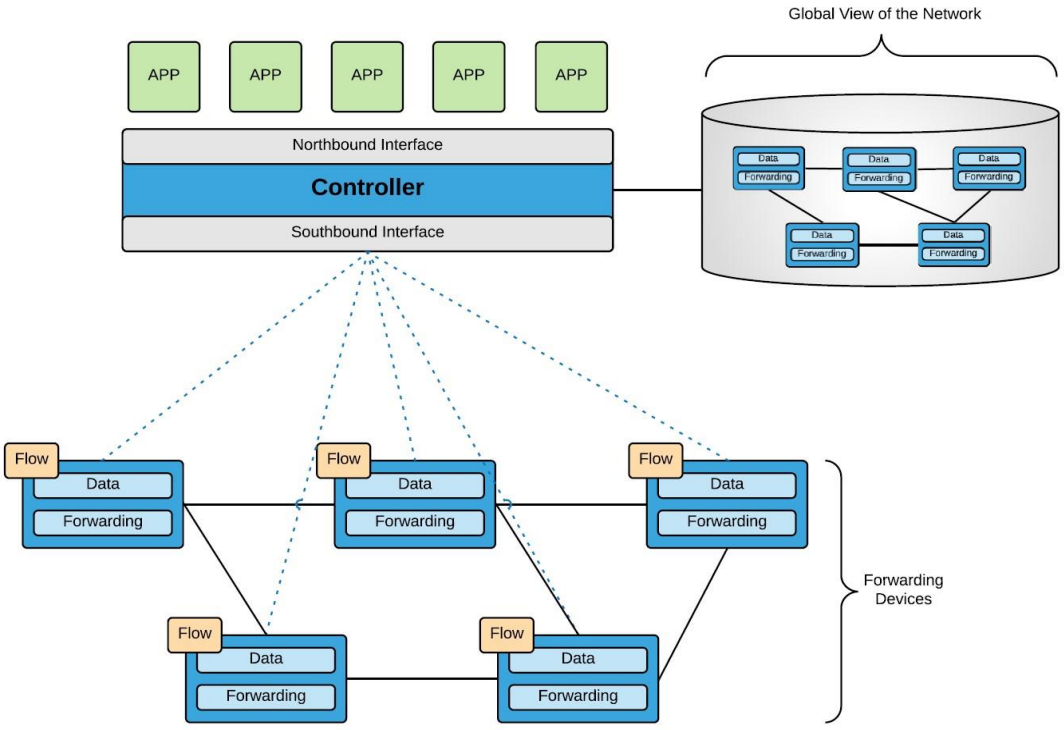


Figure 3.1: Software Defined Networking Architecture

3.3 OpenFlow

OpenFlow protocol represents the first and the most commonly used standard protocol for the communications between the control and data plane layers of the SDN

architecture. It is standardized by the ONF. ONF is a non-profit foundation that deals with the advancement and the standardization of OpenFlow. The standardization of OpenFlow protocol makes the control plane - data plane separation possible and facilitates the manipulation of the data plane devices by high-level applications. OpenFlow-based SDN solution is currently being implemented in a variety of networks where the control plane is abstracted from the underlying data plane device. The FDs or data plane of the network such as a switch or a router can easily be managed through the OpenFlow interface. The advantages offered by OpenFlow have made networking vendors to include OpenFlow support in most of their switches, routers and wireless access points. By including OpenFlow in network devices, researchers can run their experiments without the need to reconfigure networking devices to alter their working behaviour. Multiple versions of OpenFlow protocols have been specified by ONF. This section briefly describes OpenFlow version 1.0, 1.1, 1.2 and 1.3. Although OpenFlow 1.5 is the latest version but the focus only on the aforementioned versions because the features offered by OpenFlow 1.5 are not relevant to the requirement of this work.

OpenFlow 1.0 was released in December 2009 [62]. It is the most commonly deployed version of OpenFlow. The OpenFlow 1.0 specification exactly described the data plane FDs. It consists of a flow-table and the means to talk to the control plane that resides on a remote controller using the OpenFlow Protocol. OpenFlow 1.1 added the support for a pipeline of multiple cascaded flow tables, group-table and MPLS label-related actions [63].

OpenFlow 1.2 extends the number of supported protocols by adding support for IPv6. Switches running OpenFlow 1.2 and above can match packets based on their IPv6 protocol number, source and destination addresses, flow label, traffic class and various ICMPv6 fields. In OpenFlow 1.2 [64] flow match fields are described in the OpenFlow eXtensible Match (OXM) format, which is a compact Type-Length-Value (TLV) structure. This way it is possible for anyone to define new match entries in an extensible way. Also, OpenFlow 1.2 is more resilient, scalable and efficient compared to the previous versions since a switch can be connected to one or more controllers simultaneously [64].

OpenFlow 1.3 was released in June 2012 [65]. It supports all OpenFlow 1.1, 1.2 features and provides new features for monitoring and operations management with more robust QoS support. In contrast to OpenFlow 1.0 based systems, OpenFlow 1.3 can provide different benefits in terms of the supported protocols, possibility to load

balancing, fault tolerance, and richer support for QoS such as basic rate limiting or even more complex QoS using diffserv [63].

3.3.1 OpenFlow Switch

OpenFlow Switch is a new concept for networking devices that most commonly known as datapath processing considering that it only implements the data forwarding. There are two types of OpenFlow switch classifications based on the supported operations: OpenFlow-only, and OpenFlow-hybrid. The first type processes the received packets only by the OpenFlow pipeline, while the second implements both the OpenFlow pipeline and the normal L2 switch functionality. Each OpenFlow switch has a secure channel with the centralized OpenFlow controller. The controller manages the switches using OpenFlow protocol leveraging this channel. The controller manages the switch via this protocol, by add, update, and delete flow entries, both reactively (in response to packets) and proactively. OpenFlow protocol specifications outline the switch structure and operations. This includes the number and type of tables, the mandatory match fields, and the possible actions a switch can take if a match occurs. This section explains the structure of OpenFlow switch as specified by OpenFlow protocol versions 1.0, 1.2, and 1.3.

3.3.1.1 Switch as Specified by OpenFlow 1.0 Specifications

Figure 3.2 shows the basic structure of OpenFlow switch as specified by OpenFlow protocol 1.0 specification. The switch basically consists of a secure channel and one flow-table. The table contains multiple flow entries; each flow-entry consists of i) matching rules that are composed of a set of ingress port and L2/L3/L4 header fields, which may be variously wildcarded or masked, ii) a list of one or more actions attached to each match rule, iii) counters for collecting statistics about the flows [66].

In OpenFlow 1.0, actions associated with flow-entry include forwarding actions, where the switch forwards the packet to the port specified in the output action or floods it to all ports. Furthermore, the controller can instruct the switch to forward all the traffic that match a certain flow-entry to the controller. More actions that empower OpenFlow switch to be used for network access control or even as a firewall are available such as the drop packets action. It is also possible to modify the header of the incoming packets in OpenFlow 1.0 including the modification of VLAN tag, IP source, destination

addresses, etc. By default, if no flow-entry matches the incoming packet headers, then the switch notifies its controller about the packet using an OFP-Packet-In message [67]. OFP-Packet-In message may either contain the entire packet or just a fraction of the packet header. In the second case, the packet is stored in a buffer at the switch and the OFP-Packet-In message contains the buffer ID of the stored packet. The controller that receives the OFP-Packet-In notification message identifies the correct action for the packet and sends an OFP-Flow-Mod Message asking the switch to handle the upcoming packets according to the new rules and an OFP-Packet-Out message to forward the buffered packet. Whenever a packet is processed by an OpenFlow switch, the statistics are updated using various counters in the switch. OpenFlow controller may query the switch for this information that can be used in its applications. OpenFlow 1.0 switch keeps statistics about flows that are processed by flow-entry, port, and queue [68].

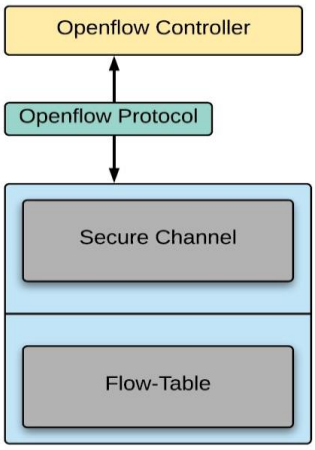


Figure 3.2: OpenFlow 1.0 Switch Structure

3.3.1.2 Switch as Specified by OpenFlow 1.2 Specifications

Figure 3.3 illustrates the main components of OpenFlow 1.2 switch. The switch consists of i) processing pipeline; ii) switch ports; iii) secure channel. The processing pipeline is updated to include a new table named group-table and instead of one flow-table the switch includes set of multiple flow-tables linked together and sequentially numbered, starting at 0. Packets are processed through a pipeline of multiple flow-tables starting from the first table. The packet is compared against the flow-table entries. If the packet matches a single flow-entry then the respective instruction-set are applied, and the flow-entry counters are updated, while if more than one matching entry are found the most specific entry with the highest priority is selected [64].

The instructions of the selected flow-entry may include updating the action set, updating the metadata value, and performing actions. It also may send the packet to another flow-table using the Goto-Instruction where the same procedure is repeated [68]. In the processing pipeline, metadata fields are collected from the packet during the matching process and used to pass information between the tables as the packet traverses through them. Flow-table entries contain instructions instead of actions. flow-entry is only allowed to send the packet in a forward direction to a flow-table with greater number than its own flow-table number. If the instruction-set of a flow-entry does not contain a Goto-instruction, the pipeline processing stops, and the accumulated action set is executed on the packet. Flow-table entries can also direct the packet to a group-table. Group-table is a special and more sophisticated table designed to facilitate more complex and specialized packet operations that are common across multiple flow entries and it supports more complex forwarding behaviours such as load balancing, fault tolerant and link aggregation. It consists of one or more group entries. Each entry contains group-id, group type, action buckets, and counters.

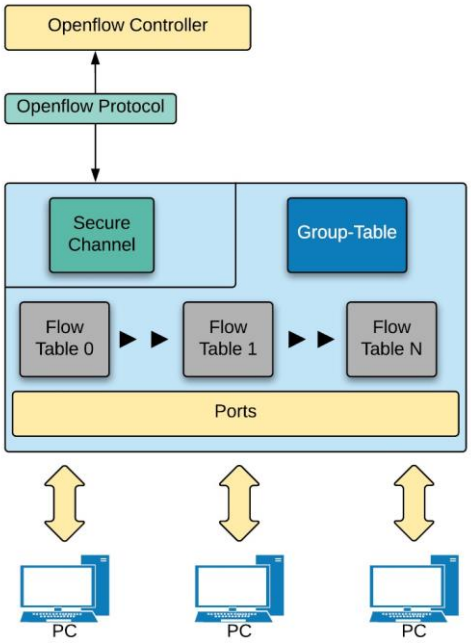


Figure 3.3: OpenFlow 1.2 Switch Structure

3.3.1.3 Switch as Specified by OpenFlow 1.3 Specification

The main components of an OpenFlow switch as specified by OpenFlow 1.3 specifications are shown in Figure 3.4. It consists of: i) one or more flow-table; ii) group and meter-tables; iii) set of OpenFlow ports; iv) secure channel with the centralized OpenFlow controller.

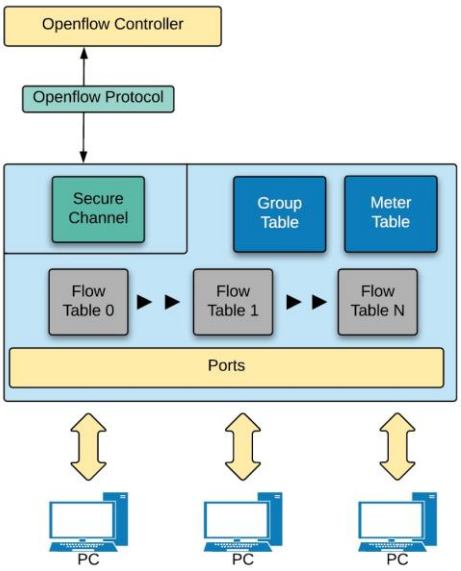


Figure 3.4: Main Component of OpenFlow Switch

A. Flow Table

The flow-table has a similar structure to the traditional networking devices forwarding table. Each table contains a set of flow entries that are used in the processing pipeline to perform packet lookup, header manipulation, and forwarding decisions. Each flow-entry consists of three main components; first, matching fields, this includes incoming port number, source or destination MAC address, source or destination IPv4 address, source and destination port number etc. second, set of instructions to be executed on the matched flow; finally, counters to record a detailed statistic about the flow as shown in Figure 3.5.

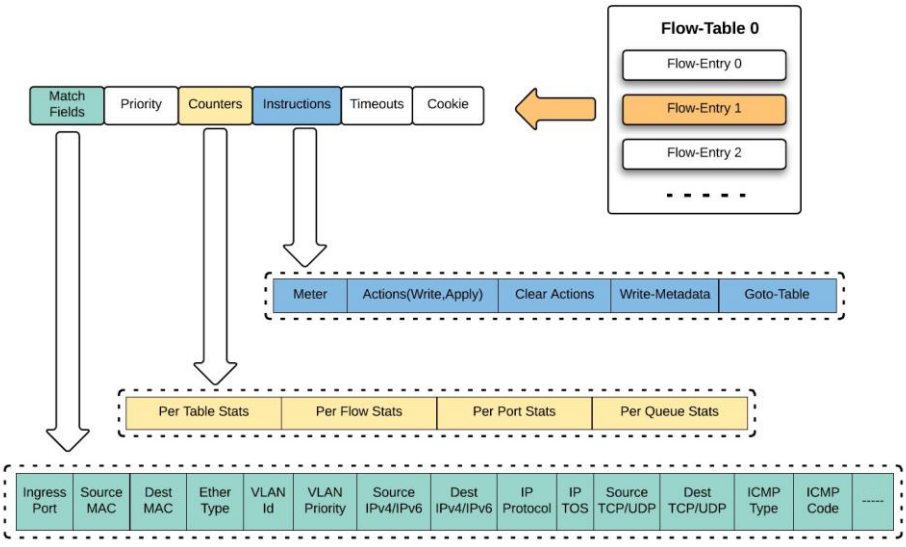


Figure 3.5: Flow Table Basic Structure

B. Group Table

Group-table is a special table used to perform complex operations. It consists of multiple group entries as shown in Figure 3.6. Each entry consists of: i) group identifier which represents a unique identifier used a key id in the group-table. Normally frames are sent to the group-table by the flow-table entry utilizing the group identifier; ii) group type, four types are support by OpenFlow switches for which more detail is provided in the next sub-section; iii) counters store statistics about the packets handled by the group-entry and are updated each time packets are processed by the entry; iv) an ordered list of action buckets that includes actions to be executed on the packet.

I. Group Types

Four types of groups have been defined by the OpenFlow specification, but the switch does not have to support all types. In other words, two types are mandatory while the other two are optional. This section describes these types in detail.

- **All**: it is the first required group type that is mandatory to be supported by all switches running OpenFlow 1.2 or later. All the group-entry action buckets are executed on the packet processed by group-entry type ALL. This type is often used to implement broadcast and multicast operations. To do that the received packet is cloned several times (equal to the number of action buckets), and each copy is processed by a single bucket. To send the packet back through the same port it was received on, the group must include an extra bucket, which includes an output action to the OFPP_IN_PORT reserved port.
- **Select**: It is an optional group type that is primarily designed for load balancing. Unlike the ALL group type, packet is processed by only one action bucket based on a selection algorithm that is implementation specific (e.g. round robin). Although OpenFlow specification does not specify a selection algorithm but it does point to a few features that should be supported by the implemented selection algorithm. This includes equal load sharing selection that can utilize bucket weights. In this type, if the outport specified by the match action bucket is down, the switch may restrict bucket selection to the remaining set (those with forwarding actions to live ports) instead of dropping packets destined to that port [64].
- **Indirect**: this is the second required type, although is it a group type but it only has one action bucket and all the received packets are handled by this bucket.

The purpose of this type is to accommodate a common set of actions to be utilized by multiple flow entries to simplify the switch pipeline operations and support faster and more efficient convergence.

- **Fast failover**: This is an optional group type designed specifically to detect and overcome port failures. A list of buckets is associated with this group, and each action bucket has port and/or group parameters, which are used to monitor the liveness of the bucket. Normally upon receiving a packet, the bucket list is evaluated, and the first alive bucket is picked to handle the packet, but if no bucket is alive then the packet is dropped. With this group type, a path switch can be performed without the need to contact the controller.

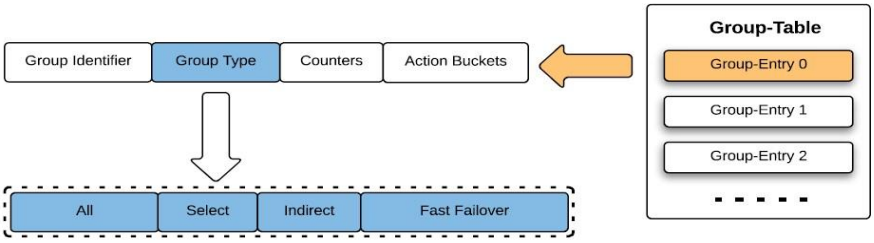


Figure 3.6: Group Table Basic Structure

C. Meter Table

The basic structure of meter-table is illustrated in Figure 3.7. Meter-table has one or more entries, and each entry consists of: i) meter identifier; ii) meter bands; iii) counters. Meter-table is used to provide per-flow metering, which helps the implementation of simple QoS operations like rate-limiting. At the same time, more complex QoS frameworks such as Diffserv is supported by OpenFlow 1.3 specification by combining per-flow metering with per-port queuing. To provide per-flow metering, meter entries are attached directly to the flow-table entries by the means of flow-entry instructions.

Meter-entry measures the packets directed to it and helps control the rate of those packets. As previously mentioned, meter-entry includes one or more meter-bands and each band has a type and rate associated with it. Packets are handled by only one meter-band, more specifically the band with the highest configured rate that is lower than the current measured rate. If the current measured rate is lower than the configured rate of all meter bands, then no meter band is applied [67]. Two meter-band types are specified in OpenFlow 1.3 specification, namely drop and DSCP-remark as shown in Figure 3.7.

The first type is used to define a rate-limit, while the second can be used as a simple Diffserv policer by increasing the drop precedence of the DSCP field.

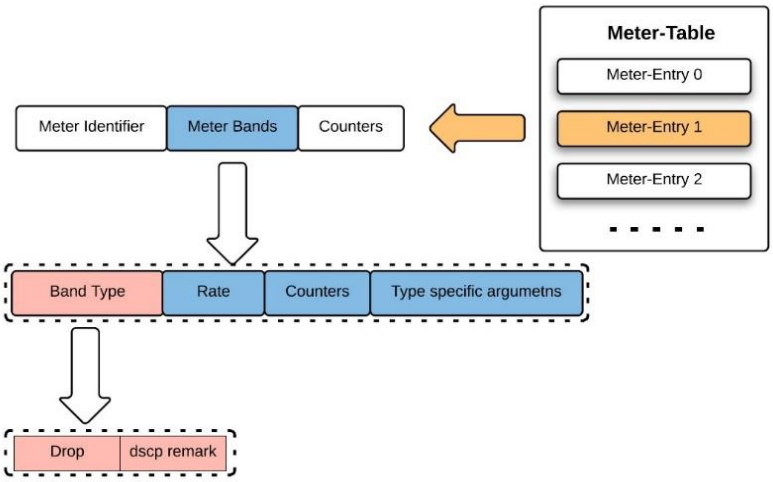


Figure 3.7: Meter Table Basic Structure

D. Processing pipeline procedure

The processing pipeline operation starts immediately after receiving a packet from the switch port. As previously mentioned, flow-tables are sequentially numbered, the received packet is always matched against the flow entries of the first flow-table. If no match is found, then the table miss-entry is executed. Figure 3.8 illustrate the switch processing pipeline procedure.

Upon receiving the frame, an empty actionSet and metadata are attached to the frame and the header information of the received frame is matched against the flow entries of flow-table 0. If no match is found, the table miss-entry is executed. The table miss-entry is normally used to either drop the packet or send it in an OFP-Packet-In message to the controller. Alternatively, if a match is found, then the counters of the match entry are updated, and the instructions associated with the match entry are executed on the received frame. If a Goto-instruction is included in the matched flow-entry instructions, then the packet is sent to the table specified by the instruction. The packet is only allowed to be sent to a table with a higher sequence number. In other words, pipeline processing can only go forward and not backward. The pipeline processing ends when the matched entry instructions do not have Goto-instruction. At this stage the accumulated actionSet is executed on the packet. The actions may send the packet to a group-table for feature processing and at the end the packet is either sent to the network through an output port or dropped if no output action is included in the actionSet.

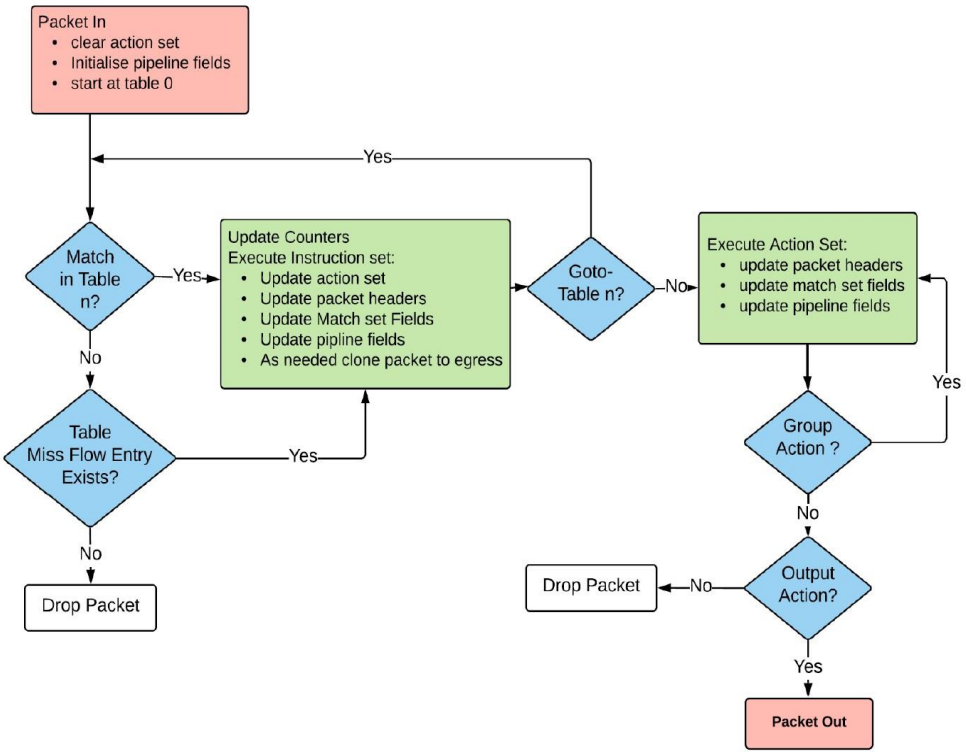


Figure 3.8: OpenFlow 1.3 switch processing pipeline procedure

E. OpenFlow Ports

OpenFlow ports are interfaces that are simply used to exchange packets between the switch processing pipeline and the network. OpenFlow ports are available for the switch processing pipeline, which may be equivalent to the network interfaces provided by the switch hardware but mostly this is not the case considering that some of the network interfaces can be disabled for OpenFlow and other additional logical interfaces that can be provided by the switch to the processing pipeline.

OpenFlow switches are normally connected to each other using OpenFlow ports. A packet travels from the first switch egress port and is received by the second switch ingress port. OpenFlow ingress port is considered as a property of the packet throughout the processing pipeline and represents the switch port on which the packet was received. In the processing pipeline the packet ingress port can be used as matching criteria to identify the type of the flow and define how the packet should be handled by executing one or more actions on the packet. OpenFlow ports can be classified as physical, logical, and reserved ports. As previously mentioned, these ports can be used as ingress or egress ports. OpenFlow ports have counters and they have state and configuration [67].

I. Physical Ports

OpenFlow physical ports are switch defined ports that represent one-to-one mapping with the switch hardware interfaces. For instance, the switch Ethernet interfaces are mapped to OpenFlow physical interfaces. In a virtualized environment, OpenFlow physical interfaces are mapped to the virtual interface of the virtual switch.

II. Logical Ports

OpenFlow logical ports represent higher level abstractions defined by the switch and used to perform complex operations e.g. link aggregation groups, tunnels, loopback interfaces. OpenFlow logical port does not correspond directly to the hardware interface of the switch as it may map to various physical ports. The logical port operations are not specified by OpenFlow protocol and it is implementation dependent. Logical port should interact with the switch processing pipeline in same way as physical ports, with an extra property attached to the flow received by logical port to precisely identify the logical port in the switch processing pipeline.

III. Reserved Ports

Another type of switch defined OpenFlow port are the reserved ports. These ports can be used to forward the received traffic using OpenFlow or non-OpenFlow methods. In the first case, these ports can send the traffic to the controller or flooding out all the interfaces except the one on which it was received, while in the second case, these ports can be used to forward the traffic using the normal switch processing. Specified by OpenFlow protocol, it is required that the switch implements the following reserved ports.

- **All**: reserved port that is used as egress port only. It is used by the switch to send a packet to all the ports except the port from where the packet is received and ports that are configured to not forward.
- **Controller**: switch defined reserved port that can be used as ingress or egress port. Normally used to communicate with the control plane. It works as egress port and encapsulates the packet within a packet-in message and sends it to the controller, and it identifies the controller message when it works as an ingress port.
- **In Port**: another reserved port supported by the switch. This port is used only as an output port and is normally used to send the received packet back to the same interface it was received on.

- **Any**: It is not an ingress port or egress port. It is a special value used by some OpenFlow commands as a port wildcard when no port is specified.
- **Table**: unique ports that are only used in the action list of OFP-Packet-Out message, where it is used to send the received packet to the first flow-table to be processed by the switch processing pipeline. Also a few reserved ports that are not mandatory to be implemented by OpenFlow switch but still supported by many switches.
- **Local**: reserved port that can be used as either ingress or egress port.
- **Flood**: a reserved port that can be used only as an egress port. In general, it uses the traditional non-OpenFlow pipeline to send the packet out to all the switch interfaces excluding the port from where the packet was received. The switch may also use the packet VLAN ID or other criteria to select which ports to use for flooding.
- **Normal**: optional reserved port that can only be used as an egress port. The same as the FLOOD port, this port uses the traditional non-OpenFlow pipeline to forward the received packet. In general, the process of routing or switching the packet is implementation independent and it is not specified by the OpenFlow specification.

3.3.2 Connection Establishment between Switch and Controller

All OpenFlow capable switches need to register itself with a controller. The TCP handshake process is started by the switch through sending a TCP sync message to the controller IP address at the default TCP port 6633. The controller replies to the switch by sending a sync acknowledgement message. When the sync acknowledgement message arrives at the switch, an acknowledgment message is sent back to the controller and the TCP session is established between the two devices. After that echo request and reply are exchanged between the two devices to maintain a healthy connection. The same procedure is used when a new OpenFlow switch is added to the network.

3.3.3 OpenFlow Messages

Three types of messages are specified by the OpenFlow specifications. This includes controller-to-switch, asynchronous, and symmetric, each with multiple sub-types [67].

The name is the recipe with controller-to-switch messages considering that it describes the messages generated and sent by the controller to the switches, these messages are used to manage, configure, and inspect the state of the data plane switches. The asynchronous messages represent the class of messages generated and sent by the data plane switches to notify the controller about a new event or state change in the network. The last type describes the messages generated and sent by either the controller or the switch without any solicitation. Section 3.4.2.3 describes in more detail the most common messages of each type and at the same time which of them are modelled in the OpenFlow 1.3 model.

3.4 Simulation Model

In order to fully understand the design and the implementation of the OpenFlow 1.3 model it is required to understand OMNeT++ simulator. In this section a brief description about OMNeT++ simulator is provided and then the design and functionality of OpenFlow 1.3 model is presented with more focus on the controller and switch entities.

3.4.1 OMNeT++ Simulation Framework

OMNeT++ is an object-oriented modular discrete event network simulation framework that can be used to model wired and wireless networks, network protocols, and many other things. Here only the OMNeT++ aspects that are more relevant to understanding OpenFlow 1.3 model is described. More information about OMNeT++ framework can be found in [71]. OMNeT++ in itself is not a network simulator, rather it is a framework that provides infrastructure of components and tools called modules, which can be combined together to form the simulation network. These OMNeT++ modules have gates, which act as interfaces. The gates are connected by predefined connection links. The modules communicate with each other by sending and receiving messages through those modules' gates. At the top of this Ease of Use hierarchy is the network, which has no gates to the outside world. In OMNeT++, the module's structure (gates, connections, etc.) is described in files written in NED (Network description) language. The implementation of the module's behaviour is written in C++, while the parameters' value that customize the module behaviour and define the simulation topology can be assigned in either the NED or .ini files [69].

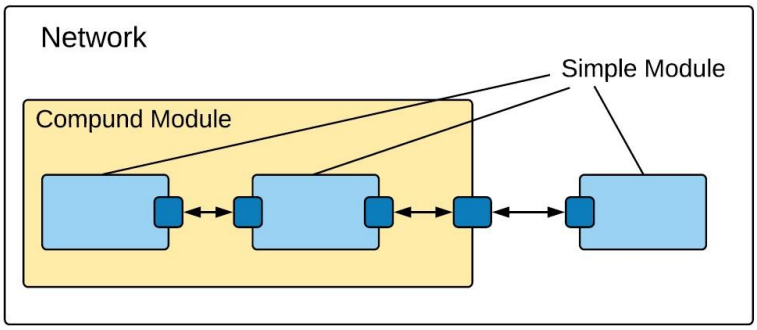


Figure 3.9: OMNeT++ Basic Building Blocks

The active modules are written in C++, by using the class library of the OMNeT++, and they are called simple modules (which sit at the lowest level of the module hierarchy). Simple modules can be grouped to form compound modules. Simple modules in different compound modules can connect with each other only through the compound module's gates (the connections cannot sidestep module hierarchy) as shown in Figure 3.9. Overall, the OMNeT++ simulation model consists of simple modules communicating via messages. The creation, deletion, modification, storage, transmission, and reception of messages is the main job of the simple modules, hence the whole OMNeT++ model is there to accomplish this job. To create or destroy a message object you need to use the C++ New or Delete operators. The message object is an instance of a class called cMessage, or one of its subclasses. Practically, fields should be added to the cMessage to customize it upon your simulation requirements, by creating subclasses to extend the cMessage class. The same goes for the network packets, since it is an instance of the cPacket, which is sub-classed from the cMessage.

3.4.2 OpenFlow 1.3 Simulation Model

OpenFlow 1.3 model is developed as extension to the INET library of OMNeT++ simulation framework. The model was carried out as an upgrade to the OpenFlow 1.0 model [70]. The model provides a basic implementation of OpenFlow 1.3 devices, including OpenFlow 1.3 compliant switch nodes, controller, and secure channel. Due to the intrinsic complexity of the OpenFlow 1.3 standard, the model does not support all the features specified by the OpenFlow 1.3 specification standard [65].

However, the proposed model has features that allow the simulation of at least several important aspects of OpenFlow 1.3 based networks; moreover, it provides a very good basis for further extensions as well as for the development of a complete tool for the future. The model structure is built using the NED languages, while the functionality

is built completely in object-oriented C++. The model includes vital functionality that provides a scalable and reusable API. OpenFlow 1.3 processing pipeline is used in the model with the support for group, and meter tables as well as OXM. Figure 3.10 shows the UML diagram of the most important classes that compose the model. It is important to remark that the diagram only reports the most important data members and functions. Some details about the relationship among classes have been omitted due to space limitations. In this section, the model building blocks are described in terms of structure and functionality. This includes OpenFlow-compliant switch, OpenFlow controller, and OpenFlow messages for the communication between the switch and the controller over a secure OpenFlow channel.

The general structure of the OpenFlow 1.3 switch and controller is illustrated in Figure 3.15 and Figure 3.16 respectively. The proposed model implements the OpenFlow switch and controller nodes as compound modules. The switches can be connected with each other and with other nodes (e.g. routers, host, etc.) in order to compose a network. The controller module realizes a set of common functionalities to control and inquire an OpenFlow network. According to the OpenFlow specification all the FDs need to register with the controller using OpenFlow connection channel. This connection is used for exchanging OpenFlow messages between the control plane (OpenFlow controller) and the data plane (OpenFlow switches) in order to implement the forwarding behaviours. In the OpenFlow controller module, the controller creates a complete overview of all switches and links between the switches. The controller module calculates the best path for the destination node and instructs the switches to create flows that adhere to this shortest path.

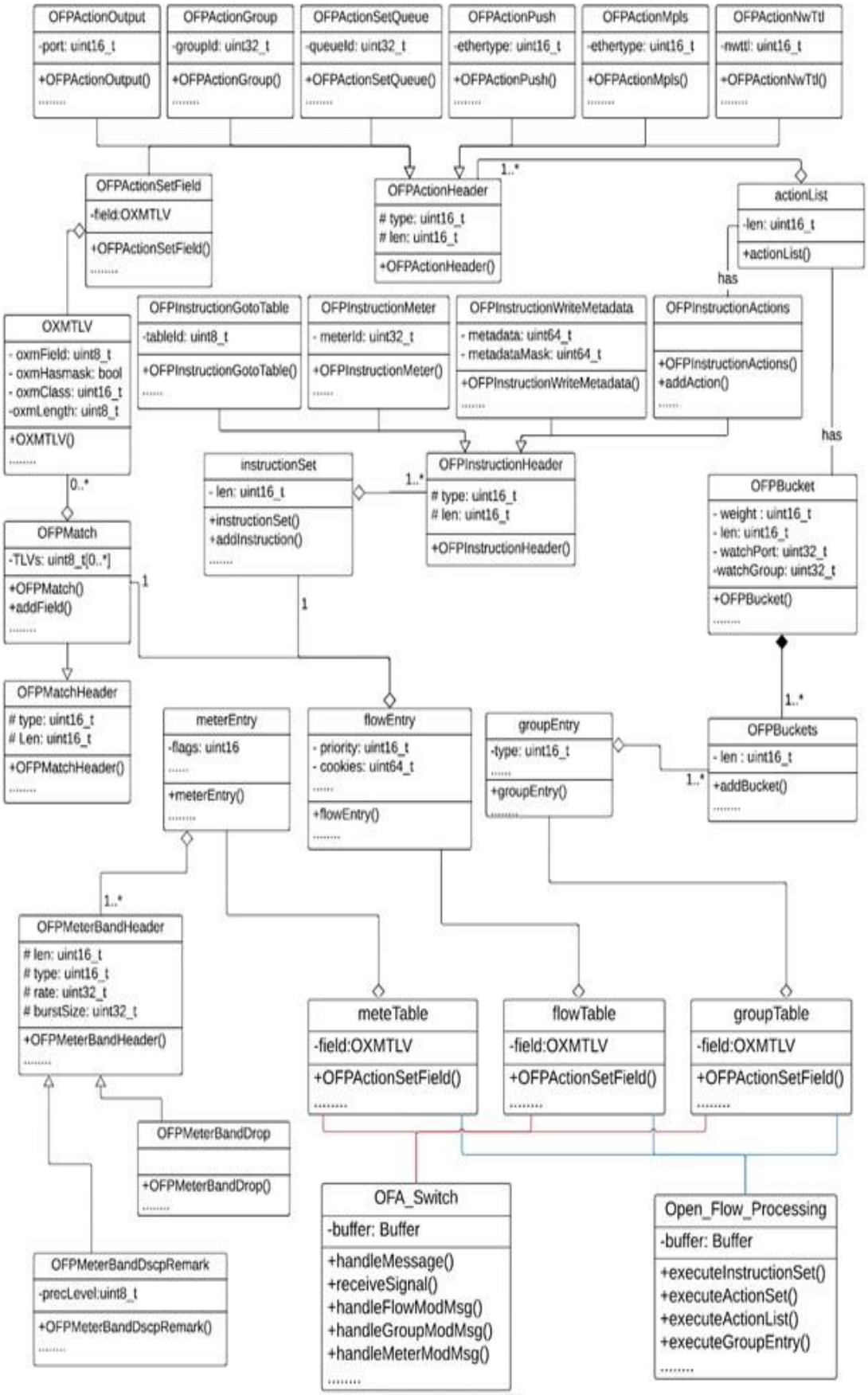


Figure 3.10: OpenFlow 1.3 Module Classes UML Diagram

3.4.2.1 OpenFlow Switch Node

OpenFlow Switch module is illustrated in Figure 3.15 and it consists of OpenFlow plugin and forwarding plane modules. The functionality of the OpenFlow plugin is implemented using the OFA_Switch module, while the forwarding plane consists of: Flow, Group, Meter tables, Buffer, and Open Flow Processing modules. The next sub-sections present an extensive and comprehensive description of the OpenFlow Switch module building blocks.

A. OFA_Switch Module

It represents the connection point of the Switch module with the OpenFlow controller module. The functionality of the module is implemented by the OFA_Switch class. At the beginning, the OFA_Switch module establishes a TCP session with the controller and negotiates the supported OpenFlow version and capabilities [70]. In fact, the OFA Switch module receives messages from both the controller module and the Switch's Open Flow Processing module. Controller messages are received through the TCP connection, while NO-MATCH-FOUND notification message are received from the Switch Open Flow processing module using the OMNeT++ signals concept.

In our opinion the most interesting and immediate use case for an OFA_Switch module is to aid the handling and validating of the controller messages. Therefore, the OFA_Switch Implements handleMessage() and receiveSignal() methods to handle the messages received from the controller and the Open Flow Processing module respectively. The handleMessage() is responsible for handling the messages that have arrived from the controller. This method works like a classifier, since it first checks if the received message is an OpenFlow message, and if not, then the received message is ignored and deleted at a later stage. Otherwise, it identifies the type of the OpenFlow message and calls the method that handles this type of message as shown in Figure 3.12. For example, if the controller wants to change the forwarding path for a certain flow, it sends OFP-Flow-Mod message to modify the flow-table of the FDs. When the OFA_Switch module receives the OFP-Flow-Mod message from the controller, it calls the handleFlowModMsg(). This method checks command field of the OFP-Flow-Mod message to identify if the controller wants to add, modify, or delete the flow-entry.

Vital functionality is introduced in the OFA_Switch module to perform a validation check for the OFP-Flow-Mod message before updating the flow-table as shown in Figure 3.11. If the message failed the validation check, an error message is sent to the

controller with the error type and code that is specified in OpenFlow 1.3 switch specification. The validation check is an interface that defines two main functions: ofpActionsValidation(), and ofpActionsValidateSetFieldReq(); the first function returns ofp-error-msg, if the output port is greater than OFPP MAX or out of the switch ports, if group action is specified in the action list. It looks up the group-id presented with the action in the group-table, and if no match is found, ofp-error-message is returned. The second function validates the set field action and returns ofp-error-msg if the request does not meet the prerequisite. The Group-Mod, and Meter-Mod messages are handled by following the same procedure used to handle the OFP-Flow-Mod message as shown in Figure 3.12.

The handleGroupModMessage(), and handleMeterModMessage() methods are used to handle the aforementioned messages respectively. Furthermore, OFA_Switch module implements functions related to the communication with the controller module, such as sending OFP-Packet-In messages. The OFP-Packet-In message is sent to the controller when the switch receives a packet and fails to match it against one of the flow-table entries, and the Open Flow processing module signals NO-MATCH-FOUND to the OFA Switch module as shown in lower part of Figure 3.12. Based on the switch configuration, the OFA Switch module either encapsulates the complete packet within the OFP-Packet-In message or buffers the unmatched packet and only sends the buffer-id with the OFP-Packet-In message [70]. In addition to performing all the above functionality, it is also responsible for the exchange of connect, OFP-Features-Request, OFP-Features-Reply, OFP-Multipart-Request, OFP-Multipart-Reply, and error messages with the controller module.

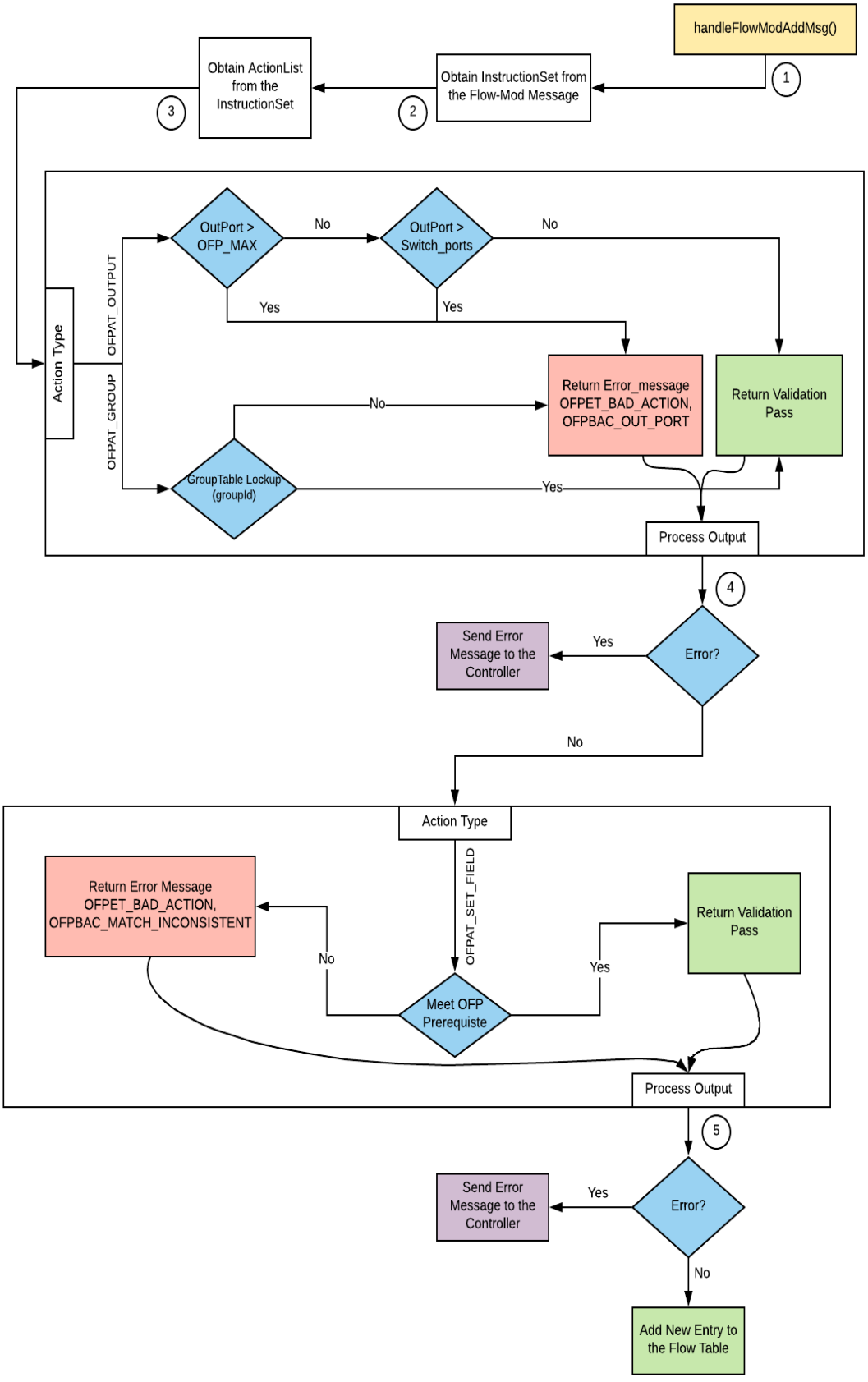


Figure 3.11: Actions Validation Procedure of the OFP-Flow-Mod Message

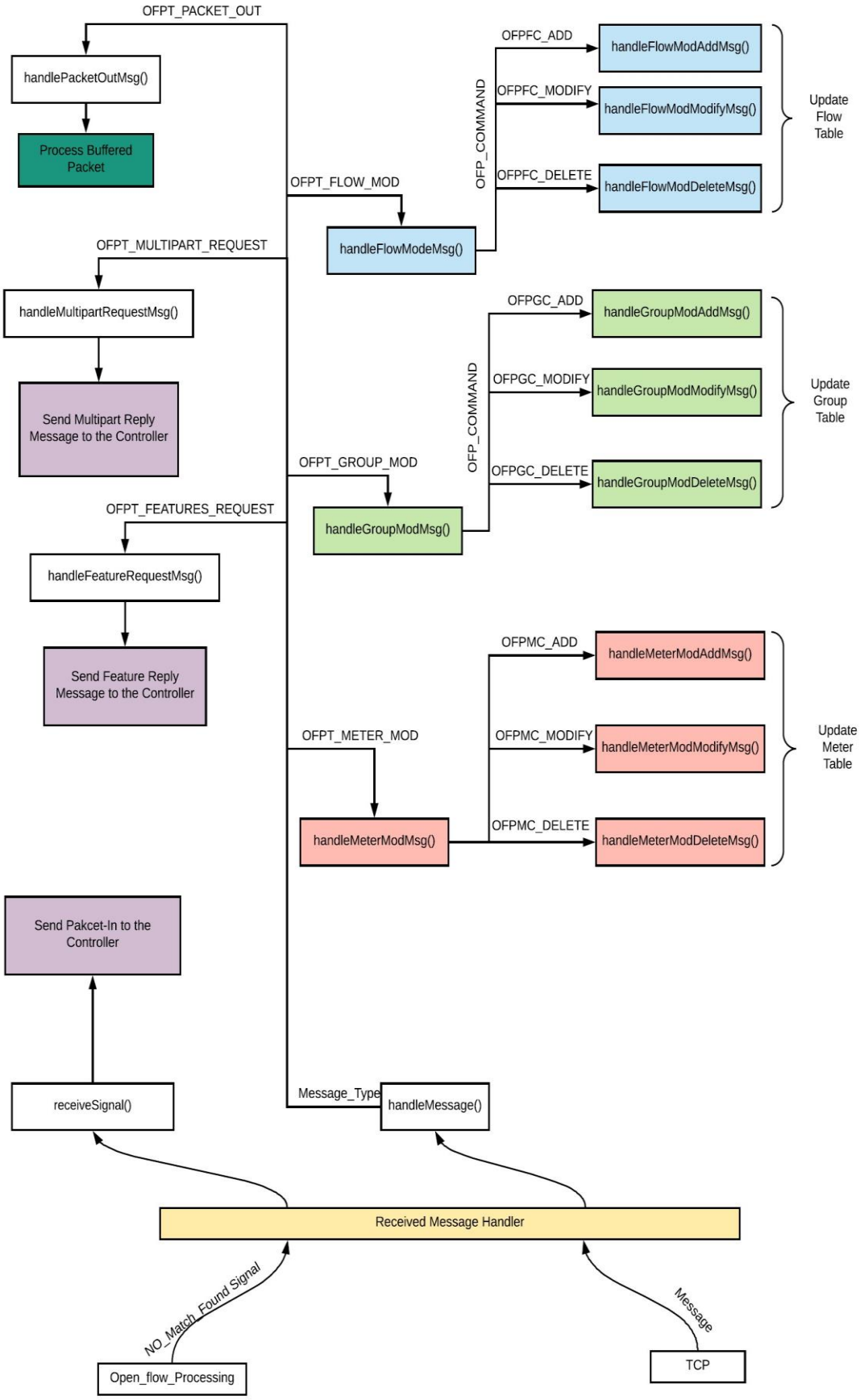


Figure 3.12: OFA_Switch Module handleMessage Procedure

B. Open Flow Processing Module

The module structure is build using NED language, while the functionality is implemented by the Open Flow Processing class. The Open Flow Processing class provides the OpenFlow 1.3 pipeline. It is necessary to mention that some of the Openflow 1.3 pipeline related features is not implemented, such as the support for multiple flow-tables.

The actual pipeline processing is triggered immediately after receiving a packet from the OFPort module. First Metadata, and empty actionSet are attached to the packet. Then match fields are extracted from the packet headers and used to query the flow-table. If headers of the received packet do not match any flow-entry, then NO-MATCH-FOUND signal is sent to the OFA_Switch module to notify the controller by sending an OFP-Packet-In message. Otherwise, the instructions attached to the matched flow-entry are executed. The method executeInstructionSet() of the Open Flow Processing class is called when the flow-table lookup finds a match. As shown in Figure 3.13, the method loops though all the instructions attached to the selected entry. It first checks the type of the instruction and based on the result it calls another method to execute this type of instruction. The instructions can send the packet to a meter table to enforce QoS on the packet, apply an action list immediately on the packet or update a packet actionSet. The flow instruction can also point to a group-table using OFPActionGroup. When the pipeline processing stops, the accumulated actionSet is executed on the packet.

For example, if the selected flow-entry has an instructionSet consist of two instructions: the first one is OFPInstructionMeter, and the second is OFPInstructionActions. The executeInstructionSet() method will check the type of the first instruction and then call the applyEntry() method of meter-table class. Then it checks the type of the second instruction and if the result is ofp-Instruction-apply-actions it calls executeActionList to execute the actionList on the packet. Otherwise, it calls writeActionsToActionSet to update the packet actionSet. The applyEntry method of meter-table class can pass, drop, or modify the prec-level of the packet. Therefore, the executeInstructionSet() will check if the packet is dropped by the meter before the execution of the second instruction.

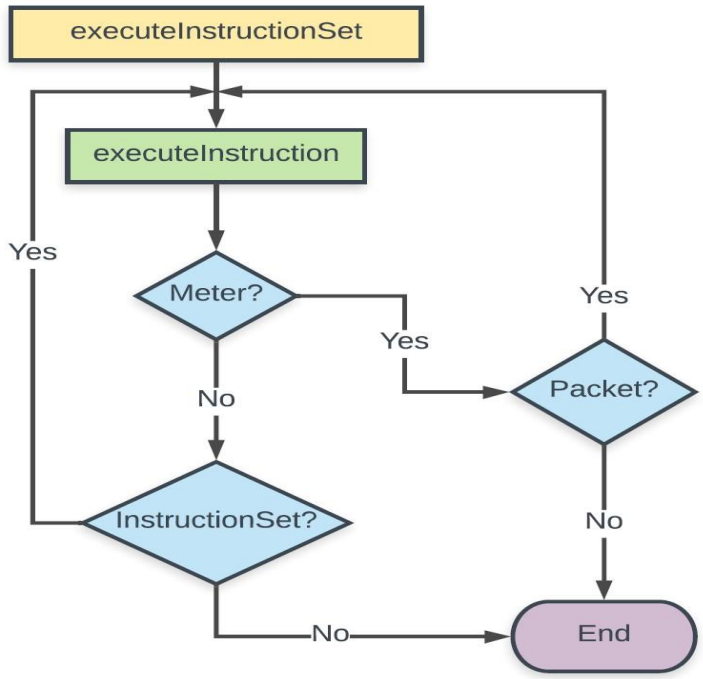


Figure 3.13: executeInstructionSet method Operations

C. Flow Table

This is a simple module without any gate. It represents the flow-table and each OpenFlow switch has an instance of this module. This module has methods to read, interact, update the flow-table. At the same time, it requires function calls because handle message is not implemented. This module works as a container for one of more flow entries. A vector from C++ standard library is used to store these entries and OMNeT++ Watch Marco is used to keep track of the table activity during runtime as shown in Figure 3.14.

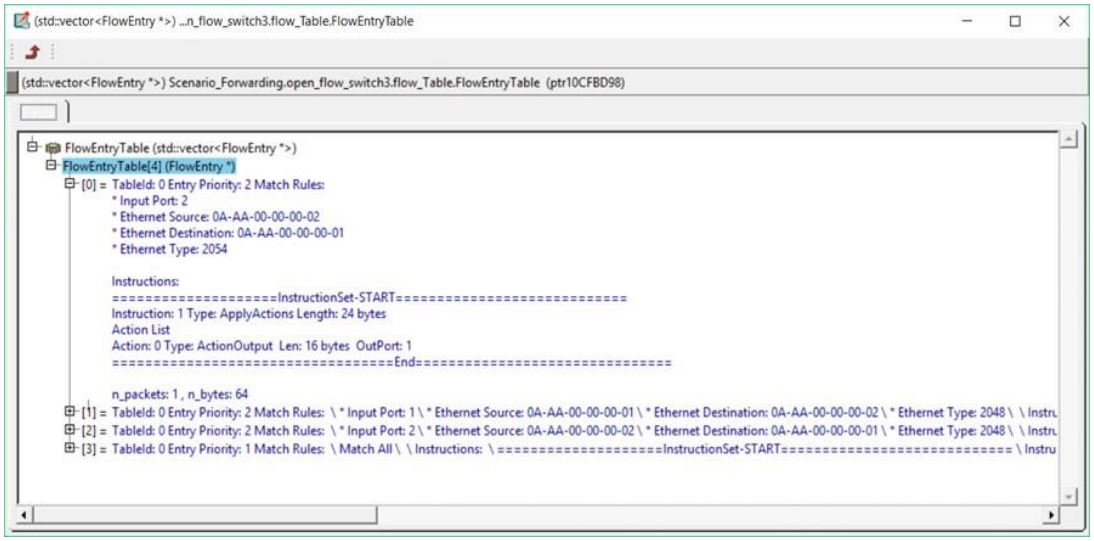


Figure 3.14: Flow Table content at simulation runtime

D. Group, Meter Tables Modules

Group and Meter tables have been added to the switch module. They are implemented by the group-table and meter-table classes respectively. These modules provide methods to query, modify, delete, add, and execute entries of the table. According to the OpenFlow 1.3 specification [65] a meter-entry consists of meter-id, one or more meter-bands, and counters. Meter-table can be pointed by the flow-entry to perform either simple QoS such as rate-limiting or more complex QoS such as Diffserv. Following the same design strategy, each group-table entry consists of group-id, action buckets, and counters. These tables are designed to prevent overlapping between entries and have a unique ID for each entry. Table manipulation error handlers are also available. Meter and group-tables are connected to the flow-table through an OpenFlow instruction.

E. Buffer

Very simple module is used to hold the received packet while the OpenFlow switch is asking the controller module for instructions. This module does not have a connection to any other module. The Open Flow Processing and OFA_Switch modules use a direct call to store and retrieve the packet from the buffer.

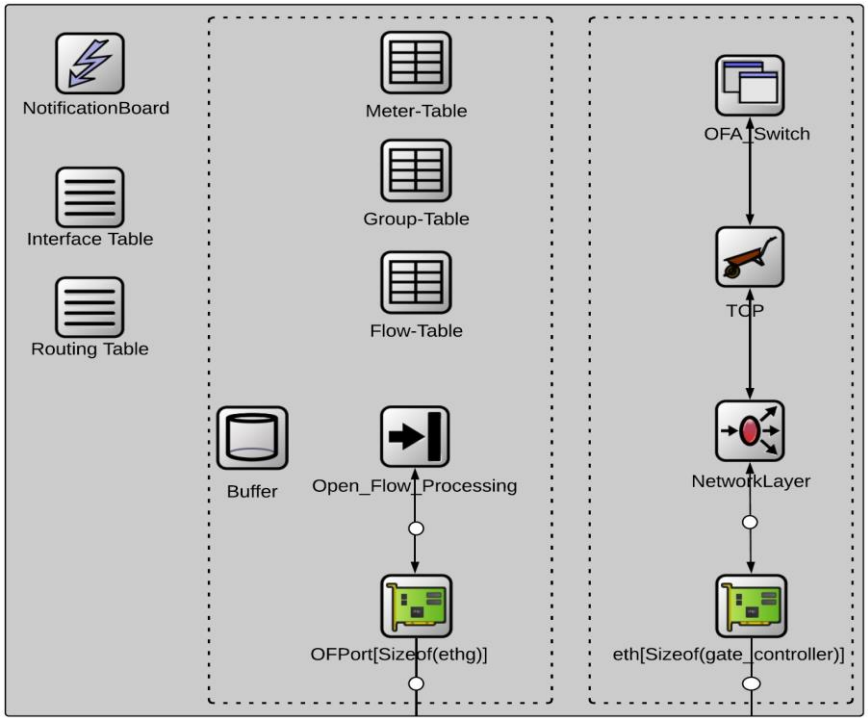


Figure 3.15: OMNeT++ Module of OpenFlow Switch

3.4.2.2 OpenFlow Controller Node

OpenFlow controller node plays a crucial role in the OpenFlow architecture. As illustrated in Figure 3.16, it consists of an OFA Controller module and a collection of applications realized in a separate module called CtrApp. In general, the controller node is responsible for the transmission and the reception of OpenFlow messages. In this respect, the module has the objective of modelling the transmission of OFP-Feature-Request, OFP-Multipart-Request, OFP-Packet-Out and OFP-Flow-Mod messages and the reception of OFP-Feature-Reply, OFP-Packet-In from the connected switches, Finally, it is also responsible for the exchange of connect messages during the initial setup procedure. In that context, the controller node sends an OFP-Feature-Request message to the data plane switches upon session establishment. This message is generated by the

OFA Controller module, so the CtrApp does not need to process this typically. The switch responds with an OFP-Feature-Reply message, which is also handled by the OFA Controller module, without any intervention from the CtrApp module, while the OFP-Packet-Out and OFP-Flow-Mod messages are sent based on the application's behaviour and needs. The application specifies the messages' command and asks the OFA Controller module to send it to the switch.

The communication between the OFA Controller module and the CtrApp is again realized via the OMNeT++ signal concept [69]. It is intended to provide a programmatic platform for controlling one or more OpenFlow switches. The most important tasks of the controller are the network discovery, Flow modification, and packet routing. OMNeT++ cTopology class is used by the controller Node to provide useful network functions such as network discovery, host tracking, and routing. The controller node is enhanced to provide a simple, easy to use and well-defined API that facilitates the creation of any new control and management applications.

The CtrApp contains a set of built-in applications such as Hub, Switch and Forwarding. If the network needs to be managed differently, new behaviour application needs to be written. The application tells the controller module how to manage the FDs. Then the controller module configures the FDs by using OpenFlow protocol. Figure 3.17 shows and example of the interaction between the individual entities of the controller module to perform network operations. In this example, a switch application is used as the brain

of the network but in the same way different applications can be utilized to alter the network behaviour.

The outer structure of OpenFlow controller node is kept the same as the previous OpenFlow 1.0 module, while new functionality has been adopted to support OpenFlow 1.3 message structures as well as the support for OXM based on [71]. Ongoing work is being carried out to create a controller application that simulates the behaviour of LTE core network (Evolve Packet Core). This application aims to enhance the 3GPP EPC performance by removing the GTP and providing DiffServ QoS for the user Traffic. In that respect, a new interface has been added to send OpenFlow meter and group modification messages to the connected switches. The controller sends OFP-Meter-Mod to modify the switch meter-table entries. In the same way OFP-Group-Mod is sent by the controller to modify the group-table of the FD.

OpenFlow modification messages are used for all the group and meter-table entry operations such as adding new entry, deleting existing entry, and modifying one of the table's OpenFlow instructions. OFP-Multipart-Request message is used by the controller to obtain statistics from data plane switches and these messages are used by the controller to gather the status of individual table, entry, or port from the data plane switches. OFA Controller provides the tools to obtain the data plane status and feed it to the application layer as needed or requested.

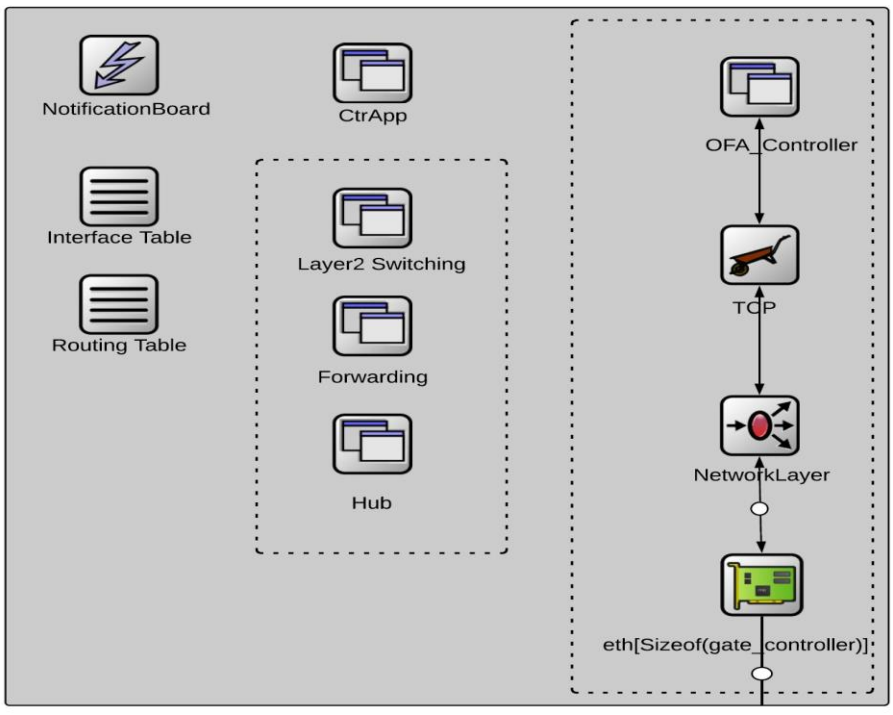


Figure 3.16: OMNeT++ Module of OpenFlow controller

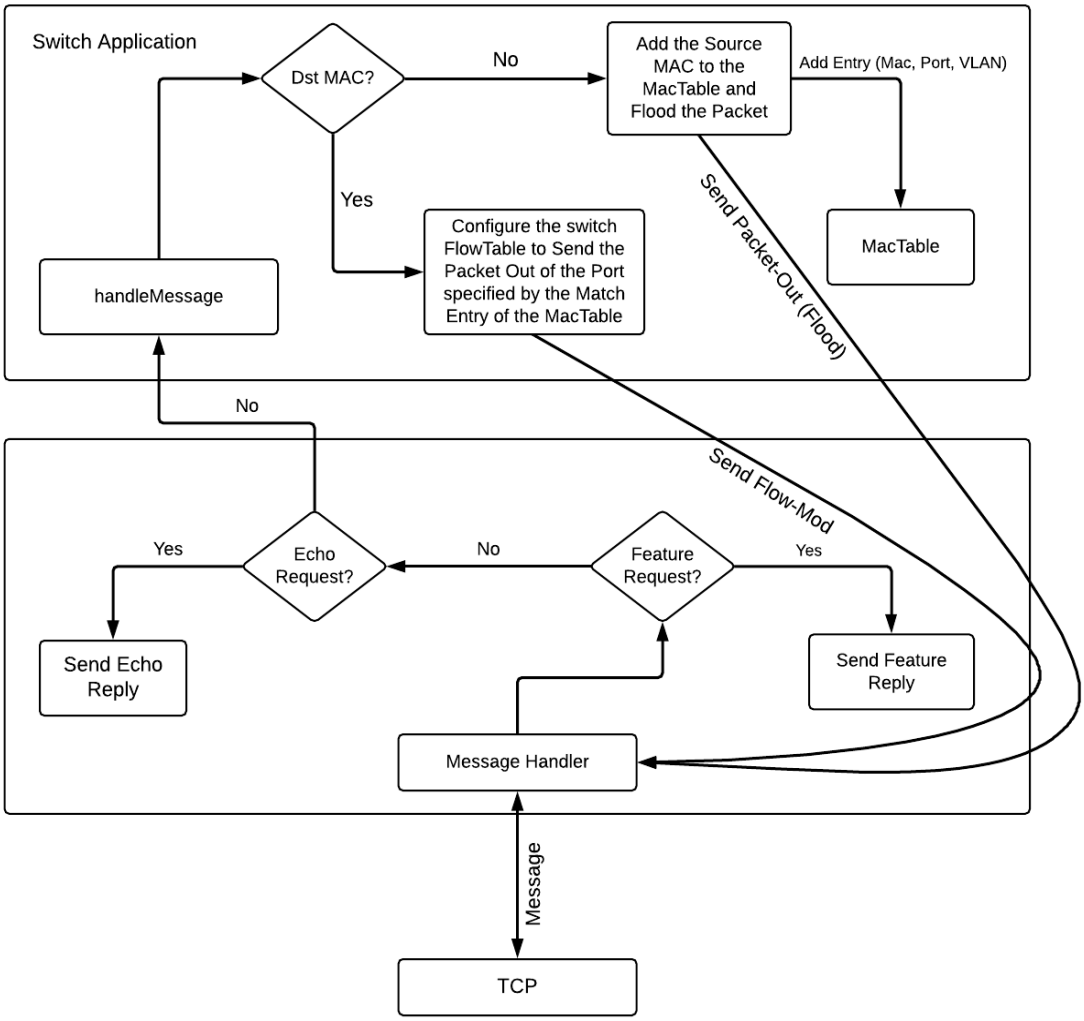


Figure 3.17: Controller Node Operation with Switch Application

3.4.2.3 OpenFlow Messages

OpenFlow supports three classes of communications: controller-to-switch, asynchronous and symmetric messages each with multiple sub-types. The controller-to-switch communication messages are purely initiated and sent by the controller and may or may not require a response from the OpenFlow-compliant switch.

As explained in previous sections, the controller communicates with the FDs using OpenFlow protocol utilizing OpenFlow messages. These messages are used for feature detection, configuration, and programming of the switches. They are also utilized by the controller to inspect the state of the connected switches.

Asynchronous communication messages are sent by the OpenFlow switch to the controller without any request. The switch sends asynchronous messages when: i) it receives a packet and does not know how to forward it; ii) notify the controller about

various state changes at the switch; iii) report errors. Some of the important asynchronous messages are OFP-Packet-In and OFP-Flow-Removed messages.

Finally, symmetric messages are initiated and sent without solicitation from either side, i.e., the controller or an OpenFlow compliant switch. These are OpenFlow messages like Hello and Echo request/reply that can be used to monitor the control channel and make sure it is healthy and available.

OpenFlow 1.3 model implements the most important subset of messages that allows exhaustive simulations of OpenFlow enabled networks. Every message in the OpenFlow 1.3 model inherits from OFP-Header message class, which contains basic OpenFlow header information, allowing an easy way to create new OpenFlow messages just by extending it. All the implemented messages in OpenFlow 1.0 model [70] that include the OFP-Features-Request and the OFP-Features-Reply, OFP-Packet-In, OFP-Packet-Out, OFP-Flow-Mod are updated to OpenFlow 1.3 and new messages to modify the group and meter-table have been implemented. Messages in OMNeT++ are defined as a C++ class. Figure 3.18 shows the class representation of the messages implemented in the OpenFlow 1.3 model.

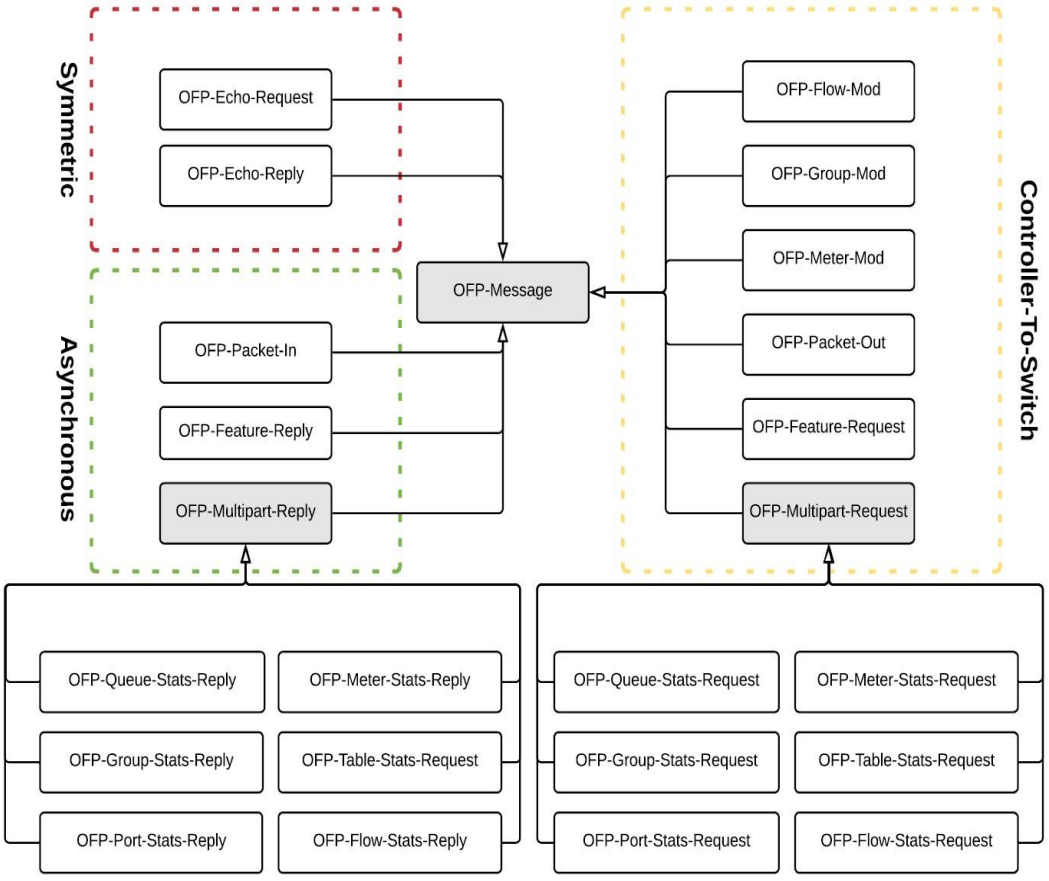
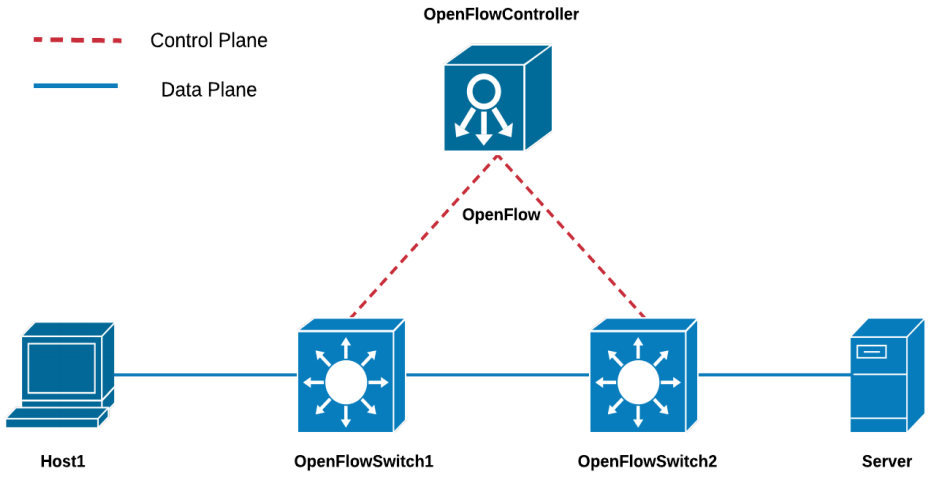


Figure 3.18: Messages Implemented in OpenFlow 1.3 Model

3.5 Openflow 1.3 Model Validation

To validate that Openflow 1.3 OMNeT++ model works according to the standard, a network consists of client, server, two Openflow switches, and Openflow controller has been simulated as shown in Figure 3.19. The client eth0 interface is connected to Openflow switch1 eth0, while the server eth0 interface is connected to Openflow switch2 eth0 interface. The switches are connected to each other by the mean of interface eth1. The controller is connected to both switches using TCP.



Openflow 1.3 based network topology

Figure 3.19: Openflow 1.3-based Network Topology

At the start of the simulation the Openflow switches setup a TCP session with the controller and maintain it during the simulation, then the server picks a random time between 0.1ms and 1ms to send UDP traffic to the client at the other end.

Figure 3.20 shows that after the setup of the TCP session the controller sends a FeaturesRequest message to all the connected Openflow switches, upon receiving the controller request message, each switch response back with FeaturesReply message.

Figure 3.21 shows the interaction between the controller and the switch. The server is configured to send UDP packets to the client IP address, therefore it first sends ARP message to know the MAC address of the client because this is an ethernet network. First the ARP packet is sent to the Openflow switch and because it doesn't have a flow entry at the beginning of the simulation, the switch sends the ARP packet encapsulated in Openflow PacketIn message to the controller. Upon receiving the PacketIn message, the controller application is written to

handle ARP request, therefore, the controller sends back two PacketOut messages to the switch in question.

- The first packet instructs the switch to drop the ARP request,
- The second message contains the ARP response to be send back to the Server.

When the server receives the ARP response, it starts sending UDP packet to the client address. The first packet received by the switch is send to the controller and the controller response with Flow_Mod message to configure the switch with a new flow entry to should be used to handle the upcoming traffic of this flow.

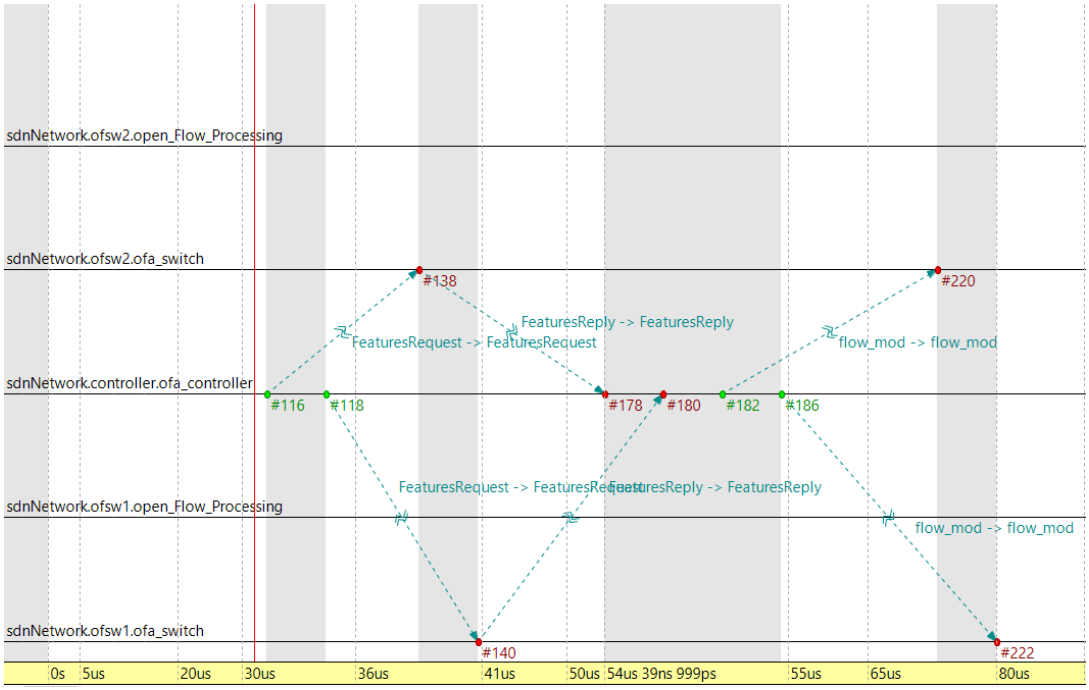


Figure 3.20: Openflow 1.3 Channel Setup Interaction

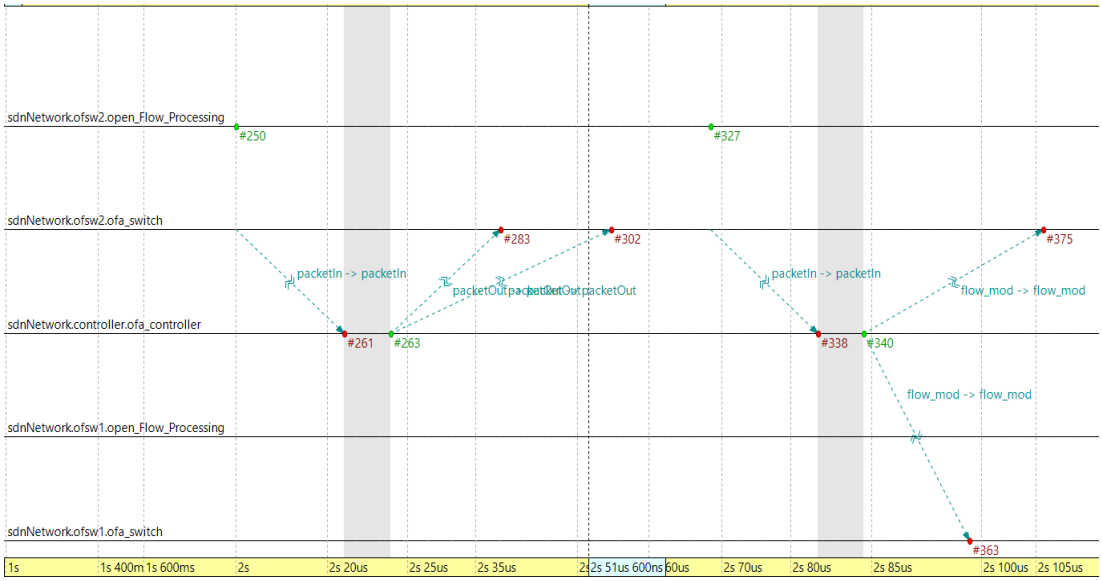


Figure 3.21: Openflow 1.3 First Packet Handling Procedure

Figure 3.20 and Figure 3.21 proof that the Openflow 1.3 model works based on the standard specification of Openflow 1.3. Also, serviceTime Parmenter is added to both the switch and the controller modules to simulate the processing time of these entities.

3.6 Summary

This chapter briefly describes the differences between the network technology that used nowadays and SDN that considered to be one of the emerging technologies that met to reshape the structure of networking. Then OpenFlow protocol described in detail with focus on the switch structure requirement based on OpenFlow specifications from 1.0 to 1.3. Finally, the chapter describes our OMNeT++ implementation for OpenFlow 1.3 protocol. The model describes the structure and the functionality of the controller and switch nodes. This section explains how the basic components of the OpenFlow node interact with each other's to perform different tasks that are required to simulate OpenFlow-based network.

CHAPTE 4

4 Evolved Packet System Technologies and Implementation Model

4.1 Introduction

This chapter discusses the EPS architecture and its simulation platform in OMNeT++. This chapter describes the building components, communication interfaces and protocols of the EPS, it also explains network procedures such as Initial Attachment, Access Bearer Setup and Release procedures, and X2-Handover. Finally, it briefly describes simuLTE and illustrates in detail the structure, operations, and functionality of several networking entities like eNodeB, SGW, PGW and MME.

4.2 Evolved Packet System

The EPS architecture has been designed to provide seamless IP connectivity between the UE and external Packet Data Networks (PDNs). The main functional elements and connection interfaces of EPS system architecture are illustrated in Figure 4.1. EPS consists of two important networks namely the access and core networks. The access network is also known as the Evolved Universal Terrestrial Radio Access Network (EUTRAN) and it consists of multiple base stations denoted as the eNodeB. The core network known as the EPC is consists of multiple functional elements that include: MME, SGW, PGW, Home Subscriber Server (HSS), and Policy and Charging Rule Function (PCRF) [72]. The eNodeB represents the essence of the access network and it is responsible for the air interface towards the UE and S1 interface towards the core network. Meanwhile, the EPC connects the mobile network to external PDNs that include: i) Internet; ii) IP Multimedia Core Network Subsystem (IMS); iii) private

corporate networks, which provides the UEs with a variety of services. The SGi interface is utilized to maintain the connection with the outside networks. The EPC is all-IP-Network that provides always-on connectivity to UEs and supports IP version 4, IP version 6, or dual stack IP version 4/version 6. When the User device is switched on, the EPC sets up a basic IP connection to the external world and maintains it until the device is switched off.

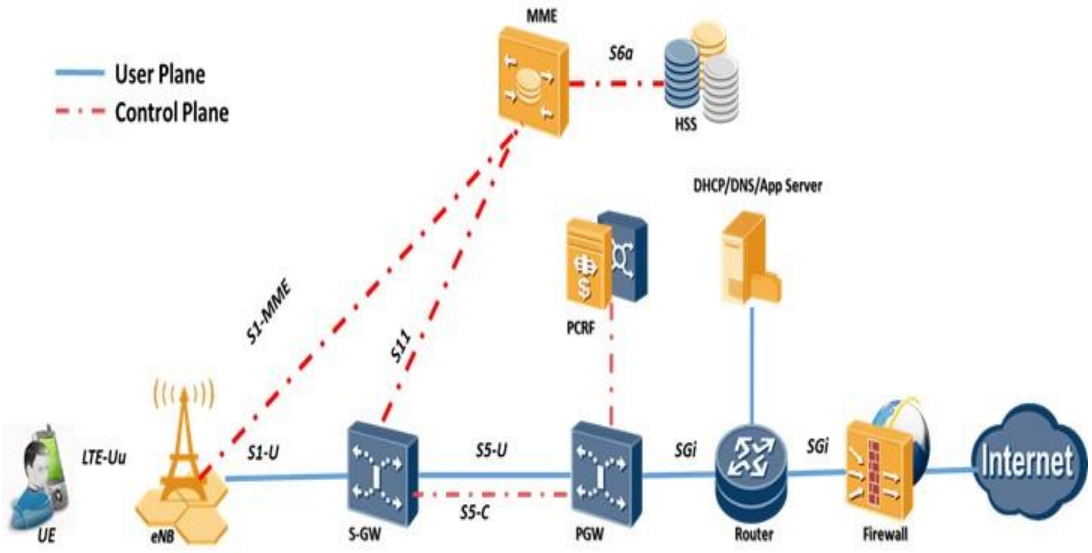


Figure 4.1: Main Component of the EPS Architecture

4.2.1 Evolved UMTS Terrestrial Radio Access Network

Figure 4.2 shows the basic building blocks of the EUTRAN. The EUTRAN has only one element known as the eNodeB and it is responsible for the communications between the UEs and the core network (EPC). The eNodeB performs two important functions that can be summarized as follows: First, it manages and organizes UEs' low level operation utilizing several control-signalling messages such as handover commands. Secondly, it handles the downlink and uplink radio transmissions to/from the UE using the analogue and digital signal processing functions of the LTE air interface [72].

The eNodeB represents the control point of UEs in one or more cells. In LTE, UE communicates with a single eNodeB cell at any one time. The eNodeB that is handling the UE communication is known as its serving eNodeB. Each eNodeB is connected to the EPC and nearby eNodeBs through the S1 and X2 interfaces respectively. These interfaces are not direct physical connections. In reality, control and data traffic are routed through multiple hops in the underlying transport network to reach the desired

target. The X2 interface is not mandatory and is used to exchange signalling and data traffic between source and target eNodeBs during UE handover. If eNodeBs are not connected through the X2 interface, then the S1 interface is used to handle the functionality offered by X2 interface, although indirectly and more slowly.

Several eNodeB types can be used as a part of the EUTRAN namely: Macrocell, Microcell, Picocell, and Femtocell [73]. The latter is a home eNodeB (HeNodeB) that is used to provide coverage for a small area like a home. HeNodeB is purchased by a user and belongs to a Closed Subscriber Group (CSG) and can only be accessed by mobiles with a Universal Subscriber Identity Module (USIM) that also belongs to the CSG. From an architectural point of view, HeNodeB can connect to the core network (EPC) through a direct connection like the other types of eNodeB or through an aggregation point known as a home eNodeB gateway.

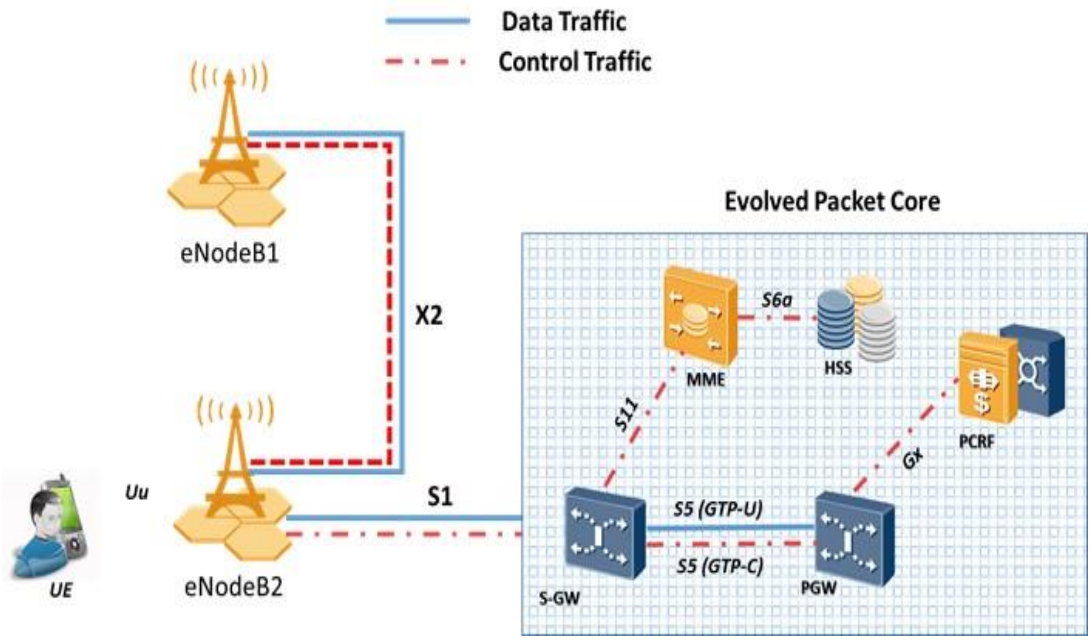


Figure 4.2: Evolved UMTS Terrestrial Radio Access Network Main Components

4.2.2 Evolved Packet Core

EPC represents 3GPP latest evolution in mobile core network architecture. The EPC is characterized as being a simplified IP based mobile core network with a flat architecture and supporting packet switching only. It uses a large variety of IP-based protocols to transport all services. As previously mentioned, the EPC consists of several network function elements that include the HSS, SGW, PGW, MME, and PCRF. Figure 4.3 shows the main components and the communication interfaces of the EPC.

4.2.2.1 Home Subscriber Server

The Home Subscriber Server (HSS) is a centralized database for UEs. It maintains user-related and subscriber-related information. This includes identification, security, location, and profile parameters, which are used for mobility management, call and session setup, user authentication and access authorization.

4.2.2.2 Serving Gateway

The SGW acts as a router that represents the interconnection point between the radio-side and the EPC since it is logically connected to the EUTRAN through the S1-U interface and to another gateway, specifically the PGW through the S5 interface. It deals with the user plane that includes IP data traffic forwarding between the eNodeB and PGW and vice versa. It is also characterized for being a local mobility anchor for UE handovers between eNodeBs and also between LTE and other 3GPP accesses. A typical mobile operator might have several SGWs, where each of them is responsible for all the UEs in a specific geographical region. During the UE Initial Attachment procedure, it is assigned to a specific SGW, but the SGW can be changed if the UE moves away from the geographical region covered by the current SGW.
It also participates in some of the control plane operations that include S1-U and S5-U tunnel establishments.

4.2.2.3 Packet Data Network Gateway

The PGW represents the point of interconnect between the EPC and the external IP networks. The PGW together with the SGW enables data traffic forwarding between the UE and external IP networks. The PGW also performs various functions that include: i) UE IP address allocation; ii) policy enforcement; iii) packet filtering. The PGW routes packets to and from the PDNs through the SGi interface. Each PGW is connected to the external networks that include network operator's servers, Internet or the IP multimedia subsystem. APN is used as an identifier for a PDN. In a typical mobile network deployment, network operator uses several distinct APNs to differentiate between its own services and the Internet. In the Initial Attachment procedure of each UE, a default PGW is assigned to provide an always-on connectivity to a default PDN such as the Internet. However, it also possible for a UE to be assigned to additional PGW gateways when it wants to use a different service that may be offered

by private corporate networks. Each PGW stays the same throughout the lifetime of the data connection.

4.2.2.4 Policy and Charging Rule Function

PCRF represents an important network element that is responsible for charging and policy control, as shown in Figure 4.3 where the Gx interface is used to connect the PCRF to PGW in the EPS architecture. The PCRF provides guidance for the policy and charging treatment that a particular service data flow should receive during initial bearer setup or modify the QoS parameters of an existing bearer. This is done by either referring to a predefined Policy and Charging Control (PCC) rule, or by composing a dynamic PCC rule. Upon receiving the PCC, the Policy and Charging Enforcement Function (PCEF) which is part of the PGW implements the decision. PCRF has multiple signalling interfaces that are mainly used diameter applications, in a similar way to the interface between the MME and the HSS. Figure 4.3 shows only the Gx interface, but it also has the Gxx interfaces to the PCEF and Bearer Binding and Event Reporting Function (BBERF), the Rx interface to the application function, and the S9 interface between the home and visited PCRFs.

4.2.2.5 Mobility Management Entity

The MME represents the main control element in the EPS architecture. It deals with the high-level control operations of the mobile that include sending control signalling related to mobility and security for the EUTRAN. The same as the SGW, a typical deployment of the mobile network may have multiple MMEs each one is in control of a specific geographical region. Every UE is managed by a single MME that is denoted as the serving MME, but that can be changed if the UE moves outside of the geographical region covered by the current MME.

The MME is involved in the UE authentication and authorization and it is also responsible for the tracking and paging procedures of UE in the idle mode since it keeps track of UE location and state information. The Non-Access Stratum (NAS) signalling terminates at the MME and is responsible for ciphering and integrity protection of these NAS messages. It is also responsible for the generation and allocation of temporary identities to UE, which is then used in all subsequent procedures to identify the UE in the network. The UE changes its state to idle-mode after a period of inactivity, and the MME is responsible for paging it in case of network-initiated events. The MME is also

characterized for being the element in the EPS architecture that handles handovers between eNodeBs as well as to other access networks. After passing the security check the MME manages the process of data bearer establishment and teardown. This process involves selecting the appropriate SGW and PGW for a UE at the Initial Attachment procedure and relocating the SGW during intra-LTE handover that is required for changing the current SGW.

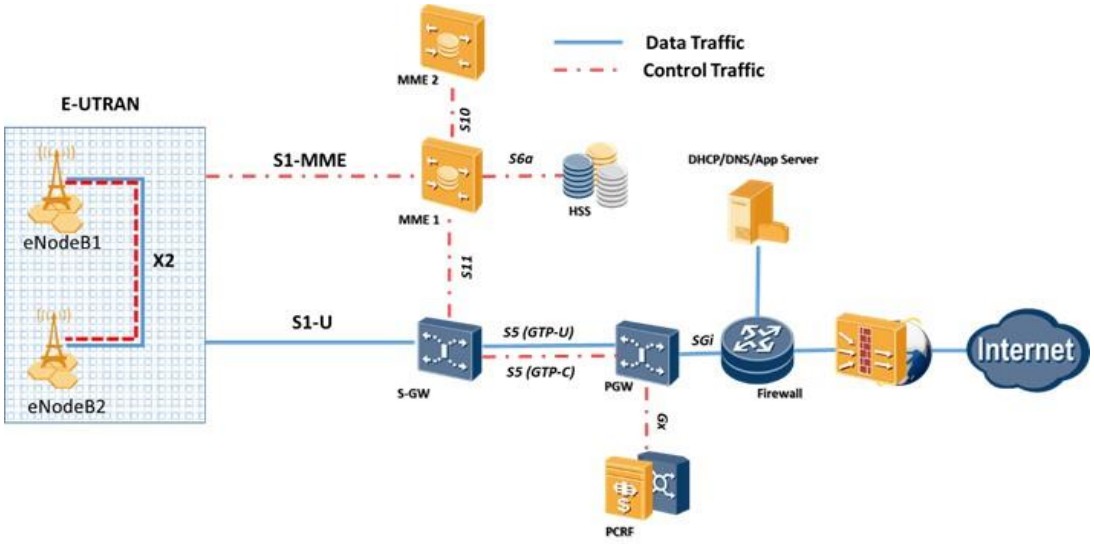


Figure 4.3: EPC basic network function elements and the connection interfaces

4.3 Interfaces and Protocol Stacks

A protocol stack is associated with each interface of the EPS. This protocol stack has two planes, namely: protocols in the Control plane and protocols in the User plane. These protocols are used by network elements to exchange signalling messages and forward data traffic respectively.

4.3.1 Signalling Protocols

Figure 4.4 shows the signalling control protocols used in the EPS network. These include control protocols used over interfaces in both access and core networks. The access network includes protocols such as the RRC protocol which represents the signalling protocol used in the air interface between the UE and the serving eNodeB.
Figure 4.6 shows the protocol stacks over the X2 interface in control and user planes, X2-AP is the signalling protocol used over the X2 interface to allow the neighbour eNodeB to communicate with the serving eNodeB. Also, each UE utilizes two signalling protocols that lie under the umbrella of the NAS protocol to communicate

with the MME through the air interface. These protocols are used to control the way the network handles traffic from the UE to the outside and vice versa, and it is also used to maintain an updated information about the UE in the EPC network. These protocols are formally known as the EPS Session Management (ESM) and the EPS Mobility Management (EMM). ESM and EMM messages travel from the UE to the core network embedded in a lower-level RRC and S1AP messages over the Uu and S1 interfaces respectively.

The protocol stack for UE communication with the eNodeB and MME is illustrated in Figure 4.5. In the core network, the MME represents the main control element. It has interfaces that connect it to multiple elements such as another MME, SGW, HSS, and eNodeBs. The MME controls all the eNodeBs that are registered with its pool area by using the S1 application protocol (S1AP). It also uses the Diameter protocol to exchange control information with the HSS over the S6a interface. Basic Diameter is a protocol standardized by the IETF based on the Remote Authentication Dial-In User Service (RADIUS) and is used for authentication, authorization, and accounting.

The Diameter protocol is implemented on the S6a and Gx interfaces between the MME and the HSS and between the PGW and PCRF respectively. diameter used on these interfaces represents an enhanced version of the basic diameter to provide the necessary features required by the EPS network. GTP Control part (GTP-C) represents the main protocol implemented by several interfaces in the EPS core network (EPC). This includes S10 interface between 2 MMEs, S11 interface between the MME and the SGW, S5/S8 interface between the SGW and the PGW. The GTP-C contains a mechanism to provide a peer-to-peer communication between the aforementioned EPC elements over the S10, S11, S5/S8 interfaces. It is also used to set up, manage and teardown the GTP-U tunnel over the aforementioned interfaces. EPS uses GTP-C version 2, commonly known as GTPv2-C, while version 0 and 1 of GTP-C is used in older mobile networks such as 2G and 3G. Specifically, 2G before release 99 uses GTP-C version 0 which is also known as GTPv0-C, while 2G after release 99 and 3G networks uses GTP-C version 1, denoted as GTPv1-C.

4.3.2 User Plane Protocols

The EPS data plane represents the mechanisms to correctly forward the traffic from the UE to the PGW and vice versa. It also provides the tools to precisely change the traffic direction as a response to the UE movement from point A to point B in the network. A

well-known protocol is utilized by the 3GPP to implement the data plane mechanisms. To be more specific the GTP User part (GTP-U) is the main protocol used in the EPS to implement the data plane forwarding. In EPS only version 1 of the GTP protocol is used, and it is mostly known as GTPv1-U, while in older generations of the mobile network such as 2G and 3G version 0 is used, where it is commonly referred to as GTPv0-U. In EPS, GTP represents the main data transport protocol in the S1 interface, while S5/S8 can use either GTP or Generic Routing Encapsulation (GRE) as the data transport protocol. This thesis, only explains the EPS implementation when the S5/S8 is implemented by using GTP. GTP is described in more detail in Section 4.3.3.

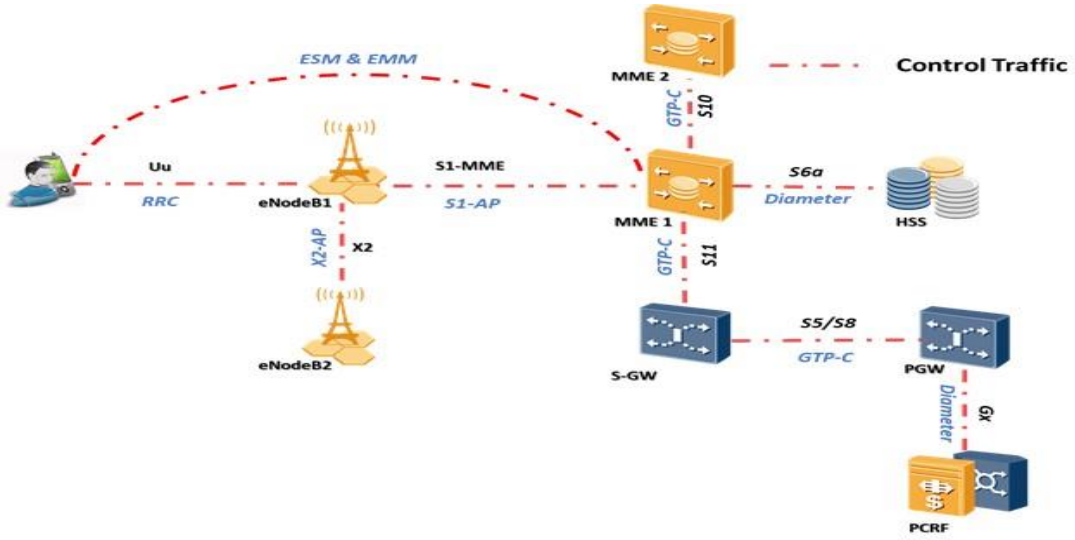


Figure 4.4: EPS communication interfaces and protocols

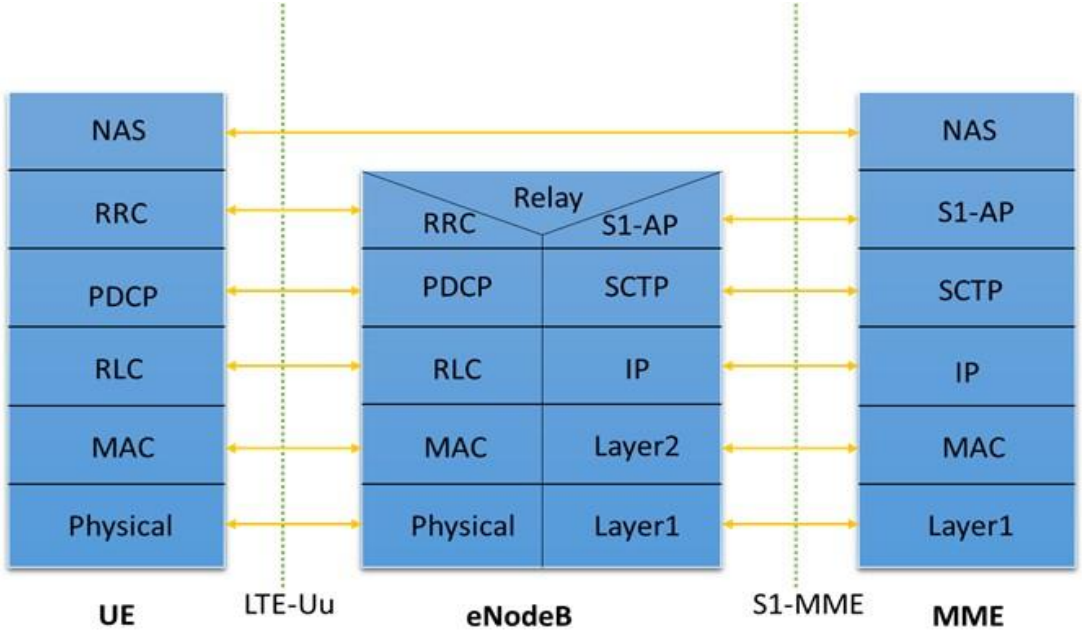


Figure 4.5: Control Plane Protocol stack for compunctions between a UE and MME

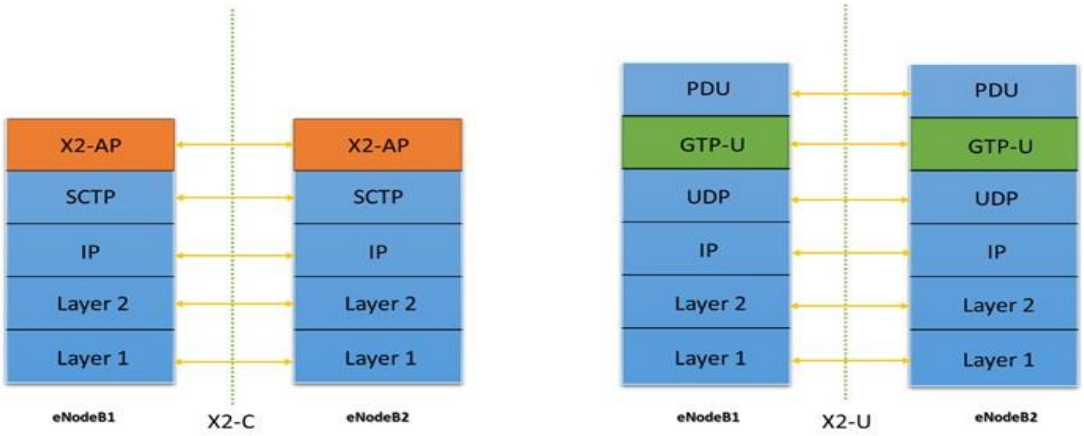


Figure 4.6: Data and Control Plane Protocol stacks for X2 Interface

4.3.2.1 Data Transport

Figure 4.7 shows the end-to-end protocol stack used in the EPS to provide the UE with an always-on connectivity to the PDN. As previously mentioned, Section 4.3.2, will focus only on the EPS implementation when the S5/S8 interface uses GTP. The downlink traffic is used as an example to explain how the user traffic is delivered from the Internet service to the UE mobile device. During the Initial Attachment procedure, the UE is assigned an IP address from the same address space of the PGW. The Internet routes data packets destined to a mobile device to its PGW, and when the traffic is received by the PGW, it first identifies the SGW that is currently looking after the mobile device, then it encapsulates the incoming packet with another IP packet and sends it to the SGW IP address, upon receiving the packet by the SGW. It decapsulates the incoming packet and repeats the same process on the S1 interface toward the eNodeB. Finally, the eNodeB uses the air Interface to properly deliver the packet to the mobile device.

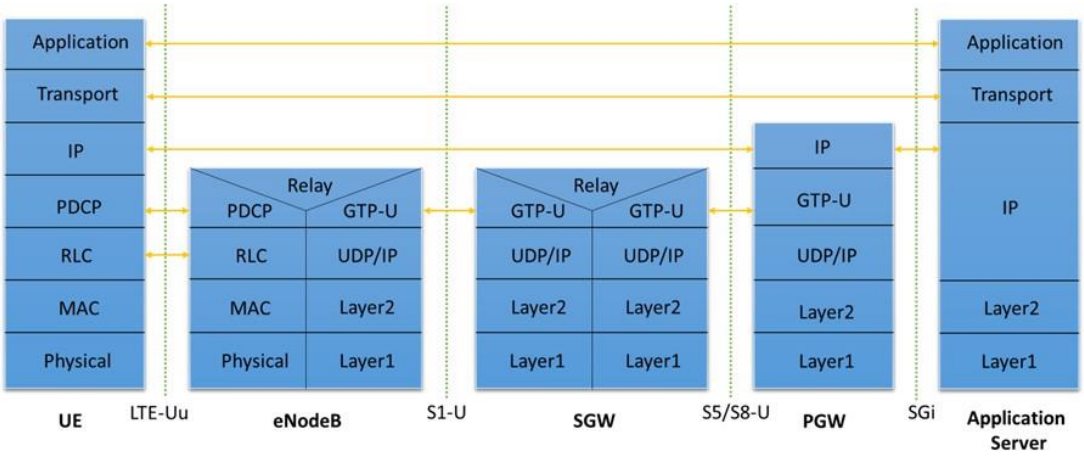


Figure 4.7: User Plane Protocol Stack

4.3.2.2 EPS Bearer

Users' traffic in the EPS is forwarded from one node in the network to another by utilization of what is called bearers. EPS Bearers is a bi-directional data pipe which is used to forward data traffic from UE to the PGW and vice versa, while it guarantees a certain level QoS. The term QoS refers to the network resource used to transfer the data to ensure that the data meets a specific level of data rate, error rate, and delay. EPC bearer carries User traffic that belongs to one or more services classified by a service data flow. In turn, each service data flow consists of one or more packet filters, and each filter represents a data flow such as the audio and video streams that represent a specific service.

In EPS, the same QoS is applied to all the traffic handled by the same EPS bearer. Based on the QoS, the EPS bearer can be classified to either Guaranteed Bit Rate (GBR) or non-GBR bearer. The name GBR bearer is always associated with a Guaranteed Bit Rate that represents the average data rate that should be received when using this type of bearer. GBR bearer is commonly used for a real-time traffic such as voice. Differently, non-GBR bearer offers no data rate guarantees and is mainly used for non-delay sensitive traffics such as web browsing. Also, EPS bearer is most commonly known as a default or dedicated bearer, where default bearer can only be a non-GBR bearer while the dedicated bearer can be either GBR or non-GBR bearer.

An always-on default bearer is assigned to each UE through the Initial Attachment procedure described in Section 4.4.1. In this procedure, after the UE is successfully registered to the network, IP address and a default bearer are assigned to the UE to provide it with always-on connectivity to default PDN such as the Internet. At this stage, the UE has an IP address and a default EPS bearer to the default PDN network. The UE may also request or be assigned one or more dedicated bearers that connect it to the same network. Dedicated bearers do not require a new IP address to be assigned to the UE instead it runs as a child to the default bearer and shares the same IP address. As previously mentioned, a dedicated bearer is normally used to guarantee a certain level of QoS. UE may have up to 11 EPS bearers to get connectivity to different networks with QoS differentiation.

The EPS bearer spans through multiple interfaces, namely Uu, S1 and S5/S8 interfaces. Therefore, it consists of the compensation of the radio, S1, S5/S8 bearers as shown in Figure 4.8. Each bearer is associated with QoS parameters that are used to maintain the

delay, error and data rate guaranteed by the EPS bearer. The radio bearer QoS is implemented by properly configuring the air interface protocols, while the S1 and S5/S8 bearers are implemented by utilization the GTP-U tunnels. It is also worth mentioning that the radio bearer is combined with the S1 bearer and is sometimes referred to as the Evolved Radio Access Bearer (E-RAB) as shown in Figure 4.8.

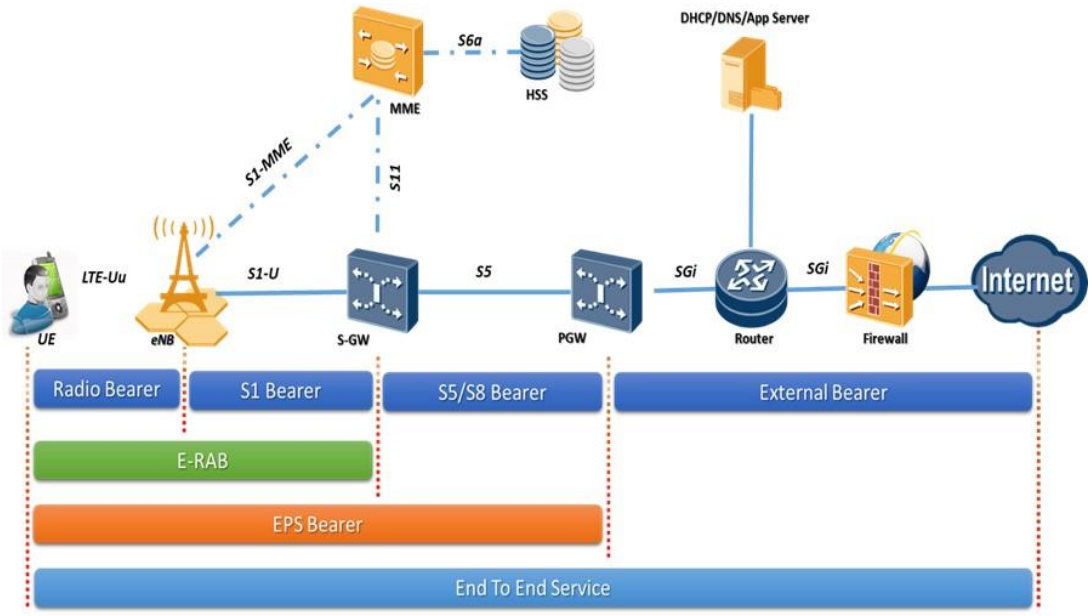


Figure 4.8: EPS Bearers

4.3.3 GPRS Tunnelling Protocol

GTP is a collection of protocols used by 3GPP Packet core networks (GPRS/UMTS/EPC) to play a major role in User session establishment, traffic forwarding, and mobility. It can be classified as GTP-C, GTP-U, and GTP variants. The GTP-C represents the control part of GTP and it defines the language used by the EPC control plane to exchange control information. The GTP-C has three versions: 0, 1, 2 and operates on top of UDP as the transport protocol. GTP v2 offers backward compatibility only with GTP v1 but explicitly offers no support for fall back to GTP v0. GTP-U represents the user part of GTP and defines the mechanisms for data forwarding between network elements in the EPC network. It has only two versions: 1 and 2 and like GTP-C, it uses UDP as the transport protocol. Finally, GTP which is also known as GTP Prime is used for interfacing with Charging Gateway Function (CGF) in GPRS and UMTS networks. In the EPS, GTP-C is used in the S11/S5 interfaces between the MME, SGW and PGW nodes to exchange control plane

signalling, while GTP-U is used in the S5/S1-U interfaces between the PGW, SGW and wired part of the eNodeB to forward user traffic in the data plane.

4.3.3.1 Tunnelling Using GTP

In EPC, GTP is used to implement a mapping between the S1 and S5/S8 bearers, and these bearers are implemented between the PGW, SGW and the wired part of the eNodeB. As previously mentioned, a bearer is a bi-directional data pipe, this pipe is represented by a GTP tunnel. Each tunnel utilizes two unique identifiers denoted as TEIDs, one for the uplink and other for the downlink. During the session establishment procedure, the GTP-C signalling messages are exchanged between the involved network elements to set up and store the TEIDs at both ends of each tunnel. For a better understanding of how EPS bearers operate, downlink data flow coming from the Internet and destined to a UE is used as a demonstration example.

As shown in Figure 4.9 the UE has established two bearers to handle two types of traffic namely, video and email traffic. The reason for establishing a different bearer for each traffic is because they require different QoS. It is necessary to differentiate between the incoming traffic in order to assign each traffic to the correct EPS bearer. This is done by utilizing the Traffic Flow Template (TFT) associated with each EPS bearer. Each TFT consists of a list of packet filters. Each filter represents a flow in the EPS bearer. Packet Filters use information like the source and destination IP address, transport layer source and destination port numbers, protocol type to match a flow. The filters may use a range of the above-mentioned information to match a traffic as well. When the Internet traffic has reached the PGW, the PGW inspects the incoming traffic and compares it with all the packet filters that it has. This way the traffic is always assigned to the correct EPS bearer, and then the PGW lookups the GTP-U tunnel information.

The most important information is the downlink TEID and the UE's SGW IP address. Then the Packet is encapsulated in a new GTP header that includes the downlink TEID and new IP layer headers that include the SGW IP address as the destination IP address with a new layer 2 headers and then the packet is forward to the SGW. When the packet is received by the SGW, it first inspects the GTP-U header and extracts the TEID and uses it to identify the correct EPS bearer and lookups the destination eNodeB IP address and TEID. The packet is then forwarded to the desired eNodeB following the same procedure performed by the PGW previously described. A similar process happens with the uplink traffic from the UE to the Internet.

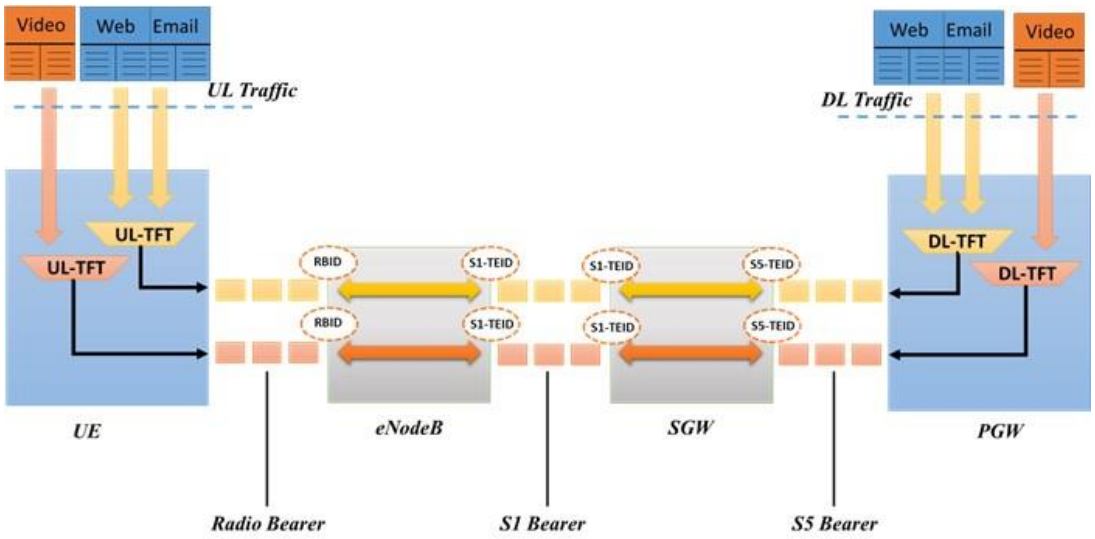


Figure 4.9: Traffic Flow Template to EPS Bearer to GTP-U tunnel mapping

4.4 EPS Procedures

This section describes the essential and most important procedures used in EPS networks to provide connectivity to end users. These include EUTRAN initial attachment, Access Bearer Release, Service Request, and X2-based handover procedures.

4.4.1 EUTRAN Initial Attachment Procedure

The initial attach procedure starts immediately after switching the mobile device on and it has four main objectives that include: i) register UE to the network; ii) sets up and configure air interface; iii) acquire an IP version 4 address and/or an IP version 6 address; iv) sets up the default EPS bearer for the UE. Subsequently, EPS enables the UE to have always-on connectivity to the outside world. Figure 4.10 illustrates the call flow for the Initial Attach procedure in the EPS network. A brief description of the Initial Attach procedure, including the main messages and their important Information Elements (IEs), is provided below:

- The Initial Attachment procedure starts with the UE sending an Attach Request Message to the eNodeB, where this message contains the UE's International Mobile Subscriber Identity (IMSI), UE Capabilities that indicate the supported NAS and AS security algorithms, attach type, RRC parameters, and ESM message container for requesting data connectivity. The ESM Message

Container includes the requested PDN type whether it is IPv4, IPv4 or IPv4v6, Protocol Configuration Options (PCO) and security parameters.

- When the Attach Request message reaches the eNodeB, a unique identifier called the eNodeB UE S1AP ID is assigned to the UE, and after a successful selection of the MME, the NAS message along with information on the current location of the UE is encapsulated in an S1AP: Initial UE Message and forwarded to the MME.
- Upon receiving the S1AP: Initial UE Message, the MME starts the authentication and security procedure by requesting the authentication vectors from the HSS using the authentication Information provided by the Request message. The HSS sends an Authentication Information Answer message that includes the requested information back to the MME, and after acquiring the authentication vectors from the HSS, a sequence of Authentication Request and Authentication Response messages are exchanged between the MME and UE to mutually authenticate each other. On a successful authentication, the MME sets up the NAS security parameters towards the UE in order to enable ciphering and integrity protection of further NAS messages, then a unique identifier known as the MME UE S1AP ID is assigned to the UE by the MME. A combination of eNodeB UE S1AP ID and MME UE S1AP ID can be used to identify the S1-MME connection for a UE.
- An Update Location Request Message is sent by the MME to the HSS to confirm the successful registration of the UE to the network and to request subscription information for the UE.
- The HSS sends an Update Location Ack which is sent back to the MME as an acknowledgement to the MME request. This message contains the UE subscription information which includes the subscribed PDN type, QoS profile and APN. By utilizing the received information, the MME validates the UE's presence in the Tracking Area (TA) and services requested by the UE. If the checks are OK, then a new context is created for the UE on the MME.
- At this stage, the UE is successfully registered to the network and the MME has started the SGW selection and default EPS bearer creation process. This process is started by the MME by sending a Create Session Request towards the selected SGW requesting to set up and configure the default EPS bearer. This message

includes the UE IMSI, EPS bearer ID, IP address of the PGW, APN and subscribed QoS Parameters.

- Upon receiving the Create Session Request from the MME, the SGW creates a new entry in its EPS bearer table and sends a Create Session Request toward the PGW over the S5 Interface. This message contains UE IMSI, SGW Address for the User Plane, SGW TEID for the data and control planes, APN, Default EPS Bearer QoS, and PDN Type along with other parameters.
- When the Create Session Request message is received by the PGW and if the dynamic PCC is implemented, then IP-CAN session establishment procedure is performed by the PGW to obtain the default PCC rules for the UE from the PCRF. This procedure requires the PCEF, which is part of the PGW to send UE IMSI and IP address, User Location Information, APN-AMBR, and Default EPS bearer QoS along with other parameters to the PCRF. The APN-AMBR and QoS parameters associated with the default bearer may be altered in the PCRF response to the PGW. If dynamic PCC is not implemented, then PGW applies local QoS policy. The PGW enforces the policy and generates a Charging ID for the bearer to enable charging of the subscriber and sends Create Session Response to the SGW. In the message, the PGW most importantly includes the PGW TEID and
- Address for the data plane, PGW TEID for the control plane, EPS Bearer Identity, EPS Bearer QoS, and Charging Id along with other parameters. The TEIDs and the IP address are used to establish the S5 tunnel between the SGW and PGW.
- Upon receiving the Create Session Response message from the PGW, the SGW creates a new Create Session Response message and forwards it to the MME. In this message, the SGW includes SGW TEID and IP address for the data plane, SGW TEID for the control plane, EPS Bearer Identity, and EPS Bearer QoS along with other parameters which include the PGW addresses and TEIDs for the S5/S8 interface, PDN Type, PDN Address, APN Restriction, APN-AMBR, Protocol Configuration Options, Prohibit Payload Compression, etc.
- Upon receiving the Create Session Response Message from the SGW, the MME processes the message and encapsulates an Attach Accept Message in an S1AP: Initial Context Setup Request Message and sends it to the eNodeB. The Attach Accept message most importantly includes a Unique Temporary Identifier

assigned by the MME to the UE. This identifier is known as the Globally Unique Temporary Identifier (GUTI), while the Initial Context Setup Request Message includes all the parameters that allow the eNodeB to establish the S1-U tunnel towards the SGW and allocate radio resources to the UE. These parameters include but are not limited to SGW TEID and IP address for the data plane, EPS Bearer Identity, EPS Bearer QoS, UE-AMBR, and Handover Restriction list, etc.

- When the S1AP: Initial Context Setup Request Message arrives at the serving eNodeB, the Attach Accept Message is extracted and forwarded to the UE to provide the UE with the GUTI assigned to it by the MME and a list of tracking areas in which the UE can roam freely without performing a Tracking Area Update Procedure.
- The eNodeB establishes and sets up the radio bearers to enable data transfer. This is done by sending an RRC Connection Reconfiguration message and expects RRC Connection Reconfiguration Complete message as a response from the UE.
- Then the eNodeB issues an S1AP: Initial Setup Context Response message and sends it to the MME, where this message includes the eNodeB S1-U TEID and IP address which is required to finish the S1-U GTP tunnel between the eNodeB and SGW for user traffic forwarding.
- Upon reception of S1AP: Initial Context Setup Response message, the MME issues and sends a Modify Bearer Request message to the SGW. This message includes eNodeB S1-U TEID and IP address, EPS Bearer Identity, and Handover Indication. If the Handover indication is set to true in the Modify Bearer Request message, the SGW issues and sends a Modify Bearer Request message to the PGW requesting immediately routing packets for default and dedicated bearers. The PGW acknowledges it by sending Modify Bearer Response back to the SGW.
- At this stage, a Modify Bearer Response is sent back to the MME by the SGW as an acknowledgement to the Modify Bearer Request message, and after this message, downlink packets can be delivered from the external data network to the UE through the established bearer.

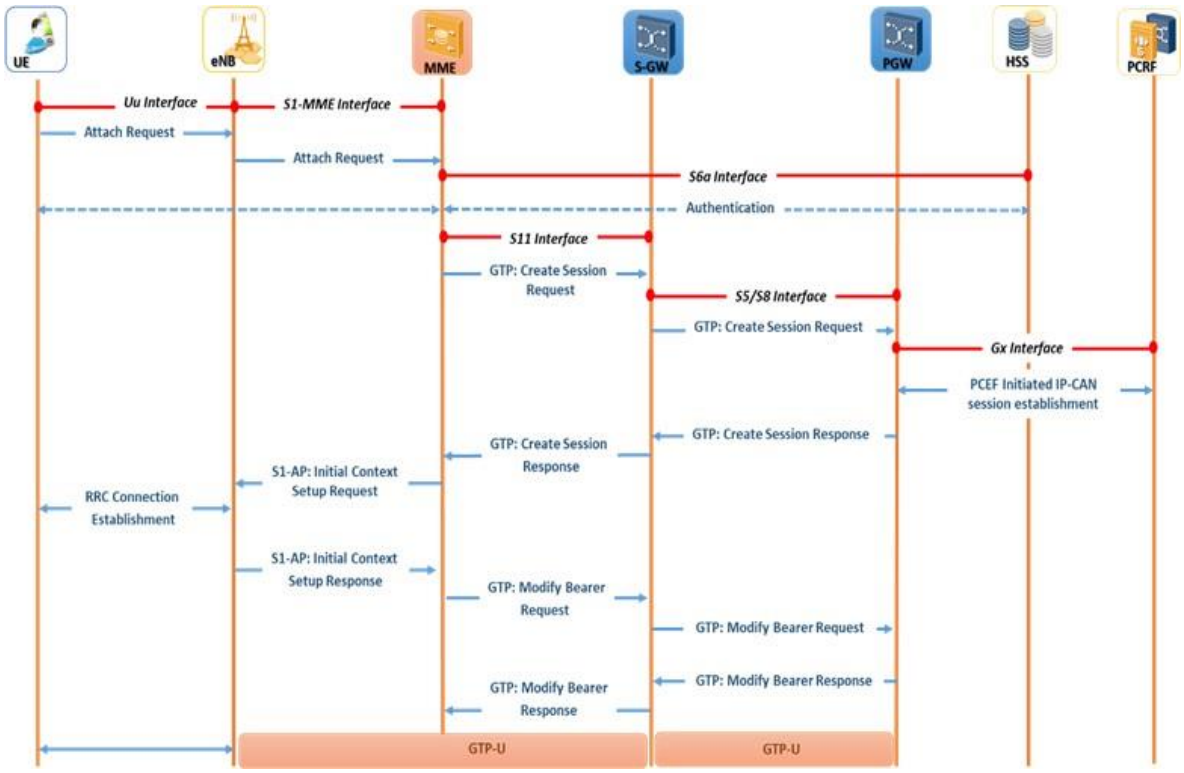


Figure 4.10: Initial Attach Procedure call flow

4.4.2 Access Bearer Release Procedure

A UE can use services offered by the EPS network after establishing the default EPS bearer through the Initial Attachment procedure. In this stage, the UE becomes fully registered to the network and in the state of EMM-REGISTERED, ECM-CONNECTED, and RRC-CONNECTED, which allows the UE to send and receive data traffic to/from the outside world by using the always-on default bearer. After a period of inactivity, the network performs Access Bearer Release procedure to release the UE radio and S1 bearers, which changes its state to ECM-IDLE, and RRC-IDLE. The network may also release the access bearer due to the loss of radio communication between the UE and its serving eNodeB or if the UE has failed the authentication or integrity checks. In this procedure, signalling radio bearers (SRB1, SRB2) between the UE and the serving eNodeB are released and the UE data radio and S1 bearers are deleted. Figure 4.11 shows the message exchanges between the network elements to perform the Access Bearer Release procedure. The process starts when the inactivity timers in the eNodeB have expired.

- This process is started by the eNodeB. It first sends S1AP: UE Context Release Request message to the MME asking to change the state of the UE from ECM-CONNECTED to ECM-IDLE.

- When the request message has reached the MME, the MME composes and sends a Release Access Bearer Request message to the SGW to tear down the S1 bearer that was previously allocated for the UE.
- The SGW releases the UE S1 bearers and responds back to the MME with a Release Access Bearer Response message.
- In this procedure, the S5/S8 bearer remains intact to provide always-on connectivity for the UE, and in this way, the downlink traffic can travel through the network until it reaches the UE SGW and because the S1 bearers are released the paging procedure is triggered to establish the UE radio and S1 bearers.
- When the Release Access Bearer Response message is received by the MME, the latter informs the serving eNodeB through the S1AP: UE Context Release Command to tear down the signaling radio bearers and release the S1 bearers.
- The serving eNodeB reacts to the S1AP: UE Context Release Command by sending an RRC Connection Release message across the air interface.
- The UE responds to RRC Connection Release message by tearing down the SRB1, SRB2, releasing all the data radio bearers, and changing its state to ECM-IDLE and RRC-IDLE.
- At the same time, the serving eNodeB releases the UE S1 and radio bearers and sends an S1AP: UE Context Release Complete message back to the MME.

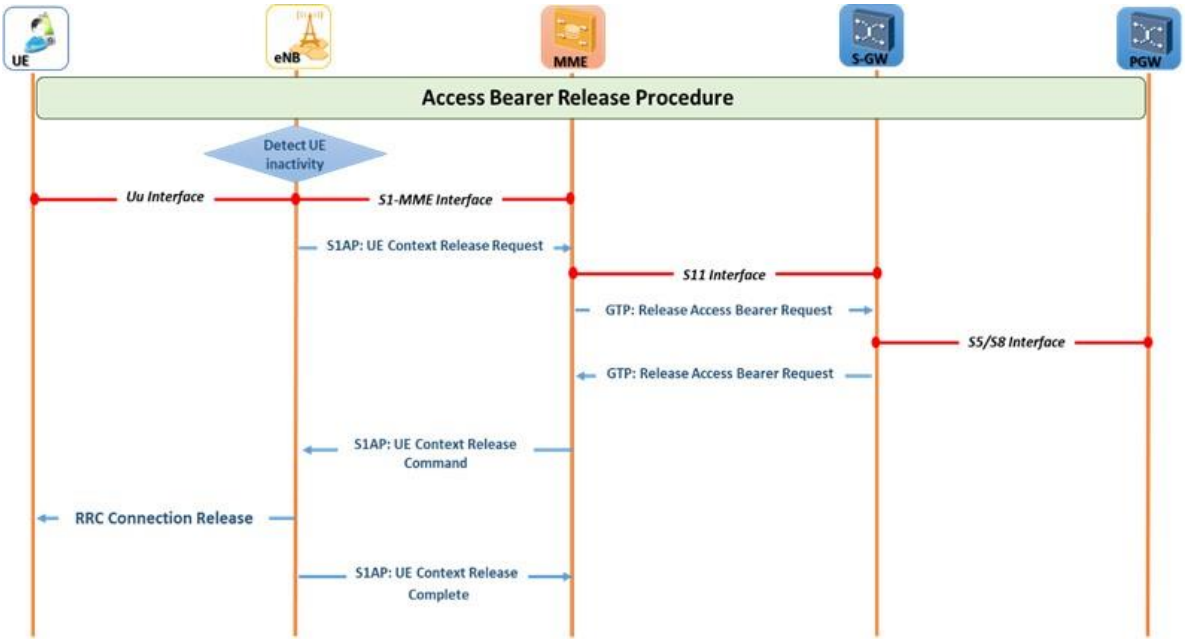


Figure 4.11: Access bearer release Procedure

4.4.3 Service Request Procedure

The Service Request procedure is performed by UEs in ECM-IDLE. This procedure can be triggered by either the network by using the paging produces or by the UE when it wishes to communicate with the network. Network-based Service Request procedure happens when the network receives a packet from an outside server to a UE in the ECM-IDLE state. In this case, the network uses the paging procedure to alter the state to ECM-CONNECTED and re-establish the SRB1 and SRB2. Also, the data radio and S1 bearers are re-established to provide the UE with the capability to exchange data with the outside server. The UE-based Service Request procedure happen when a UE in an ECM-IDLE state wants to communicate with the network. This chapter only describes the UE-based Service Request procedure. Figure 4.12 illustrates the sequence of messages exchanged between the network elements to perform the Service Request procedure.

- The process started by the UE for establishing a signalling connection with its serving eNodeB is done using a Random-Access procedure and the RRC Connection Establishment. Upon finishing the establishment of the signalling connection, the UE composes a Service Request message asking the MME to change its state from ECM-IDLE to ECM-CONNECTED. This message is included inside the RRC Connection Setup Complete message.
- When the serving eNodeB has received the message, it extracts the Service Request message and forwards it to the MME.
- The MME authenticates and updates the NAS security if it required or if the received messages have failed the integrity check
- The MME then sends an S1AP: Initial Context Setup Request message back to the eNodeB asking it to set up the UEs S1 and radio bearers. This message includes information such as the SGW TEID and IP address, mobile radio access capabilities and the security key. This information is maintained by the MME from the UE Initial Attachment procedure.
- Upon receiving the message by the serving eNodeB, the security key is used by the eNodeB to activate AS security. Then it sends an RRC message to the UE to configure the SRB2 and the data radio bearer of the UE.

- The UE considers the RRC message as an acceptance to its state change request and returns an acknowledgment. At this stage, the data radio bearer is active, and the UE can send data to the outside world.
- When the RRC acknowledgment message is received by the eNodeB, the eNodeB sends an S1AP: Initial Context Setup Request ACK message back to the MME. In this message, the eNodeB most importantly includes its S1-U TEID and IP address.
- Then the MME forwards this information to the SGW by sending a Modify Bearer Request message.
- The SGW responds back with a Modify Bearer Response message and now the UE can send and receive data traffic.

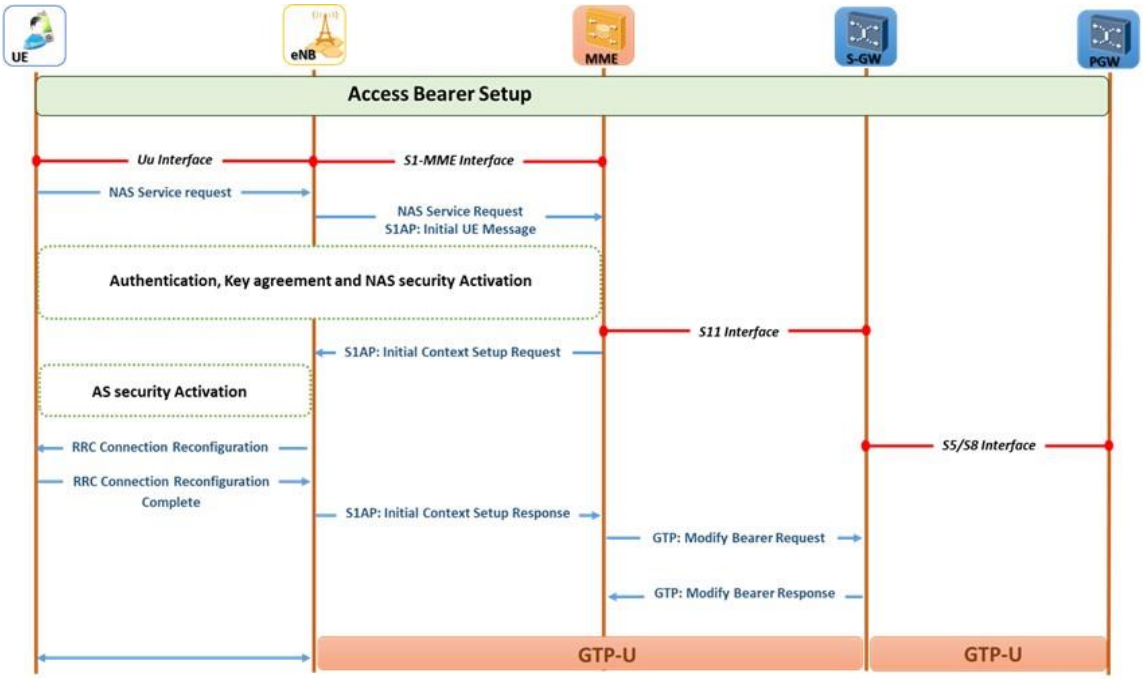


Figure 4.12: Access Bearer Setup Procedure

4.4.4 Handover

Handover is a procedure that describes the process of changing the serving cell of UE in RRC-CONNECTED, ECM-CONNECTED modes. In this process, two eNodeBs are involved, commonly known as the source and target eNodeBs. Multiple variations of UE handover are defined in the 3GPP specification based on the source and target eNodeB. First, Intra SGW and MME handover, which describes the process of handing over a UE from one eNodeB to another without changing the SGW and the MME. Second, inter SGW handover, which describes the UE handover process when the

target eNodeB is connected to a new SGW. In this case, the MME needs to ask the current SGW to release the UE context and ask the new SGW to set up a new set of bearers for the UE. Upon receiving the MME request the new SGW contacts the PGW to alter the S5/S8 tunnel direction to the new SGW. Third, inter MME handover which describes the process when the target eNodeB belongs to a new MME pool area. In this case, S1-based handover is required which starts with the source eNodeB asking the source MME to perform UE handover. Upon receiving the request, the source MME hands control of the mobile over to the target MME, and the later forwards the handover request to the target eNodeB. The focus here is only on the X2-based handover when the SGW and MME are kept the same.

4.4.4.1 X2 Based Handover Procedure

The X2-handover process is started by the source eNodeB right after receiving a measurement report from UE upon which it decides that the UE can receive a better service from a neighbour eNodeB. This chapter only covers the X2 handover that does not involve changing the SGW or the MME. As shown in Figure 4.13, the process is started by the UE performing a measurement procedure.

In this procedure, the UE sends measurement reports for all the neighbouring cells to its serving eNodeB, where the latter has made the handover decision and sends handover request message over the X2 interface to the target eNodeB. By this message, the serving eNodeB is asking the target eNodeB to take control of the UE. This message includes the UE identity, its serving MME, security key, its radio access capabilities, and a list of the UE bearers along with their QoS requirements. It also includes the global ID of the new cell and a list of the UE bearers. Admission control is performed by the target eNodeB by examining the list of the UE bearers to identify the bearers that it is willing to accept. Based on the available resources in the new cell, the target eNodeB may not accept all the UE bearers.

The target eNodeB sends back a Handover Request Ack message along with RRC Connection Reconfiguration to inform the UE about the setups required to communicate with the new cell. This message includes the UE new Cell Radio Network Temporary Identifier (C-RNTI), the SRB1, SRB2 configurations, and list of the data radio bearers that it has accepted. Upon receiving the acknowledgement message, the serving eNodeB extracts the RRC message and forwards it to the UE. At the same time, SN Status Transfer message is sent by the serving eNodeB to the target eNodeB to

convey the PDCP status of the E-RABs. Also, the serving eNodeB forwards any uplink packets that it has received out of sequence, any downlink packets that the mobile has not yet acknowledged and any more downlink packets that arrive from the SGW.

On the other hand, when the RRC Connection message reaches the UE, the latter uses the included parameters to configure itself to communicate with the new cell and perform the Random-Access procedure. Then an RRC Connection Reconfiguration Complete message is sent from the UE to the Target eNodeB. Then the UE starts reading the system information of the new cell and optionally performs the PDCP status report procedure with the Target eNodeB to minimize the amount of duplicate packet re-transmission. At this point, the UE is successfully connected to the Target eNodeB and can send uplink traffic through that eNodeB, but at the same time, the SGW still sends UE downlink traffic to the serving eNodeB. Therefore, the target eNodeB sends an S1AP: Switch Path Request message over the S1-MME interface to the UE's serving MME to change the path of the UE downlink traffic to the target eNodeB. In this message, the Target eNodeB includes all the UE bearers that it accepts to handle and the S1-U tunnel information. This includes the Target eNodeB TEID and IP address. On receiving the message, the MME sends a Modify Bearer Request message over the S11 interface to the SGW. In this message, the MME most importantly includes the list of bearers and the GTP-U tunnel parameters. When the SGW receives the MME request, it changes the UE downlink traffic to the Target eNodeB by redirecting the GTP tunnel based on the MME information for all the bearers specified by the message and deletes the rest. At the same time, the SGW sends an End Marker Packet back to the serving eNodeB to inform it about the end of the data stream.

The serving eNodeB then forwards the same message to the Target eNodeB. The SGW finishes its role by sending a Modify Bearer Response back to the MME as an acknowledgement. In this message, the SGW also includes its S1-U TEID and IP address for the Target eNodeB to use in the uplink direction. When the MME receives the SGW acknowledgment, it sends an S1AP: Switch Path Request ACK message to the Target eNodeB. In this message, the MME includes the SGW tunnel information. Then the Target eNodeB informs the serving eNodeB that the handover has been completed successfully. The serving eNodeB can release now all the UE resources.

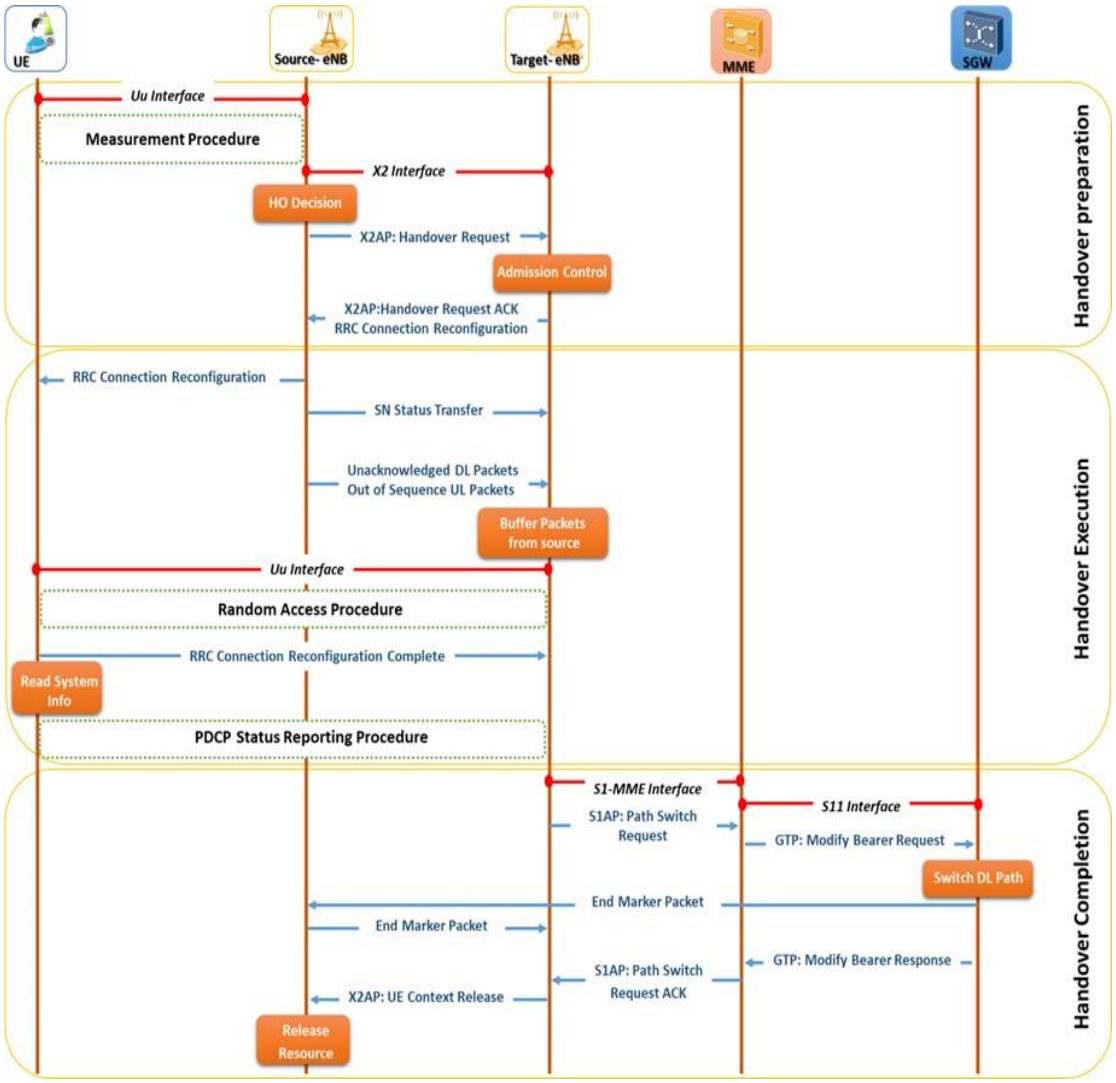


Figure 4.13: X2-Based Handover When the SGW and MME kept the same

4.5 Evolved Packet System Model

Several mobile network simulators exist, which includes commercial simulators and open source simulators. Both types have their share of advantages and disadvantages in terms of flexibility, availability, and cost. Commercial simulators are developed and maintained by specialized companies with a proprietary right, and these simulators are very accurate and reliable and can produce very detailed predictions of the performance of the simulated network. At the same time, they are quite difficult to be modified and enhanced and also, they are very costly and cannot be acquired by everyone. Few of the most popular commercial simulators include OPNET [74] and EstiNet [75] .

On the other hand, open-source simulators are normally developed and maintained by the community, they are free, can be accessed by everyone, easy to be modified and

extended to include new features and functionality. Proprietary simulators are usually not available for free use and can be directed towards very specific research purposes. The decision has been made to use open source simulator because of the lack of availability of commercial simulators and to avoid the unnecessary cost required to purchase a commercial simulator. The most common open source simulator for mobile networks includes NS3 [76], LTE-Sim [77], and simuLTE [78]. After a thorough research, decided has been made to use simuLTE because it has the required tools that are needed to build the simulation platform to simulate SDEPC which will describe in Chapter 6. simuLTE is described in the next sub-section.

4.5.1 SimuLTE Platform

This section briefly describes the simuLTE platform with a particular emphasis on the design of the UE and the eNodeB modules. Basically, simuLTE represents a system-level simulator build on OMNeT++ framework [69]. OMNeT++ is described in detail in Chapter 3 Section 3.4.1. Here will describe OMNeT++ in a way that helps understand the simuLTE platform. OMNeT++ framework revolves around that utilization of a basic modelling units known as a module. Modules can be organized in a hierarchy of compound modules that communicate by the exchange of messages.

Each module consists of structure and behaviour parts. The former is defined via .ned files, while the latter is implemented via C++ classes. This separation increases the flexibility of OMNeT++ by allowing one to change either of the two without affecting the other. LTE Network Interface Card (NIC), represents the core module of the simuLTE platform. This module is used by both the UE and the eNodeB entities to incorporate LTE functionality. The LTE NIC together with modules built by the INET framework [79] are used to create a fully functional UE and eNodeB entities as shown in Figure 4.14. The LTE NIC is modelled in layers that include PDCP, RLC, MAC, and PHY, which directly map to the LTE protocol stack.

The NIC module is built by leveraging one of the most important features offered by the OMNeT++ frameworks, which is the inheritance of both the structure and behaviour of the module. The common features and functionality of each layer of the NIC module are developed in a base class and more specific functionality is added when required. For example, the MacBase class is extended to create the MacUe and MacEnb classes that implement UE and eNodeB specific functionalities receptively.

For instance, eNodeB is responsible for the resource scheduling, which is implemented by the MacEnb class because it is a node-specific function.

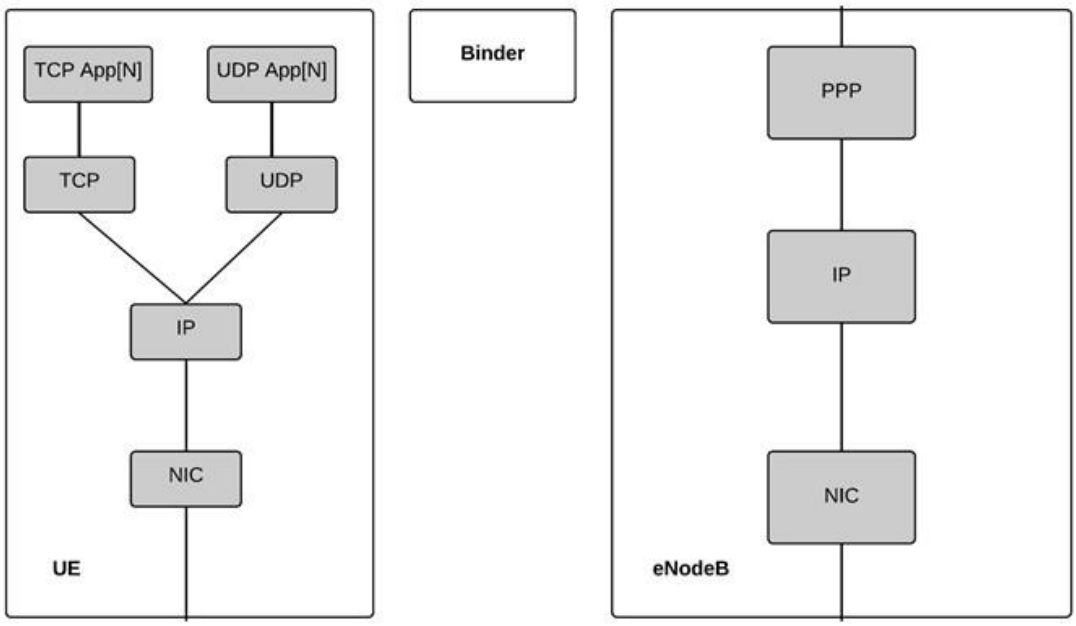


Figure 4.14: UE and eNodeB modules as implemented in simuLTE

In simuLTE, data transmission and resource accounting are modelled separately. A specialized module known as Binder is used for the resource accounting (i.e., keeping track of which RBs are being used by whom). The Binder works like Oracle with full visibility of all nodes in the system and can be queried by them to obtain shared information. This design can produce an optimal baseline by running an ideal algorithm that leverages the binder's full knowledge of the ongoing transmissions. On other hand, data transmissions are modelled through message exchanges between modules. Based on the MAC PDU and the modulation and coding scheme used by the transmitter, the binder specifies the correct amount of RBs to carry the message.

All the control channel specified by the 3GPP specification are not directly modelled in simuLTE. Instead, a direct message exchange is used to accomplish a similar solution. Interference management is guaranteed, considering that the physical layer of the NIC is associated with the ChannleModel. In each node, the ChannelModel is responsible for the computation of SINR of the received signals, which in turn is used by the PHY layer to compute the Channel Quality Indicators (CQIs) and evaluate transmission errors. The ChannelModel is defined in simuLTE as a C++ abstract class with pure virtual functions. The design increases the flexibly of the simulator and allows one to define its own ChannelModel by simply extending the abstract class and redefining the getSINR() and error() functions. In simuLTE a realistic ChannelModel

is included. It accounts for path loss, fading and shadowing. Although simuLTE covers a wide range of LTE features and provides a very good mechanism to gather network statistics, at the same time it misses a few key elements that can be summarized as follows:

- Scalability to build a large network that includes the EPC, in this case it very difficult to scale the network in terms of the number of connected UEs, the reason is that EPC bearers are configured in an XML file which makes it very difficult or impossible to build a network with 100 or 200 UEs
- Very simplified core network that mainly provides a means to specify the UEs default bearer before starting the simulator (using the ”ini” and XML files) without the ability to update/delete/ or even create a new bearer after the simulator has been started, which makes the simulation of some of the LTE procedures like handover not applicable or not accurate considering that the EPS bearer needs to be updated in the SGW entity to redirect the downlink traffic from the source eNodeB to the target eNodeB. Therefore, the decision has been made to build the required functionality to support a scalable, dynamic, and more realistic control plane for the core network of the simuLTE.

To implement the control plane functionality, a new node is modelled, and the existing nodes are extended to support the control plane operations. More specifically, MME is modelled, and the eNodeB, SGW, and PGW are extended to support both the control plane and data plane operation instead of only the data plane. All the EPC nodes are designed to have a simplified and common structure to increase the flexibility of the simulator and make it easy to modify the behaviour of any particular node to test a new feature or protocol. Also, EPC nodes have reused many of the modules that are implemented by the INET library. Therefore, the INET modules used in our simulator are described first, before explaining how OMNeT++ modularity and inheritance are utilized to simplify and optimize the LTE simulator platform.

4.5.1.1 INET Modules used as part of the EPC Nodes

The MME, SGW, PGW, and eNodeB reused the following modules that are defined by the INET library.

- **Interface Table (InterfaceTable)**: Container module that holds network interfaces (eth0, wlan0, etc) of a node. During the initialization phase, network

interface cards (NICs) register the interface to the interfaceTable. Other modules access interface information via a C++ class interface.

- **Routing Table (routingTable)**: Another container is used to store and maintain IPv4 routing entries. Also, the C++ interface is used to access this module. This interface includes many methods to add, delete, and lookup route entry.
- **Network layer**: a Compound module which groups multiple simple modules that represent protocols of the network layer. This includes:
- **IP module**: which is responsible for the Encapsulation, decapsulation, and routing of the received datagrams. This module uses the help of the routingTable module in its forwarding decision.
- **ARP module**: which handles address resolution operation.
- **ICMP module**: which is responsible for sending ICMP echo messages and handles the ICMP reply messages.
- **ErrorHandling module**: receives and logs ICMP error replies [79].
- **Transport layer protocols**: This is a module that performs the transport layer operations. It has a connection to the network layer from one side and the application layer form the other side. Currently, TCP, UDP, and SCTP are supported in the INET 2.3 library used in this work.
- **NICs**: PPPInterface, EthernetInterface, and WLAN interfaces represent examples of the NIC modules implemented in the INET library. Basically, NIC is a Compound module that normally includes queue and MAC modules. The queue is a module used to store the packet while it is waiting to be transmitted through the wire, and multiple queueing types are implemented to accommodate different needs (DropTailQueue, Random Early Detection Queue (REDQueue), DropTailQoSQueue, etc.).
- **Mobility Module**: This module is used by nodes to include movement capability. This module is responsible for the way and the direction of the node movement in the simulated playground. In wireless networks, simulation nodes are required to include this module even if they are stationary because the mobility module stores the location of the node, needed to compute wireless transmissions.

4.5.1.2 Control Plane Functionality and Operation

OMNeT++ modularity, object-oriented and inheritance have been leveraged to optimize and simplify the simulator code considering that the S1AP and the GTP-C are used in multiple EPC entities. More specifically, the GTP-C is used in the MME together with the SGW, and PGW while the S1AP is used in both the eNodeB and the MME.

The gtpControl class structure is shown in Figure 4.15. The gtpControl class implements the main operations and functionality that are shared by all three modules, a more specific class is used to implement the functionality of the MME, SGW, and PGW entities. For example, the gtpControlMme class extends the gtpControl class to add more features and functionality that are related to the MME entity. The MME module built in this simulator does not cover all the functionality specified by the 3GPP standard but at the same time, it allows users to implement and test a wide variety of LTE procedures and operations to measure and evaluate the system performance. This section describes few of the operations handled by the main gtpControl class, and describes the main features added by the gtpControlMme, gtpControlSgw, gtpControlPgw.

There is a shared method in the aforementioned classes, but with different functionality, for example, both the gtpControlSgw and gtpControlPgw have a method to handle Create-Session-Request message, also the gtpControlMme, and the gtpControlSgw has a method to handle Create-Session-Response message, but the functionality of the methods is different based on the class it is defined in. This is done by leveraging C++ function overwrite functionality. The same structure is used in all the control plane modules, therefore, the S1AP module is programmed using the same inheritance feature, and the main functionality that is shared between the eNodeB, and MME is implemented in the S1AP class. The s1apEnb and s1apMme implement the functionality of the eNodeB and the MME respectively. In our Module, the s1ap class implements the handleMessage() method, which checks the message, and if the message is received from the UDP module then, it calls the handleMessageFromUdp(). This method is an abstract method that must be implemented by all sub-classes of the s1ap class (the s1apEnb, s1apMme classes). Each class implements its own version of the handleMessageFromUdp() method. For example, the s1apMme uses this method to handle messages like Initial-UE-Message, Path-Switch-Request message etc.., while

the s1apEnb uses the same method to handle messages like the Initial-Context-Setup-Request, and Path-Switch-Request-Ack message etc. The s1apEnb is modeled as a cListener which simply means that it has the capability to listen to OMNeT++ signals. This feature is used to exchange control messages between the enbApp and the LTE NIC model.

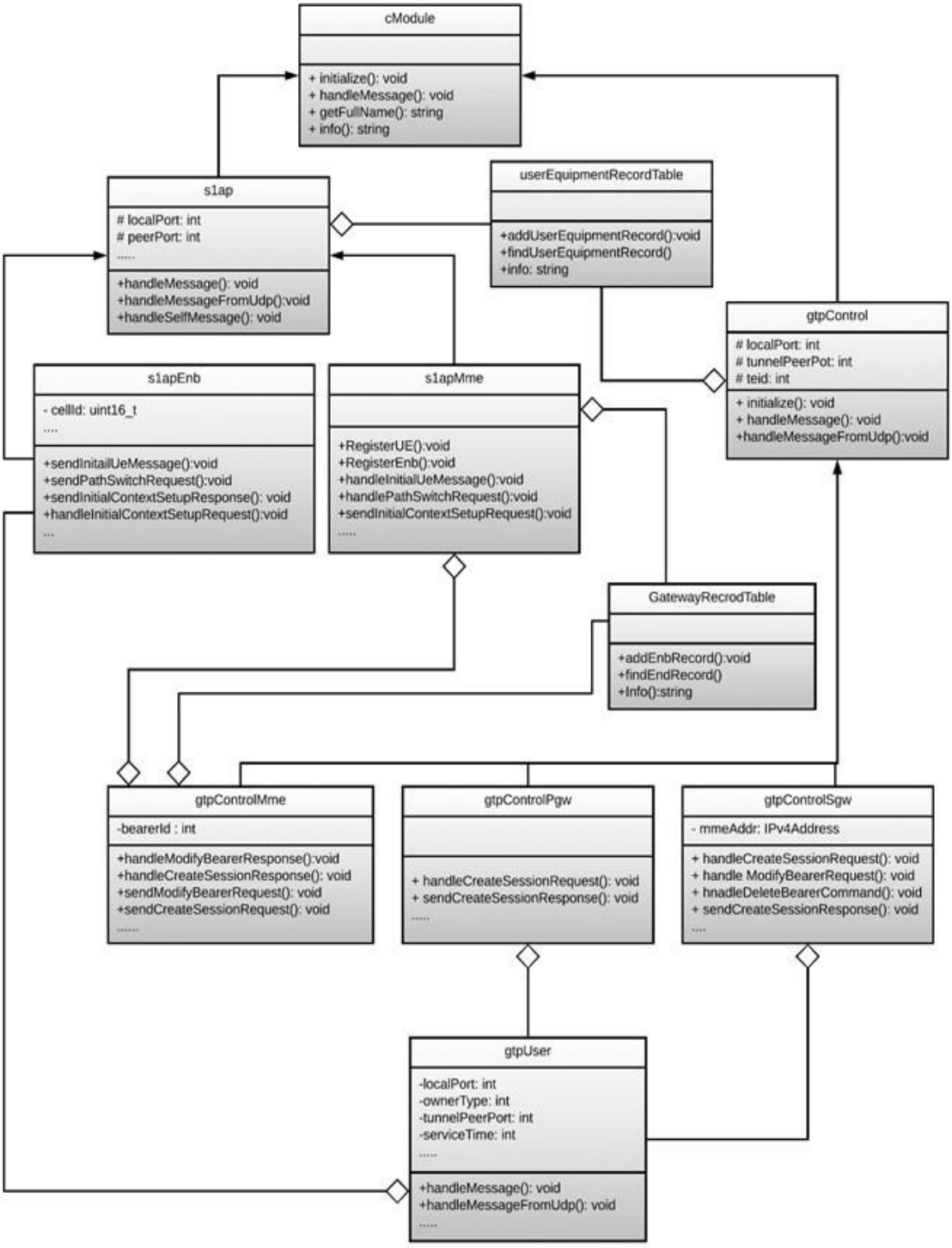


Figure 4.15: UML Class Diagram for the control plane Modules

4.5.1.3 Mobility Management Entity Module

The main structure of the MME is shown in Figure 4.16, where the MME is modelled as an extension to the NodeBase module of the INET library, the NodeBase module represents a simplified node that includes all the basic elements of each networking entity. This includes: i) layer 2 interfaces like the point to point, Ethernet, wireless LAN, and loopback interfaces; ii) interface Table module; iii) network and routing Table modules; iv) it also has modules for mobility, notificationBoard and stats modules.

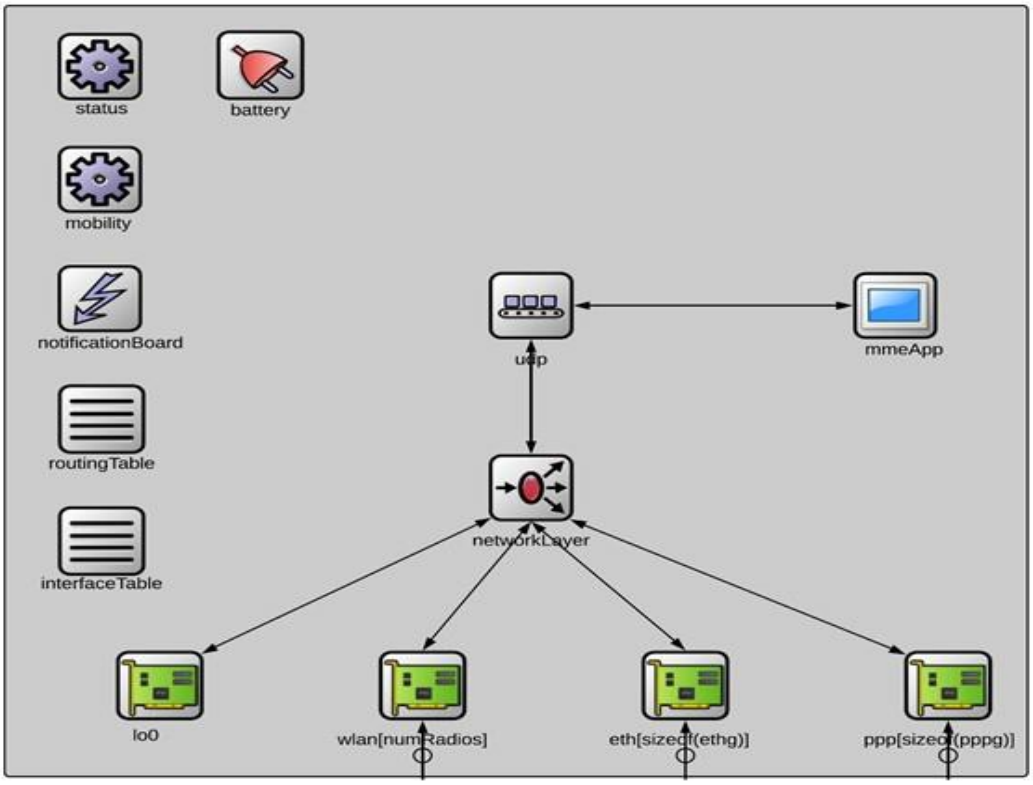


Figure 4.16: OMNeT++ Module of the MME

A. MME Application Module

The main functionality of the MME node is in the MME application (mmeApp module). This module is built as an OMNeT++ compound model that works as a UDP application. For simplicity S1AP interface in our module runs over UDP instead of SCTP as specified by the 3GPP standard. This section describes the structure and the operations of the MME Application module. This includes:

i) identifying the building modules of the MME applications; ii) describes the functionality of each module; iii) shows how these modules are connected with each other; iv) finish with the operations performed by the MME application module.

I. MME Application Structure

Figure 4.17 shows the internal structure of the MME application. It mainly consists of: i) three tables, that represent the subscribers, eNodeBs, and the SGWs that are currently served by this MME node; ii) two control modules that are responsible for handling the S1AP and the GTP-C messages. The control applications use direct call between each other to perform the subscriber procedures, for example when the S1AP module receives the Initial-UE-Message from the eNodeB, it will use one of the GTP-C module methods to send a CreateSession-Request message to the SGW. Considering that in our module the S1AP is also running over UDP, a multiplexer module is used to differentiate between the received messages and sends them to the correct control module for process.

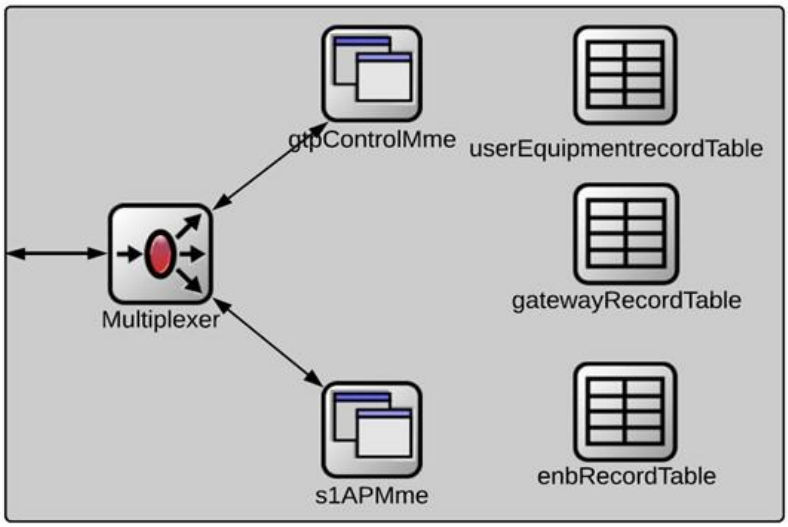


Figure 4.17: OMNeT++ Module of the MME Application

- **eNodeB Record Table:** This table contains a list of all the eNodeBs that are currently served by this MME, the table consists of one or more entries, and each entry contains information that is related to an eNodeB. This table is used by the GTP-C and the S1AP modules to perform their operations correctly and accurately. The s1apEnb at the initialization of the network finds the MME node and registers the eNodeB with the MME. This approach simplified the simulator for anyone because based on the connection setup in the network topology, the s1apEnb finds the MME and registers itself without any static configuration prior to the network run (in the ”ini” file).

- **Serving Gateway Record Table:** The same as the eNodeB record table, this table consists of one or more entries, and each entry represents a SGW information that is required to perform some of the MME application operations such as the SGW selection process. The gtpControlSgw in the SGW node is used by the network to find the MME and register itself with the MME. The SGW record table in the MME is used to keep information about each SGW in the network that has a connection to the MME.
- **Subscriber Record Table:** This table is structured in the same way as the first two tables. It keeps information about the subscribers that are currently served by this MME. The table represents a container for the subscriber's profile. The latter represents a class that is used to maintain UEs' most important information. This includes GTP tunnel information for the S1-U, S11, and S5 interfaces, S1AP-Enb and S1AP-MME identifiers, the current cell-Id and more. This table is also used by the GTP-C and the S1AP modules to perform their operations.

- **GTP Control Module for the MME:** The GTP Control plane functionality in the MME is modeled as an OMNeT++ simple module. The gtpControlMme class is used to implement the control plane functionality in C++. The module handles most of the procedures specified by the 3GPP LTE standard. Currently, the module only supports EPS bearer setup/ update/delete operations. The module functionality can be easily explained because it is structured using the same layout used in OMNeT++. In this module, the received message is handled by the main handleMessage() Method. In this method the message type is checked and based on the result a new method is called to handle the message, for example when a Create-Session-Response Message is received from the SGW, the message type is first checked in the handleMessage() method, then the handleCreateSessionResponse() method is called to handle the message. The handleCreateSessionResponse() method performs the required operation to handle this type of message and initiate the S1AP Initial-Context-Setup-Request message to be sent from the MME to the serving eNodeB.
- **S1AP module for the MME:** This module is used to exchange messages between the MME and the serving eNodeB over the S1-MME interface to perform UE specific procedures as described in Section 4.4. The same as the

GTP-C module, not all the S1AP functionality specified by the 3GPP standard are implemented by our module, but at the same time, it still allows users to implement a very sophisticated network that performs variety of operations, for example in the default bearer setup procedure, after receiving the Attach Request message from the UE, the serving eNodeB adds the message to the S1AP: Initial UE Message and sends it over the S1-MME interface to the MME. In the s1apMme module, the received message is handled by the handleMessage() method. Similar to the work of handleMessage() method in the GTP-C, this method checks the message type and based on the type, it invokes another method that deals with this type of messages. In this case, handleInitailUEMessage() method process the message and asks the GTP-C module to send a Create-SessionRequest message to the SGW. Currently, the s1apMme module implements methods to handle the Initial-UE-Message, Path-Switch-Request, Initial-Context-Setup-Response, and Erab-Release-Initiation messages and it also has several methods to send the Initial-Context-Setup-Request, and the Path-Switch-Request-Ack, etc.

- **Multiplexer:** As shown in Figure 4.17, the multiplexer module has three bi-directional connections, where the first connection represents the mmeApp in and out connections, while the second and the third connections represent the interfaces to/from the GTP-C and S1AP modules, known as to/from GTP, and to/from S1AP respectively. As previously mentioned, both the S1AP and the GTP-C are running over UDP. Therefore, the multiplexer is used to differentiate between the S1AP and the GTP messages received from the UDP module. In this module, the control-info-object attached to the received message is checked to obtain the destination port number and based on that number the multiplexer forwards the received message to either the GTP-C or the S1AP module. For example, if the destination port number is equal to 2123 the multiplexer understands that the received message is a GTP-C message and sends it through the toGTP connection to the GTP-C module.

II. MME Application Operations

The MME application module handles control messages from both the eNodeB and the SGW. Messages from the SGW are GTP-C messages sent over the S11 interface, while the eNodeB messages are S1AP messages sent over the S1-MME interface. As previously mentioned S1AP protocol messages are sent over UDP instead of SCTP.

Therefore, if the received message is not sent by the UDP Module, then the simulation is stopped, and an error message is displayed to help the user understand what the problem is as shown in Figure 4.18. If the message is received from the UDP module then the UDP destination port number is used to differentiate between the s1ap and GTP-C messages. More specifically, the gtpControlMme is responsible for handling messages sent to UDP port 2123, while the s1apMme is responsible for handling messages sent to UDP port 8789 (a random port number is used as the destination port number of all s1ap messages. This can be changed through the ".ned" file of s1ap module or the "ini" file of the simulated network). Messages sent to different UDP port numbers are simply ignored with an error message explaining that MME application does not support this type of messages.

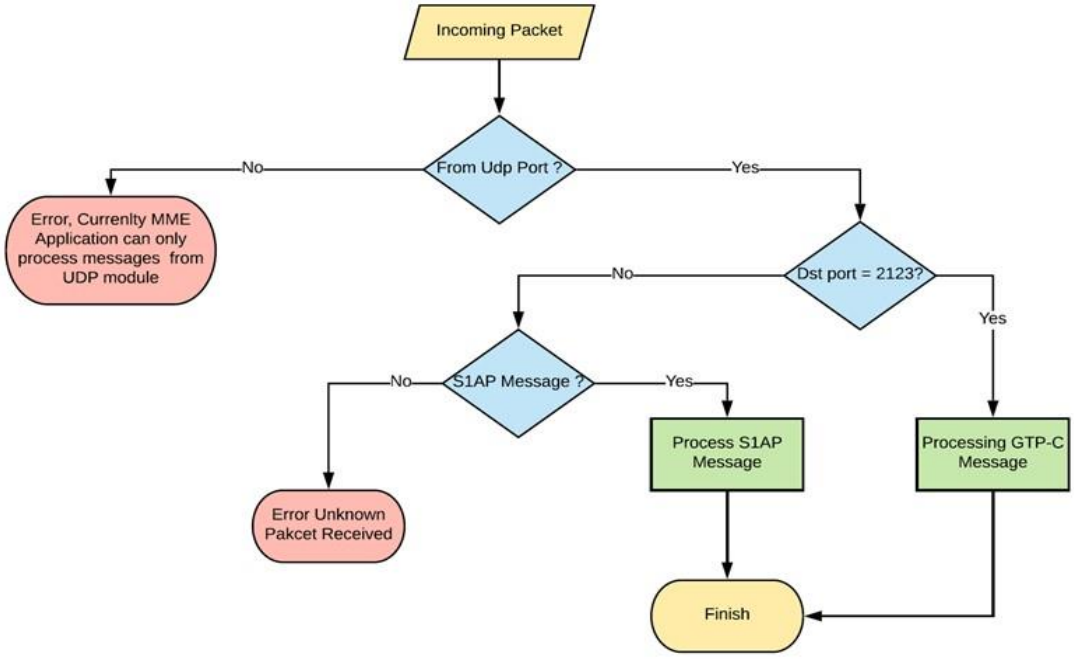


Figure 4.18: MME Application Module Operations

4.5.1.4 Serving Gateway Module

SGW in the LTE simulator is modelled as OMNeT++ compound module that is composed as follows: i) routing and interface tables; ii) mobility module; iii) NICs; iv) Transport and Network layer modules; v) SGW Application module; vi) stats modules. The main structure of the SGW Module is shown in Figure 4.19. All these modules are implemented by the INET frameworks and reused in our LTE simulator, except the SGW Application module which is implemented by us to include LTE specific functionality. The structure and functionality of the INET framework modules are

briefly described in Section 4.5.1.1, and more details can be found in [79]. This section explains the structure and the operation of the SGW application module.

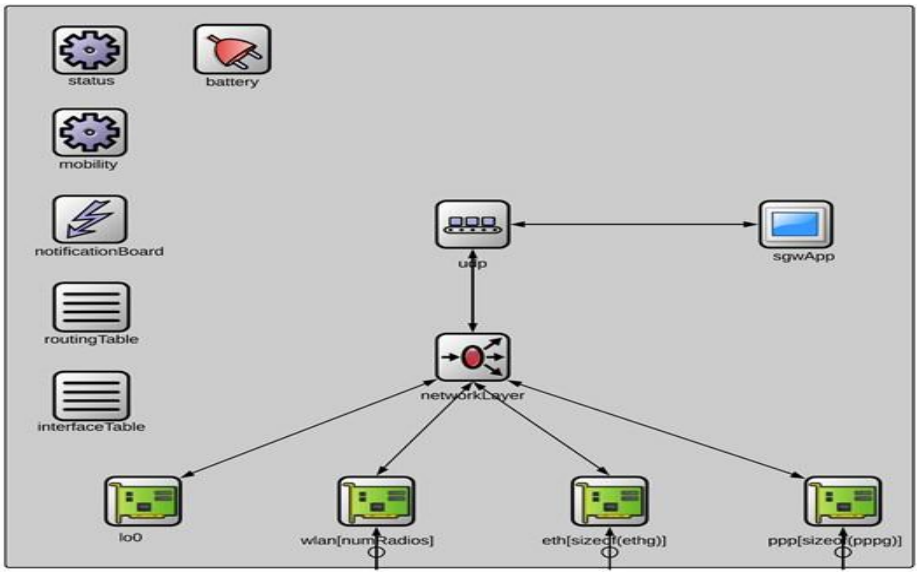


Figure 4.19: OMNeT++ Module of the SGW

A. SGW Application Module

This is an application module that simulates SGW functionality, which includes intelligence and the main operation performed by the SGW Module. The SGW module can only have one SGW application and the application must be connected to a UDP module. The SGW application has several parameters to allow the user to configure the module operations from the ".ini" file.

I. SGW Application Structure

The SGW Application consists of several simple modules that are assembled into a complete SGW application module. These include a Multiplexer, GTP-C, GTP-U, and subscriber Record table as shown in Figure 4.20. The module has a dedicated gate for the UDP layer module. The GTP-C is modelled by the gtpControlSgw module which extends the gtpControl module to add the functionality and operations performed by the SGW control plane, while the functionality and the operation of GTP-U are modelled by the gtpUser module to handle data plane traffic sent over the GTP-U tunnel. Finally, the subscriber record table is used to maintain the UEs context and used by both the gtpUser and gtpControlSgw modules.

The general structure of the sgwApp is shown in Figure 4.20, where the module has a single bi-directional connection with the UDP module. The inner simple modules exchange messages by using connections to send and receive messages except for the

subscriber record table because both the gtpControlSgw and gtpUser modules use direct calls to leverage the functionality offered by the userEquipmentRecordTable module. The multiplexer module works in the same fashion in all modules (MME, SGW, PGW, eNodeB). The main functionality is to differentiate the received traffic and send it through the correct connection to the right module. In the SGW, traffic with UDP port 2152 is sent to the gtpUser module, while traffic with UDP port 2123 is sent to the gtpControlSgw module. UE contexts are stored in the userEquipmentRecordTable module, this module is used as part of the MME, SGW, PGW and eNodeB application modules and offers the same functionality for all modules. The userEquipmentRecordTable module is fully described in Section 4.5.1.3A. I. .

GTP control and data plane messages are handled by the gtpControlSgw and gtpUser modules respectively. gtpControlSgw handles GTP-C messages, which includes updating the UE context state and responses to the received messages if necessary. Figure 4.15 shows a simplified version of the type of messages handled by the gtpControlSgw module. The gtpUser uses UE context information stored in the userEquipmentRecordTable to specify the correct configuration and direction for the received GTP-U messages.

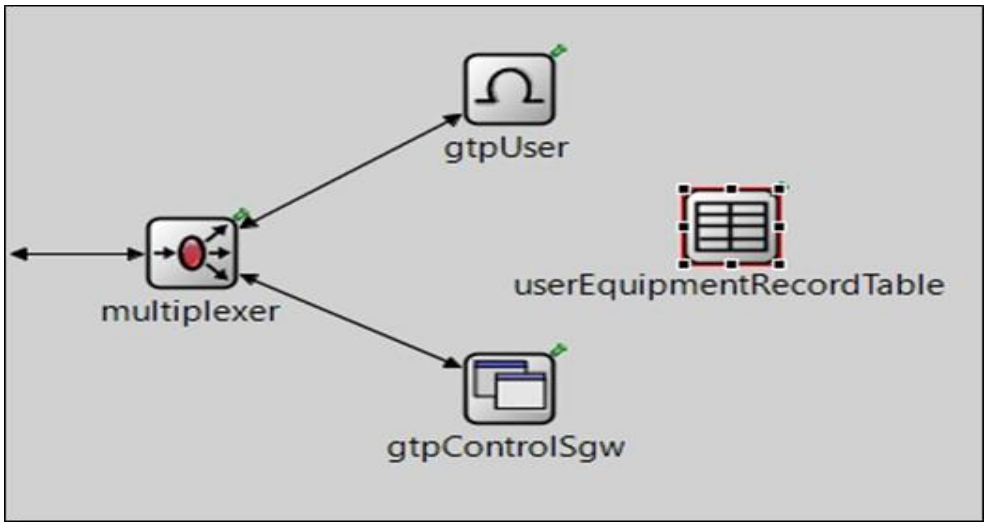


Figure 4.20: OMNeT++ Module of SGW Application

II. SGW Application Operations

The steps performed by the SGW Application to handle the received message are illustrated in Figure 4.21. Upon receiving a message, the application checks the port from which the message is received. The module can only process messages received

from the UDP module, therefore, precautionary steps are included to stop the simulation with error message explaining the problem if a message is not coming from the UDP module. If the message is received from the UPD Module, then the transport layer destination port number is checked. If it equals to 2152, then the application realizes that a GTP-U message is received and sends it to the gtpUser module to handle it. In the gtpUser module, the TEID value obtained from the received message is used to find the GTP-U tunnel information. This information is used to encapsulate the packet and send it to the correct peer node. If the transport layer destination port number is not equal to 2152, then the module knows that the received message is not GTP-U message and checks if it is a GTP-C message by trying to match the destination port number to 2123. If it does not match, then an error message is displayed. Otherwise, if it matches, then the message is sent to the gtpControlSgw module to handle the message. In this module, the message type is examined and based on the result a handler is called.

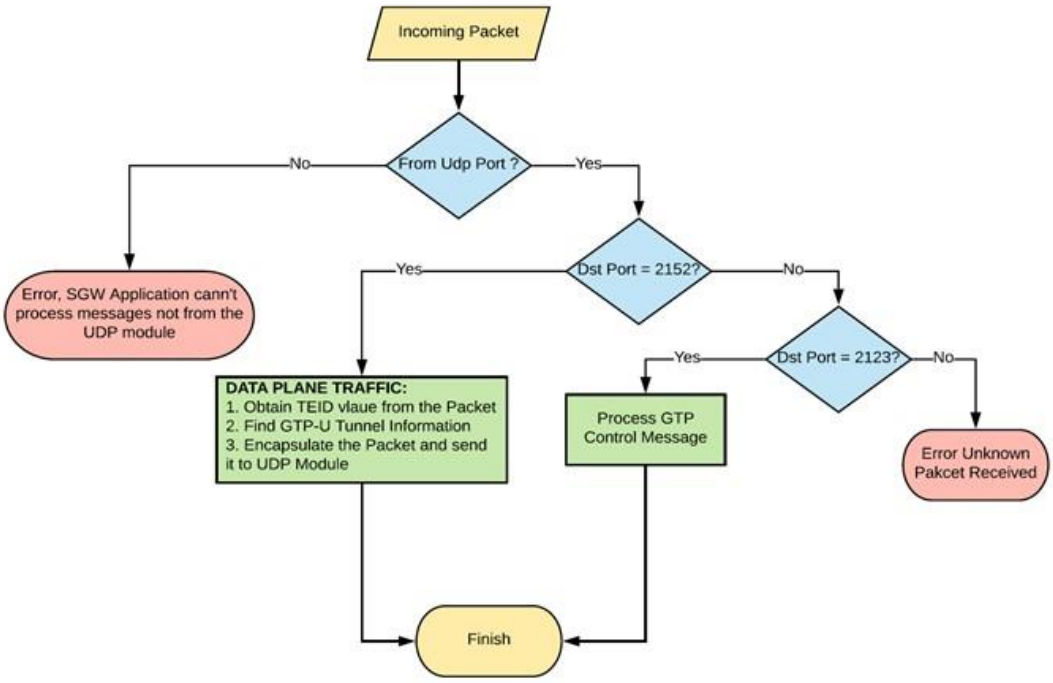


Figure 4.21: SGW Application Module Operations

4.5.1.5 Packet Data Network Gateway Module

This is a compound module that represents the functionality and operations performed by the PGW Entity. The module contains groups of multiple simple modules. The PGW module has the same structure and internal modules as the SGW module with two exceptions. First, the PGW module has another pppInterface to simulate the SGi

interface. Second, the PGW application is used instead of the SGW application. The module has parameters to specify the operation of the module. Changing these parameters from the ".ini" file allows one to modify the module behaviour. The module structure is illustrated in Figure 4.22, the design and features offered by modules implemented by the INET framework are briefly in Section 4.5.1.1. This section focus on the PGW application module to illustrate its design and explain the features offered by the module.

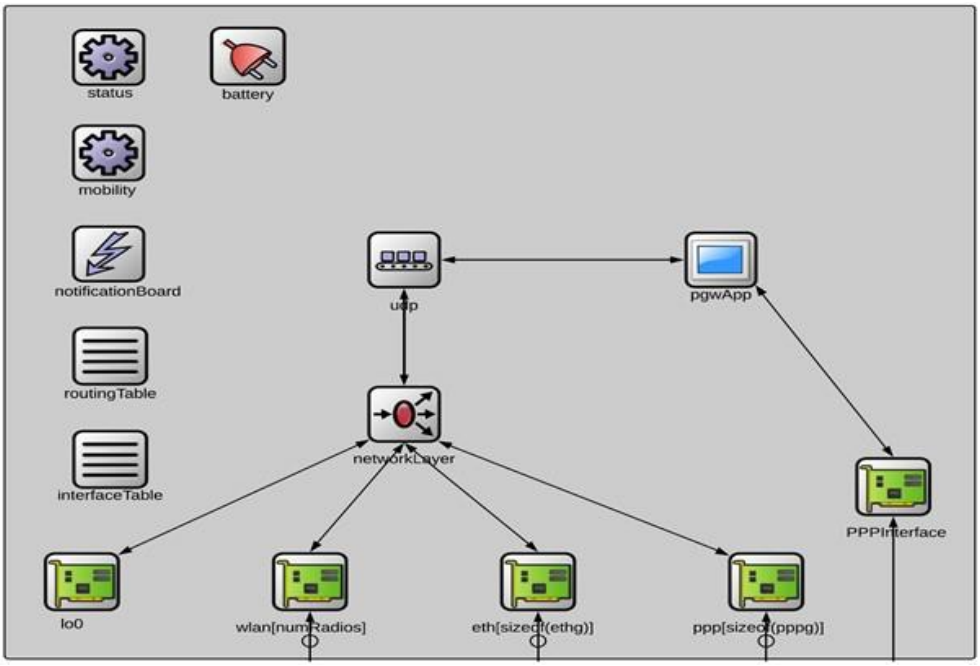


Figure 4.22: OMNeT++ Module of the PGW

A. PGW Application Module

LTE simulator provides a simple design for all of its nodes, where the important operation and functionality is implemented by the node application. In the PGW, the brain and the data plane forwarding decision is made by the PGW application. This includes bearer setup and teardown, data plane traffic filtering. In 3GPP specification, the PGW is responsible for UE IP address allocation during the Initial-Attachment-Procedure as shown in Section 4.4.1. This is not the case in our simulation because in OMNeT++, IPv4NetworkConfigurator module is responsible for assigning IP address for all nodes in the network during the simulation initialization phase. With all that, the functionality of the PGW are still the same.

I. PGW Application Structure

The environment of the PGW application module is described by Figure 4.23. It mainly consists of: i) multiplexer module that works as a decision maker to direct the received traffic to the correct module. The multiplexer uses UDP port number to specify the traffic direction; ii) userEquipmentRecordTable which represent a container to store UE context during the simulation runtime; iii) GTP data and control planes processing module which is modelled by the gtpUser and gtpControlPgw modules respectively; iv) Finally, trafficFilterModule which is used to extract header information of the packets received from the Internet and use it to specify the bearerId for these packets. The same Multiplexer and the userEquipmentRecordTable modules are used in all the nodes (MME, SGW, PGW and an eNodeB). The design and features offered by these modules are fully described in Section 4.5.1.3A. I. Figure 4.15 shows a very basic overview of the methods implemented by the gtpControlPgw class to process and generate GTP-C messages. gtpUser module is responsible for processing the received GTP-U messages by leveraging the UE context configured by the gtpControlPgw and maintained in the userEquipmentRecordTable module. trafficFlowFilter is a simple module that is responsible for classifying the traffic received from the Internet, the module has a list of filters that belong to different bearers. The header information of the received packet, more specifically (source and destination IP address, transport layer protocol and the source and destination port numbers) is used to find the bearerId, then the bearerId is attached to the packet as a part of the control-info-object, then the packet is sent to the gtpUser for processing.

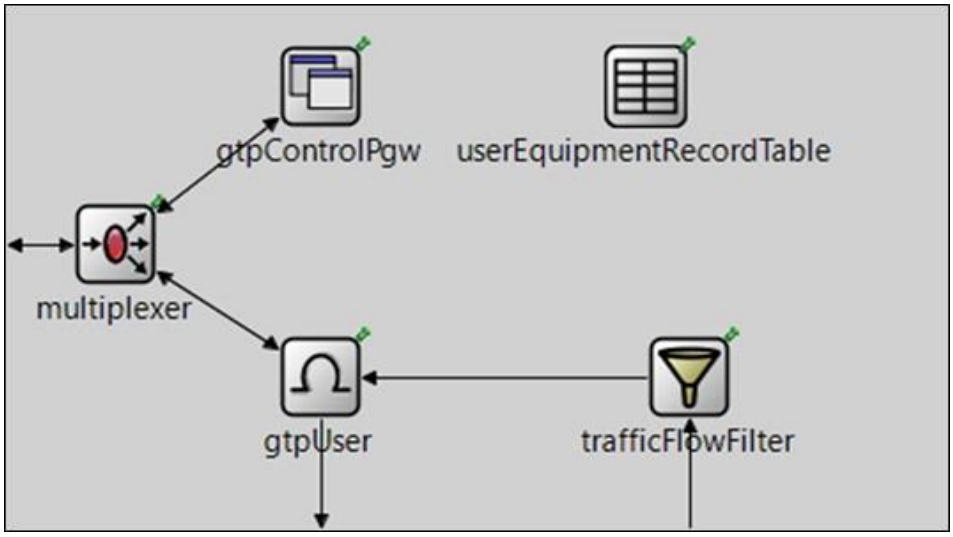


Figure 4.23: OMNeT++ Module of PGW Application

II. PGW Application Operations

The PGW application module handles both data and control planes traffic. As shown in Figure 4.24, upon receiving a message, the PGW application check the port from which it is received. There are two possible gates to receive a message. The first one from the UDP module and the second from the pppInterface through the trafficFlowFilter module. If the message is received from the UDP module, then the PGW application checks if the received message is a GTP-U message or GTP-C message or something else. To do that the PGW application checks the transport layer destination port number of the received message. If it is equal to 2152, then the message is sent to the gtpUser to handle it. In the gtpUser module, the UE packet is extracted from the GTP Tunnel and sent to the Internet through the pppInterface. Otherwise, if the destination port number equal to 2123, then the message is sent to the gtpControlPgw module to handle it. This module handles the received message based on the type attached to the message for example if Create-Session-Request message is received the module calls the handleCreateSessionRequestMessgae() methods to handle the message and replies if necessary. Messages received from the pppInterface are also handled by the gtpUser module. In this case upon receiving the message, header information is obtained from the message and used to classify the packet using the TFT list, then the bearerId is obtained and used to find the GTP-U tunnel information. The GTP-U tunnel information is used to encapsulate the packet and sends it to the correct SGW.

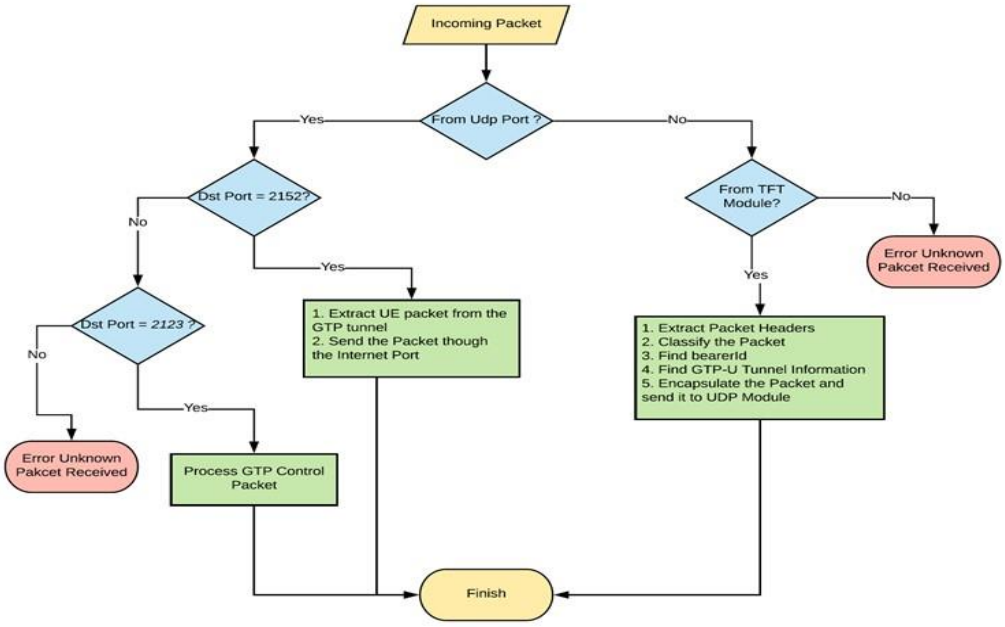


Figure 4.24: PGW Application Module Operations

4.5.1.6 eNodeB Module

eNodeB module implements the 3GPP eNodeB entity for connecting the UE to the EPC network. It is modelled as a compound module that assembles the functionality offered by multiple simple modules. The design of the eNodeB module is different from the other modules that previously described. As shown in Figure 4.25, the module consists of several basic modules that implemented by both the INET framework and the simuLTE simulator. At the same time, the enbApp module is added to support the eNodeB control plane functionality. Also, LTE NIC module is slightly modified to incorporate the control plane operations. The modules implemented by the INET modules includes the UDP, SCTP, NetworkLayer, NICs, routing and interface tables, mobility and statistics. These modules are used as a part of the eNodeB module. The design and functionality of these modules are briefly described in Section 4.5.1.1. While the deployer, LTE NIC (nic), X2App, and gtpUserX2 modules are implemented by the simuLTE. The design and the features offered by these modules are fully described in [78].

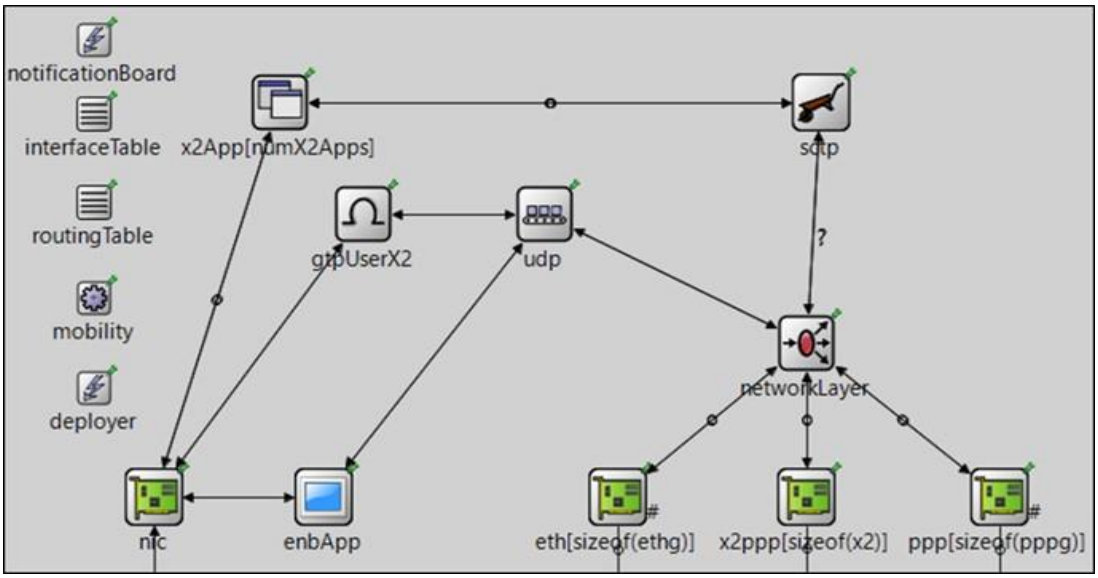


Figure 4.25: OMNeT++ Module of the eNodeB

Brief description of the aforementioned modules is presented in this section to help the reader understand the operations performed by the eNodeB module. The eNodeB module uses the X2App and gtpUserX2 to communicate with its neighbour eNodeBs and exchange control and data planes information respectively [80]. More information about the X2 protocol stack of both control and data planes are shown in Section 4.3.1. Each eNodeB module has a deployer that includes basic information about the eNodeB

to use during the simulation. The LTE NIC module is fully described in Section 4.5.1. Here few of the changes made to the NIC module to include the control plane functionality is highlighted. This includes: i) changes to the IP2LTE layer of the NIC module; ii) adding a new functionality to allow the NIC module to exchange control plane information with the eNodeB application by utilizing OMNeT++ signal mechanism; iii) modification to PDCP layer to expect the LCID with the traffic coming from the eNodeB application module.

A. eNodeB Application Module

eNodeB application module plays a crucial role in the eNodeB module. This includes: i) the generation and the process of s1ap messages; ii) process of the GTP-U messages; iii) NAS messages overlay to the EPC. This section describes the design and the operation of the module.

I. eNodeB Application Module Structure

The simple modules used to assemble the functionality of the eNodeB application compound module are illustrated in Figure 4.26. This includes the multiplexer, gtpUser, s1apEnb, trafficFlowFilter and the userEquipmentRecordTable modules. Figure 4.26 also shows how these modules are connected to each other. The multiplexer is connected to the eNodeB application from/to UDP gate from one side and the s1apEnb and gtpUser module from the other side. trafficFlowFilter is connected to the eNodeB application from one side and the gtpUser module from the other side. userEquipmentRecordTable does not have a connection with any other module and can be accessed only by a direct call.

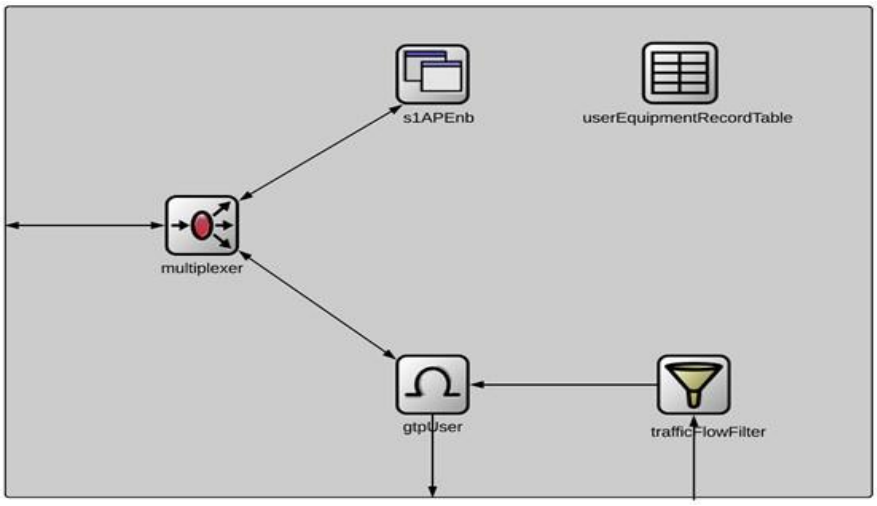


Figure 4.26: OMNeT++ Module of eNodeB Application

II. eNodeB Application Module Operations

The eNodeB application has two gates from which it can receive packets. First, from the UDP module. Second, from the LTE NIC module. As shown in Figure 4.27, when a packet arrives at the eNodeB application, the port from which the packet is received is obtained and examined. Packets that are not coming from the UDP module goes through another process to check if it is coming from the LTE NIC or not. If it not coming from the LTE NIC, then the simulation is stopped with an error message explaining the issue. Otherwise, if the packet is coming from the LTE NIC module, then the eNodeB application understands it needs to handle uplink data plane packet because in the our eNodeB module, the NIC exchanges control plane messages with the eNodeB application module by leveraging OMNeT++ signal module. The LCID value is attached to all the data plane uplink traffic received by the eNodeB application. Therefore, the LCID is extracted from the received packet and used to find bearerId for this traffic, then the GTP-U tunnel information of the bearer is used to encapsulate the packet and sends it to the UE's SGW.

In case the received packet has arrived through the UDP gate, then the UDP destination port number is used to differentiate between the control and data plane traffic. If it is equal to 2152, then eNodeB knows that it is dealing with downlink data traffic. Therefore, it uses the TEID of the received packet to find the LCID, then extracts the inner packet from the GTP tunnel and attaches the LCID to the inner packet before sending it to the LTE NIC Module. If the UDP destination port equal to 8789, then the eNodeB handles s1ap message. Otherwise, an error message is displayed to explain that the module received is an unknown message that cannot be handled.

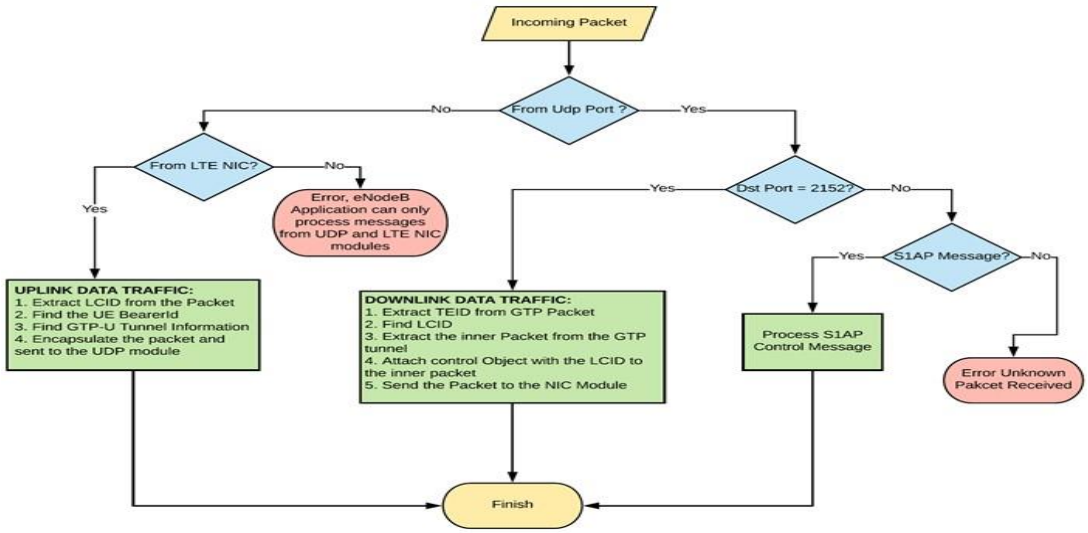


Figure 4.27: eNodeB Application Module Operations

4.5.2 Data Plane Traffic Forwarding in LTE Simulation Model

This section describes how UE traffics are forwarded by the LTE simulation model in both uplink and downlink directions.

4.5.2.1 Downlink Traffic

Downlink traffic procedure is shown in Figure 4.28. The Internet traffic is first received by the PGW entity through the PPP interface, which sends the received packet to the pgwApp without any modification. In the pgwApp, the packet is first handled by the trafficFlowFilter module, where the packet header is extracted from the packet and the 5tuple (source and destination IP addresses, source destination port number, and the transport layer protocol) is used to find the bearer-Id of the bearer to which this packet belongs. The bearer-Id is then added to control-info-object and attached to the packet before it is sent to the gtpUser module. In the gtpUser module, the bearer-Id is obtained from the control-info-object and used to lookup the S5-U GTP tunnel information. The gtpUser module has a UDP socket, used to add the new UDP, IP, Layer2 headers to the GTP packet. The gtpUser module encapsulates the received packet with the GTP header and leverages the UDP socket to add the other layers. The GTP packet is sent through the multiplexer to the UDP module, which adds the new UDP headers to the Packet and sends it to the network module, and the latter adds the layer 3 header information and sends the packet to the layer 2 interface to send the packet to the other end of the tunnel (SGW).

In the SGW, the Layer2, IP, and transport headers of received packet are removed by passing the packet through the layer2 interface, network and the UDP modules. Then the GTP message is delivered the sgwApp module. The GTP message that is sent by the UDP module to the sgwApp has a control-info-object attached to it. In the sgwApp the multiplexer module extracts the object from the message and checks the UDP destination port number and based on that port number, the message is forwarded to either the gtpUser module or gtpControlSgw module. Assuming this packet is a data plane packet (UDP destination port 2152), the packet is sent to the gtpUser module, wherein this module the packet is interrogated, the TEID is obtained from the packet and used to find the S1-U GTP tunnel information. The obtained information is used to update the GTP layer header information of the packet and the UDP socket is used to deliver the packet to the eNodeB. To accomplish this task, the GTP message is sent to the multiplexer, where the latter sends the packet to the UDP module to add a new

transport header, which sends the message to the network module to add a new IP header to the received message which sends it to the Layer 2 interface to add new layer 2 header to the packet and delivers the frame to the eNodeB.

In the eNodeB, the received packet is decapsulated by passing through the Layer 2 interface, network, and UDP modules, then the GTP message is sent to the enbApp; in this application the packet is received by the gtpUser module, which extracts the TEID from the received GTP message and uses it to find the radio bearer and because the radio bearer is not implemented in simuLTE (LTE NIC module) the TEID is map to LCID, the latter uniquely identifies the radio connection between the UE and the eNodeB, then the GTP message is decapsulated and the inner packet is obtained and sent to the LTE NIC module to be sent to the UE through the air interface. In the UE, the received message is decapsulated by passing through the LTE NIC, network and transport modules and the application data are delivered to the application module.

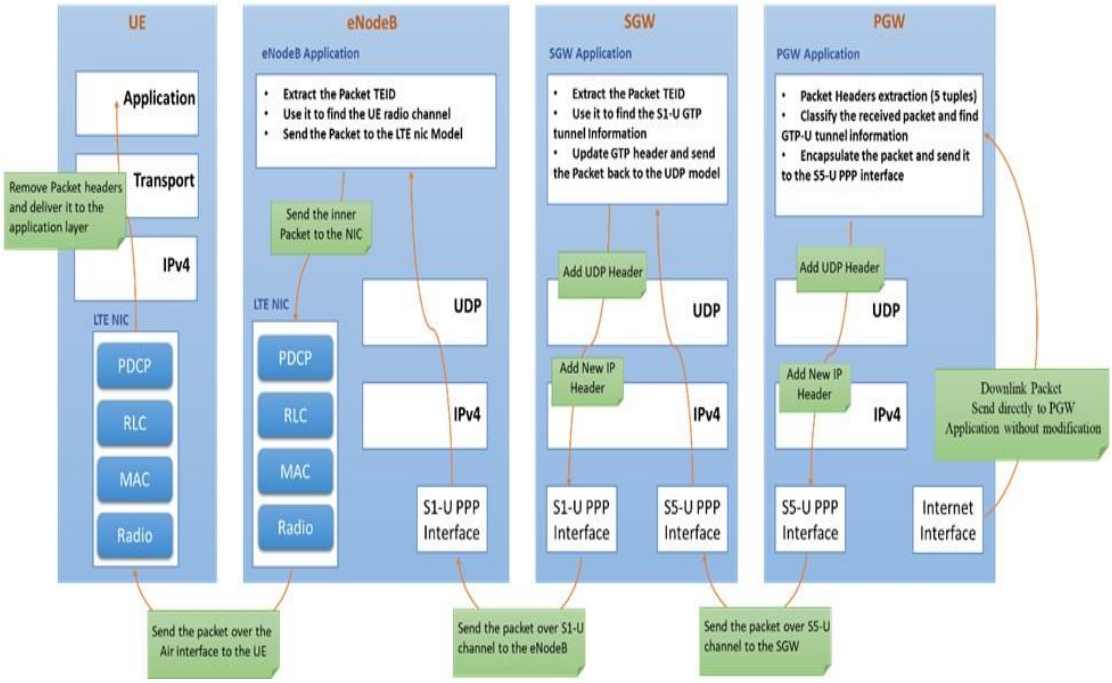


Figure 4.28: Downlink Data processing by LTE simulation Platform

4.5.2.2 Uplink Traffic

Uplink traffic describes the route of the traffic sent by the UE to the Internet or any other services provided by the network operator. In our module, the UE Entity handles both uplink and downlink traffic in the same way. As shown in Figure 4.29, uplink traffic generated by the UE application module is sent through the air interface to the serving eNodeB. This task involves sending the traffic from the application module

through the transport and network modules to the LTE NIC to be transmitted to the eNodeB.

In the eNodeB, the message received by the LTE NIC module and is sent to the enbApp module. The message sent from the LTE NIC to the enbApp has control-info-object attached to it. This object has the traffic LCID. The TrafficFlowFilter module of the enbApp receives the message and extracts the LICD from the attached object of the message and uses it to find the EPS bearer-Id. The TrafficFlowFilter module is used in both the pgwApp and enbApp but the functionality of each model is different. The module has a member variable that specifies where this module is being used and based on the value of this member the functionality or the way this module handles the received packet is specified. The message is sent from the TrafficFlowFilter module to the gtpUser module with the bearer-Id as a part of the control-info-object attached to the message. In the gtpUser module, the bearer-Id is used to obtain the S1-U tunnel information. At this point, the gtpUser encapsulates the UE packet inside a GTP message and sends it to the SGW. The GTP header information includes the UE SGW TEID. To send the message to the SGW, the gtpUser sends the packet to the multiplexer that forwards the message to the UDP module to add a new UDP header to the message and sends it to the network module to do the same and adds a new network header which includes the eNodeB S1-U address as the source and the SGW S1-U address and the destination IP address. Then the packet is sent from the network module to the layer 2 PPP interface to add a new layer 2 headers and is then sent to the SGW.

In the SGW, the message is received by the layer 2 PPP interface and passed through the network and UDP module till it reaches the sgwApp. The message is decapsulated by each module it passes through, more specifically layer 2 headers are removed and the packet is sent to the network module that removes the network header and the segment is then sent to the UDP module, which removes the transport header and sends the message to the sgwApp. The information of the removed headers is attached to the message as a part of the control-info-object. The sgwApp works in the same fashion for both uplink and downlink traffic. In the module, the TEID of the GTP message is obtained and used to find the S5-U tunnel information, then the GTP header is updated with the new information, and the message is sent to the PGW, passing through the UDP, network, and layer 2 modules, When the message passes through these modules a new header is added in each module to route the packet correctly to the PGW specified by the sgwApp.

In the PGW the message is decapsulated passing through the layer 2, network, and UDP modules. Then the GTP message is delivered to the pgwApp. In this app, the GTP header is removed and the inner packet is obtained and delivered to the correct destination, in this case, the Internet.

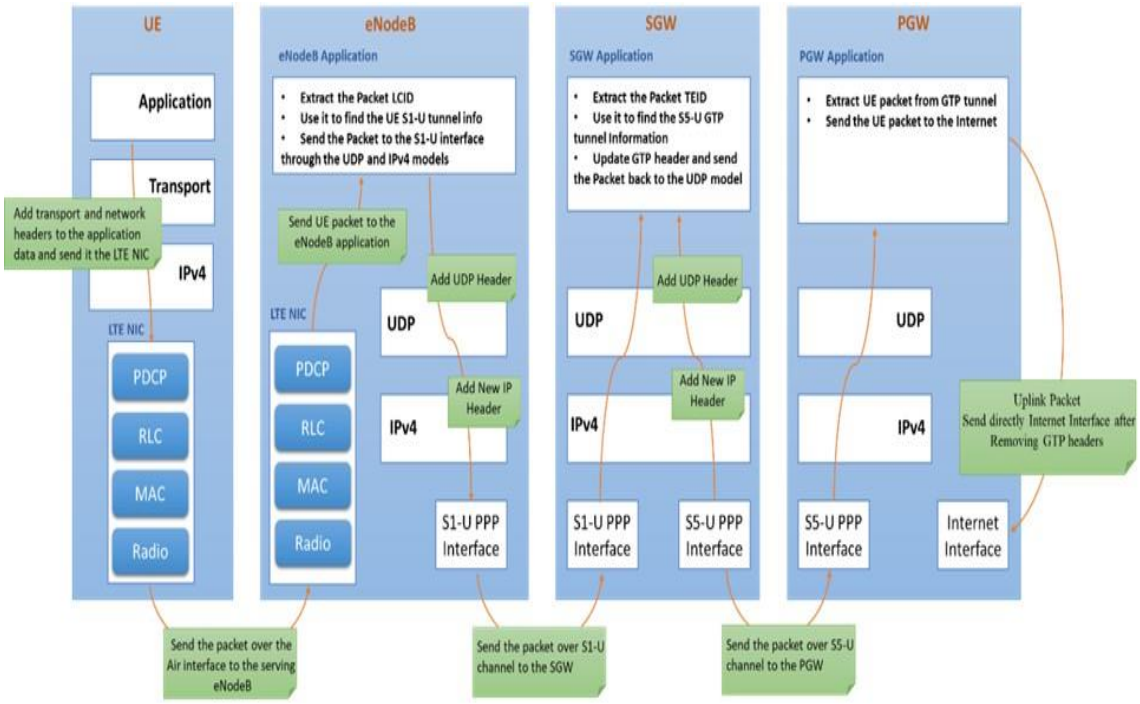


Figure 4.29: Uplink Data Processing by LTE simulation platform

4.6 Summary

The EPS system is an all IP network that consists of two sub-networks. Namely, EPC and EUTRAN. Several Interfaces are defined to allow the communication between the network entities. The S1 interface is used for the communication between the two sub-networks. The concept of EPS bearer is used for the data plane traffic forwarding. GTP represents the core of the bearer design where several procedures are specified by the standard to setup/modify/teardown the EPS bearers and provide always-on connectivity. simuLTE is the simulation platform for the EPS network in OMNeT++, it only simulates data plane operations by using static xml configuration file, which make it inflexible and doesn't support control plane procedures. This chapter showed the design and the operation of the EPS entities emphasizes on the newly added functionalities.

CHAPTE 5

5 Software Based Mobile Core Network Architecture

5.1 Introduction

In recent years, the world has witnessed a massive growth in data traffic of mobile networks, due to increasing numbers of connected devices, which requires a redesign of the network's data and control plane infrastructures to cope with the increasing demands. Researchers in both academia and industry have presented several proposals to adopt SDN in the core of mobile network. For instance, the work presented by [37][81][18] introduce the concept of extending OpenFlow protocol to support GTP tunnel operation, including traffic matching and en- and de-capsulation operations to adopt and benefit from SDN characteristics in mobile networks.

An alternative method of realizing the SDN solution is described in [7][9], which is based on the concept of the elimination of GTP tunnelling and employs the global view of the controller to handle users' operations. However, these studies are limited in that they present straightforward realizations of SDN/OpenFlow but lack a detailed analysis of the necessary procedures. A different approach from the aforementioned studies, which shifts focus to presenting the virtualized cellular network utilizing NFV to provide high availability, elasticity, and modularity, which relatively reduces the capital expenditure, and operation expense, is subsequently offered by [17][19]. This approach also leverages SDN to manage the virtualized network. This chapter makes the following contributions:

- Present mobile core network architecture based on the SDN concept called Software based Mobile Core Network Architecture (SBMCNA). The

architecture's inherent SDN characteristic to provide an abstraction layer separating the control plane from the underline data plane.

- ❖ Identifies few issues with the architecture proposed by the authors of [4] and shows how our architecture can correct the implementation issues.
- ❖ Presents an OMNeT++ module to simulate the SBMCNA architecture. The simulation platform is based on the integrations of OMNeT++ LTE model (simuLTE [78]) and the extended version of our own works of OpenFlow 1.3 [82].
- ❖ Compares two different design methods to support GTP in OpenFlow 1.3 FD.

The chapter is organized as follows: Section 5.2 presents the SBMCN Architecture and explains the functionality of its components. Section 5.3 describes the main operations realized by the SBMCNA to provide on demand service connectivity. Section 5.4 describes the implementation of the SBMCNA architecture and the developed solution in the OMNeT++. Section 5.5 shows the results obtained from the simulation and compares the system performance of two distinct design methods. Finally, the chapter ends up with a summary.

5.2 SBMCN Architecture

Figure 5.1 illustrates the main components of the SBMCN Architecture, which is composed of the elements outlined in the below sub-sections.

5.2.1 Network Controller

The Network Controller (NC) plays a crucial role in SBMCN architecture. It consists of three key components, specifically an application layer, a network operating system, and communication interfaces. The application layer composes sets of applications that implement network functions and services such as MME and SGW/PGW control planes (SPGW-C) applications that will be described later in next sub-sections.

The network operating system consists of several building blocks that are responsible for topology auto-discovery, topological resource view and network resource monitoring. The communication interfaces are the northbound, southbound, and horizontal interfaces (east-west). The northbound interface is used for communication between the application layer and the network operating system such as providing a virtual view of the network to the application layer; it also receives policies for the

application layer. The southbound interface handles the communication between the data and control planes. The horizontal interface allows communication between multiple controllers and supports third party interactions. The controller manages the data plane forwarding of eNodeB, SGW (denoted as SGW-D) and PGW (denoted as PGW-D). The NC is responsible for user session establishment and load monitoring at the data plane.

5.2.2 MME

The MME keeps the same functionality specified by 3GPP with one exception, which is the SGW and PGW selection, because this task is handled by the NC. The MME communicates with the NC using the Application Programming Interface (API). The 3GPP interface between the MME and HSS is maintained [4].

5.2.3 Serving and PDN Gateways Control Plane

The SGW-C and PGW-C represent the brain and control intelligence of the SGW and PGW respectively. These functions, together with the MME, are virtualized and packaged as applications on top of the NC, as shown in Figure 5.1. They play a major role in GTP tunnel establishment including TEID allocation. They allocate unique TEID values for the S1-U and S5-U interfaces for both uplink and downlink traffic. PGW-C is also responsible for UE IP address allocation [4].

5.2.4 Forwarding Device

The FD represents a set of OpenFlow switches enhanced with important features to optimize them for use in a mobile network. These features include GTP encapsulation/decapsulation, and a local program. Therefore, it can apply rules received from the controller in two distinctive ways that will be explained in Section 5.4.2.

5.2.5 OpenFlow-Based eNodeB

This element represents the point of interaction between the access and core network in the SBMCN architecture. It still keeps the same radio functions as specified in 3GPP standards while its S1 interface is replaced by an OpenFlow switch. Therefore, it is necessary to define a new set of control signalling to be sent over the OpenFlow link between eNodeB and the NC to handle UE operations, while the data plane is

programmed according to instructions received from the NC. Each instruction has a hard time-out that represents the flow-entry lifetime. The eNodeB maintains the radio and S1 bearers for as long as the session is active, or the hard time-out has not expired. Upon detecting UE inactivity, the radio bearer is released and the S1 bearer is maintained (i.e. the OF entry is maintained) as long as the Release Timer has not expired. The Release Timer is configured according to the traffic pattern (e.g. session type, session duration, periodic connection request, etc.).

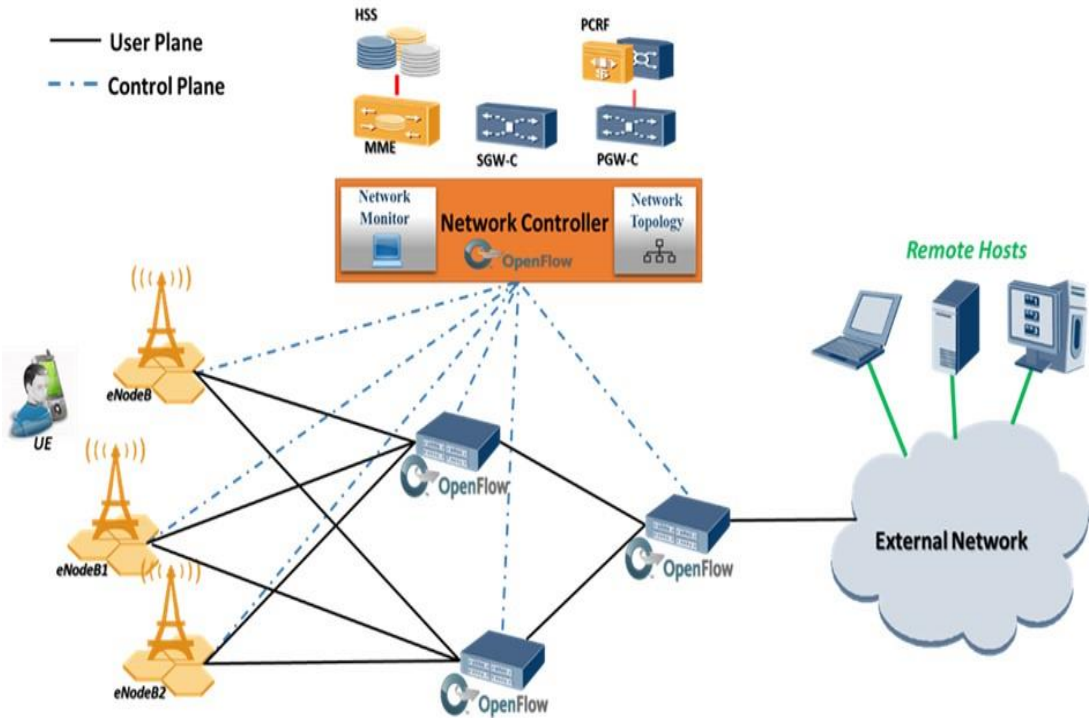


Figure 5.1: Software-Based Mobile Core Network Architecture

5.3 Architecture Operations

This section demonstrates and describes four procedures. Basically, these procedures include the initial attachment, service request, resiliency and load balancing although the same concept is applicable to the other scenarios proposed in [15][16].

5.3.1 Initial Attachment and Initial Access Bearer Setup Procedures

This procedure is started by the UE during the initial power on. It represents the steps used by the UE to register with the network and setup the default bearer. The 3GPP specification of the UE initial attachment and initial access bearer setup procedure is described in Chapter 4 Section 4.4.1 and Section 4.4.3. The authors of [4] offer a

different approach regarding this procedure. Figure 5.2 shows the UE initial attachment procedure.

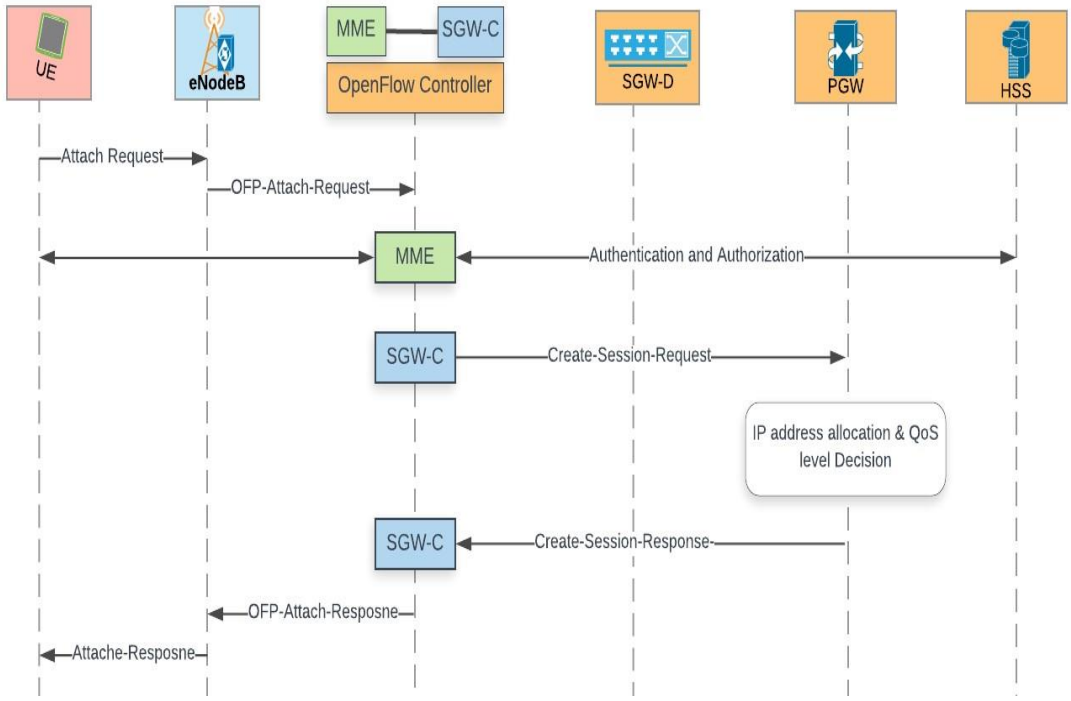


Figure 5.2: Initial Attachment Procedure as proposed by [4]

The authors proposed to use the initial attachment procedure to only perform the UE registration with the network and at the same time assign an IP address to it. In this approach the UE starts the procedure by sending an Attach-Request message to its serving eNodeB, which uses it OpenFlow connection to forwarding the Attach-Request message encapsulated in a Packet-In message to the OpenFlow controller. Upon receiving the message, the controller forwards the request to the MME application. The latter starts the authentication and authorization procedure and upon finishing the procedure, the controller notifies the SGW-C application. The SGW-C sends create-session-request message to the selected PGW with the SGW-TEID. The PGW creates an entry in its tables for the UE. At the same time, it allocates a new IP address for the UE and assigns a S5-TEID for the uplink traffic, then it responds back to the SGW-C application by sending Create-Session-Response message. After the processing of the PGW message the UE IP address is passed from the SGW-C to the MME. The latter sends Attach-Response message that includes the UE IP address to the UE through its serving eNodeB. Then the UE needs to initiate service request procedure to setup the air bearer.

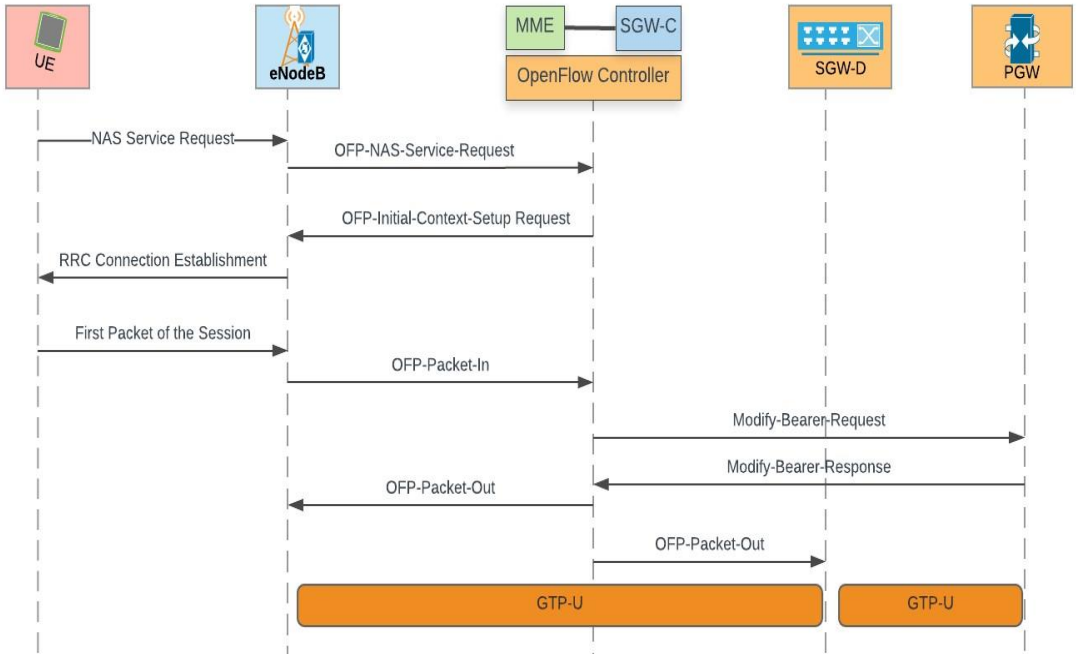


Figure 5.3: Initial Bearer Setup Procedure proposed by [4]

Figure 5.3 shows the service request setup operation. After registering with the network and obtaining the IP address, the UE starts the procedure by sending a service request message, which is passed by the eNodeB to the network controller. The eNodeB encapsulates the NAS service request message in a Packet-In message and send it through the OpenFlow connection to the network controller. Upon receiving the service request message, the NC controller asks the eNodeB to setup the radio bearer for the UE. Upon receiving the first packet of the session, the NC controller initiates the bearer setup procedure to configure the data plane FDs. The controller sets up the GTP tunnel by sending Modify bearer-request message to the PGW and Packet-Out message to the eNodeB and the selected SGW-D. The action associated with the Packet-Out message is sent to the SGW-D, which includes information such as the SGW-TEIDs of the S1 and S5 Interfaces along with the IP address and the TEID of the PGW and the eNodeB. This proposal is conceptually correct, but it has several issues that need to be addressed. This includes:

- The Create-Session-Request message send by the SGW-C to the PGW should include the TEID and the IP address of the selected SGW-D not just the TEID as the authors mentioned.
- Sending the first packet of each session to the controller will increase the signalling specially if multiple sessions need to be handled in the same way without any special treatment.

- ❖ Packet-Out message is defined by the OpenFlow specification to do a specific job, which is forwarding the packet buffered in the switches. The Packet-Out action in the authors' proposal includes information about both uplink and downlink traffic, which will confuse the OpenFlow processing pipeline and the UE packets will not be delivered to the correct next hop node.

To overcome these issues, the system should include and support the following:

- ❖ Centralize the control plane of both the SGW and the PGW and in this way the controller can configure the data plane devices to work as intended without any issue.
- ❖ Add a local agent that includes a pre-defined list of traffic classes in the FD to reduce the number of Packet-In sent by the FDs to the controller.
- ❖ Use a better way to setup the GTP tunnel in the data plane devices.

In our architecture, the FDs' local agent is configured by the controller during the initial attachment procedure and the first session of the UE new flows are handled locally without the need to send it to the controller unless the flow initiated by the UE do not match any one of the traffic classes maintained by the local agent. In this case the first packet of the session is sent to the controller in a Packet-In message for processing. Our architecture support two GTP implemented methods.

The call flow of these methods is shown in Figure 5.4. In the First method, the controller converts the information provided by the network function applications to a sequence of Flow-mod-Msg and sends them to each FD in the selected path (a sequence of switches from one edge switch to another) to install a flow-entry for each direction, as proposed by [37]. In the second method, the OpenFlow plugin of the FDs is enhanced to include extra functionality that are required to help it understand and handle a new message added as an extension to the OpenFlow 1.3 protocol.

The new OpenFlow plugin is capable of performing all the functionality specified by the OpenFlow standard and at the same time, it can intercept the User-DataPlane-Setup-Request message received from the controller and convert its information to two flow entries: one to handle the uplink traffic and another to handle the downlink traffic.

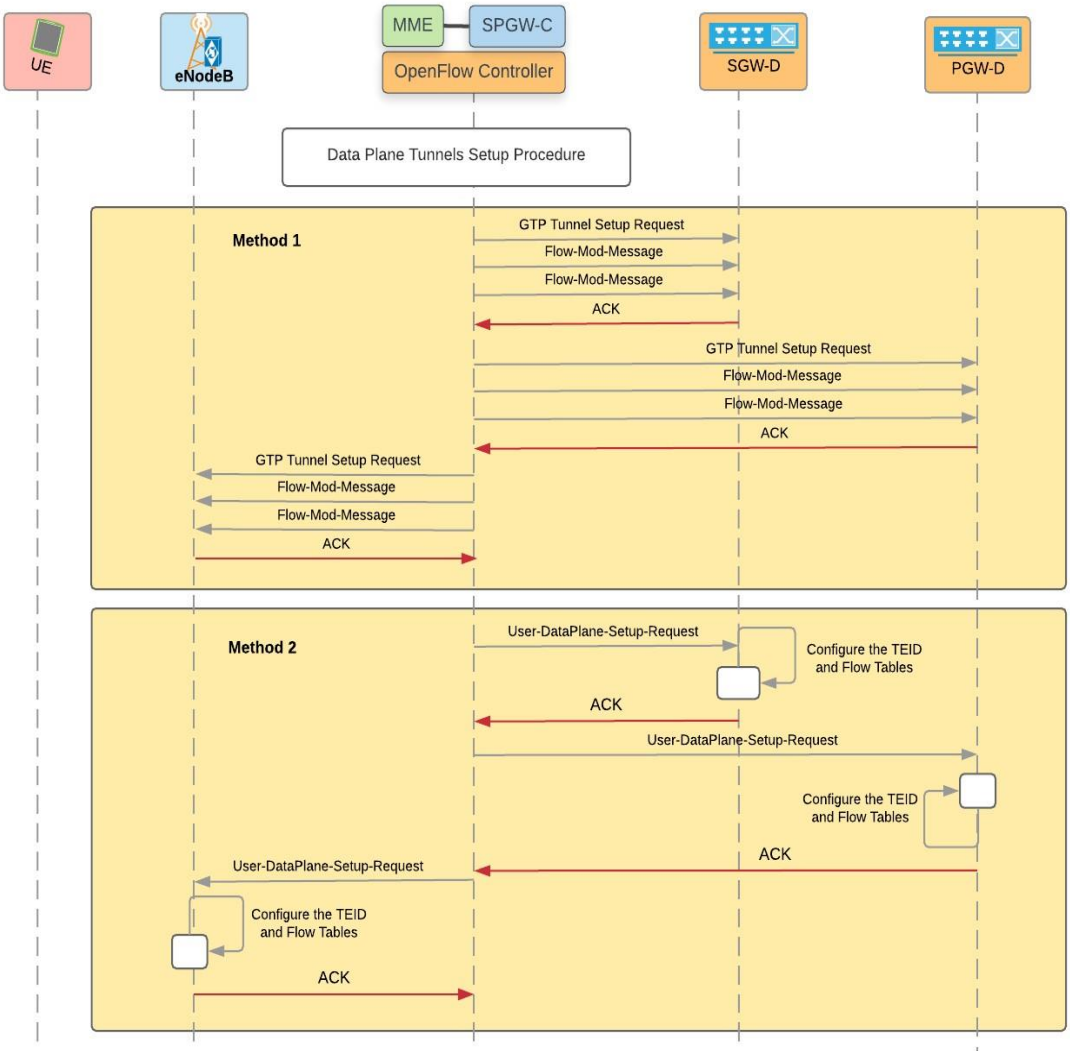


Figure 5.4: GTP tunnel Setup Methods

5.3.2 Resiliency

This work considers resiliency as a method to handle the SGW failure. The 3GPP procedure to handle SGW failure is shown in Figure 5.5 [83]. When a SGW fails, the MME notices that the SGW is not responding to the GTP echo messages that periodically exchange between them. At this point the MME starts the access bearer release procedure to release all the sessions handled by the failed SGW. This procedure requires the MME to communicate with the eNodeBs of all the UEs that have at least a session handled by the failed SGW to perform an S1 bearer release procedure. Then the MME waits for the effected UE to start a service request procedure to setup a new S1 bearer with a new SGW. This procedure gets the job done and all the effective UEs start using the network again but at the same time it increases the signalling loads and requires more time to restore the service back to the effected UEs.

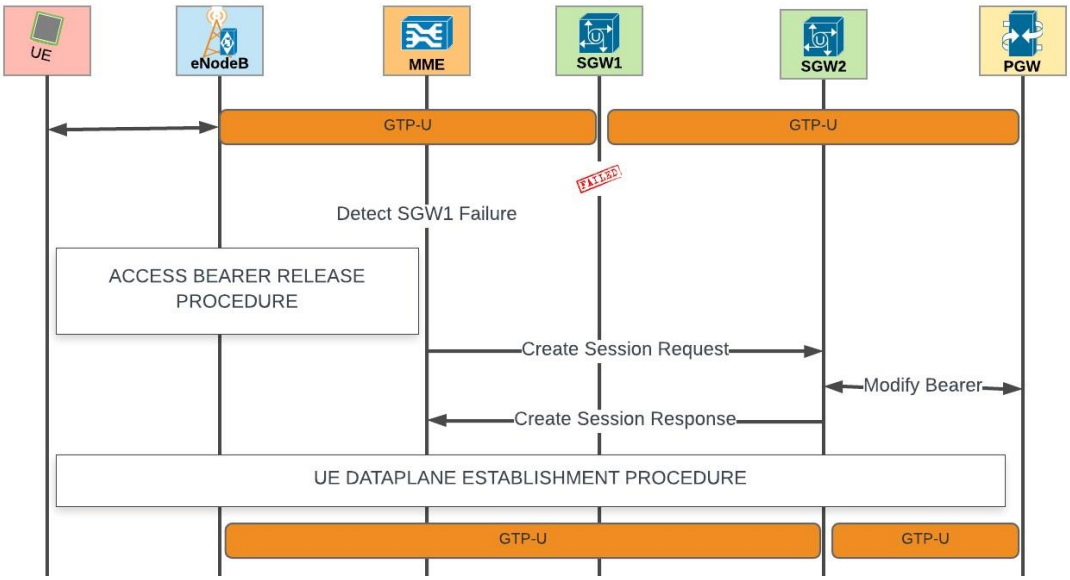


Figure 5.5: SGW failover procedure based on 3GPP specification

The SBMCNA offers a method to handle SGW failure without the need to release all the active sessions handled by the failed SGW and wait for the effected UE to initiate the service request procedure used by the 3GPP. As previously mentioned, the control plane of multiple SGWs is centralized and the data plane processing is handled by extended version of OpenFlow 1.3 switches denoted as (SGW-D). When a SGW-D has failed, the failure is discovered by the NC, the SPGW-C is notified. The latter selects another SGW-D and provides the NC with the updated information of the new link for the impacted UEs. The SGW-C may select one or more SGW-Ds based on the last logged load of the failed SGW-D; if there is a SDW-D that can handle the entire load of the impacted UEs then a single SGW-D is selected, otherwise multiple SGW-Ds are selected, based on the information received from the SGW-C.

The NC configures the flow-table of the newly selected SGW-Ds and modifies the virtual port configuration of the PGW-D to reroute the downlink traffic to the new SGW-D. This process is carried out by changing only the destination IP address and the physical port number if required, since the SGW-TEID values for the downlink traffic on the S5 interface remain the same for the impacted sessions. The NC will repeat the same operation with the serving eNodeBs to redirect the uplink traffic to the desired SGW-D in the same fashion, as the SGW-C does not create new TEID values during the restoration procedure. This chapter compares two methods to configure the data plane FDs.

In the first method, illustrated in Figure 5.6, the controller updates the virtual port configuration of the PGW-D and the serving eNodeB by sending a GTP-Tunnel-Modify-Message (the FD OpenFlow plugin is programmed to update the TEID table when it receives this message). After that the controller first configures the SGW-D2 by sending GTP-Tunnel-Setup-Message to configure the virtual ports for the S1 and S5 GTP tunnels and then it sends two Flow-Mod messages to send the downlink traffic to the S1 virtual port and uplink traffic to the S5 virtual port.

The second method is different because the FDs' OpenFlow plugin is enhanced and modified to perform the operation required in the mobile network. Therefore, the controller only sends a single message to the PGW-D, serving eNodeB and the SGW-D. The FD's OpenFlow plugin behaves based on the type of the received message. In this case the PGW-D and the serving eNodeBs update the TEID table, while the SGW-D fully configures both the TEID and flow-tables.

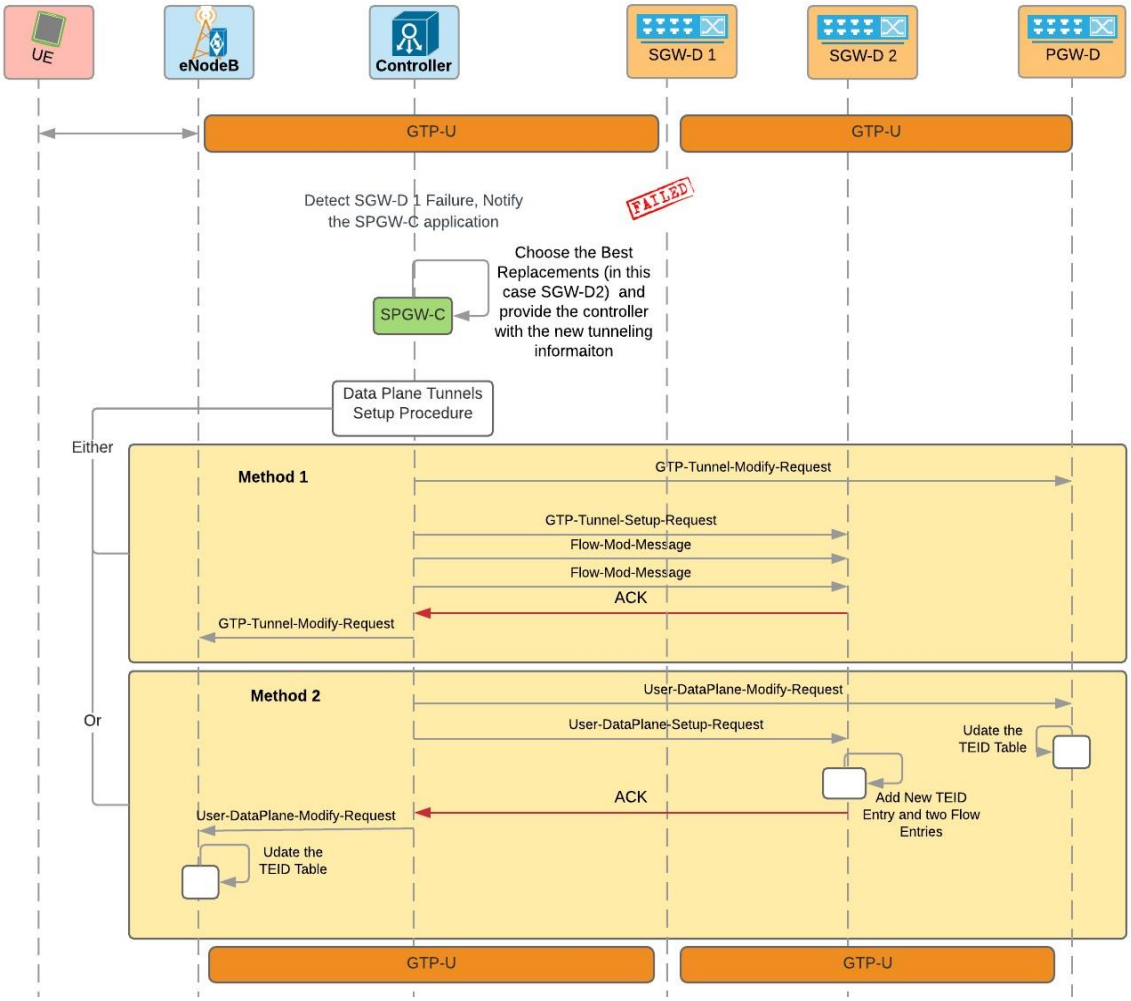


Figure 5.6: SBMCNA SGW failover procedure

A similar solution is proposed by the authors of [4]. They proposed to use SDN with a centralized management for the TEID allocation which helps reduce the time and the signalling cost required to change the traffic direction of the effective UEs. The procedure proposed by the authors is shown in Figure 5.7. The procedure starts by detecting the failure and then the SGW-C application communicates with the PGW (PGW kept the same as specified by the 3GPP) to alter the direction of the downlink traffic to the new SGW-D, then the controller sends a Modify state message to the serving eNodeB to change the direction of the uplink traffic, finally a Packet-Out message is sent to the new SGW-D to configure it.

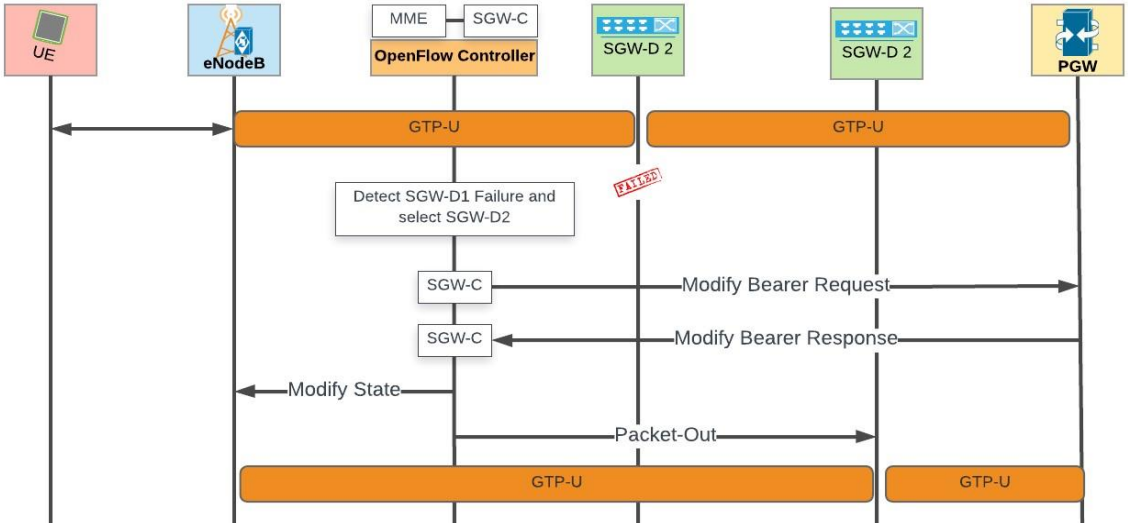


Figure 5.7: SGW Failover procedure based on [4]

In our opinion, this procedure has a problem that needs further investigation because the system cannot be implemented to function in the way they proposed by only using their method. The main issue is the authors proposed to send all the TEID tunnel information in an OpenFlow Packet-Out message. In OpenFlow, the Packet-Out message has its own purpose, which is to forward the packet currently buffered in the FD. The authors proposed to use the Packet-Out to carry the S1 and S5 GTP tunnel information, which includes the IP address and the TEID of the involved entities along with other information. For example, to handle the SGW failure, the authors proposed that the action field of the Packet-Out sent to the new SGW-D should include the IP address and the TEID of the eNodeB, the IP address and the TEID of the PGW, and the SGW S1 and S5 TEIDs. Considering that the author did not propose or suggest any change to the OpenFlow processing pipeline, this action will be ambiguous, and the FD will never be able to forward the traffic to the correct network hop.

5.3.3 Load-Balancing

Load-balancing is a widely adopted workload distribution method used to effectively optimize the network resource usage. Load-balancing in mobile networks like LTE/EPC is based on the Weighting Factor (WF). The SGW selection criteria performed by the MME utilize the weighting factor obtained from a Domain Name Server (DNS). The weighting factor of a specific gateway is derived from its actual capacity compared with the concurrent gateways in the same domain. The MME performs the SGW selection based on this information without any knowledge of the current load of the selected SGW. Therefore, some SGW may suffer from congestion periods considering that the Weighting Factor load-balancing does not consider the SGW load in real-time and the MME may keep assigning the traffic of new UEs to the same SGW. Basic representation of the load distribution based on the 3GPP specification is shown in Figure 5.8.

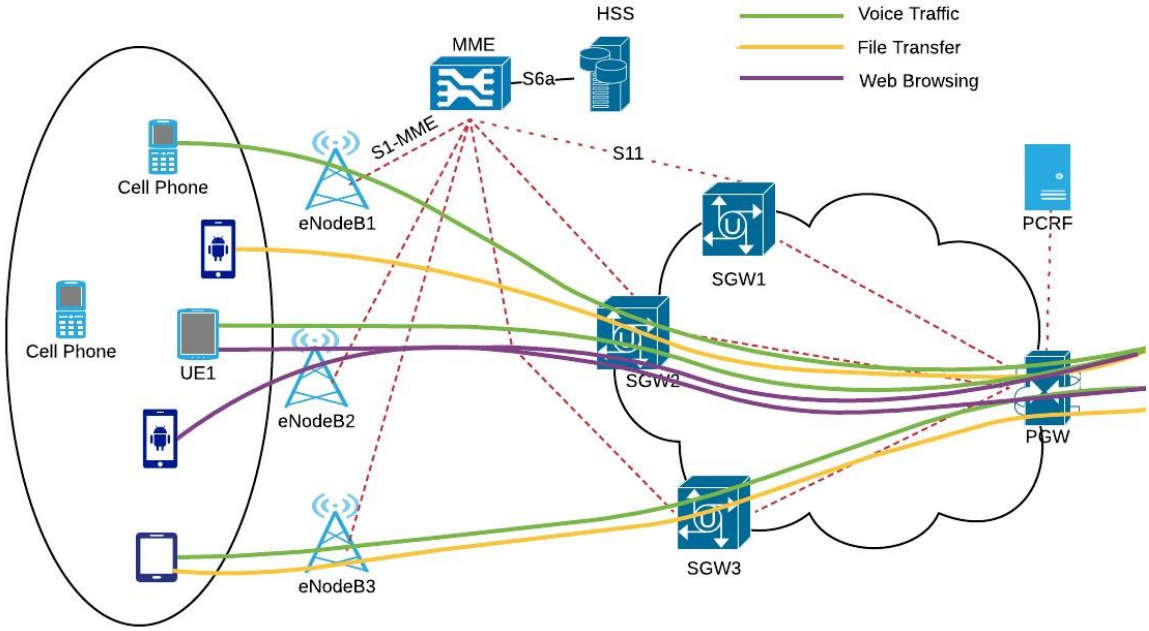


Figure 5.8: 3GPP Load-Balancing Solution

With the emergence of SDN, load-balancing can be handled in a non-dedicated way, which could significantly reduce the cost. The model proposed by [4][40] used the SDN concept to perform per-session load-balancing that distributes UE sessions across multiple SGW-Ds in the same domain. The authors proposed to use the controller knowledge about the network statistics to offer more efficient solution for the load-balancing in the mobile network. Considering that the first packet of any new session

is sent to the controller for processing, the controller can use the real time statistics with the session type to balance the network loads between the SGW-Ds, as shown in Figure 5.9. The authors argue that this solution will lead to a better traffic distribution.

Although this mechanism provides optimized resource utilization, it also introduces a few issues. First, the first packet of each session needs to be sent to the controller even if the session does not need special treatment and can be handled in the same way the eNodeB handles the other UE sessions. Secondly, the controller may route UE sessions through multiple SGW-Ds as with the sessions of UE1 shown in Figure 5.9. UE1 web traffic is handled by the SGW-D1 and the voice traffic is handled by SGW-D2. In this case, if the UE1 hands over from one eNodeB to another, then more control signalling messages are required to change the downlink traffic direction. This is because the controller will have to modify the flow entries of multiple SGW-D devices compared to only one in the 3GPP method of implementation.

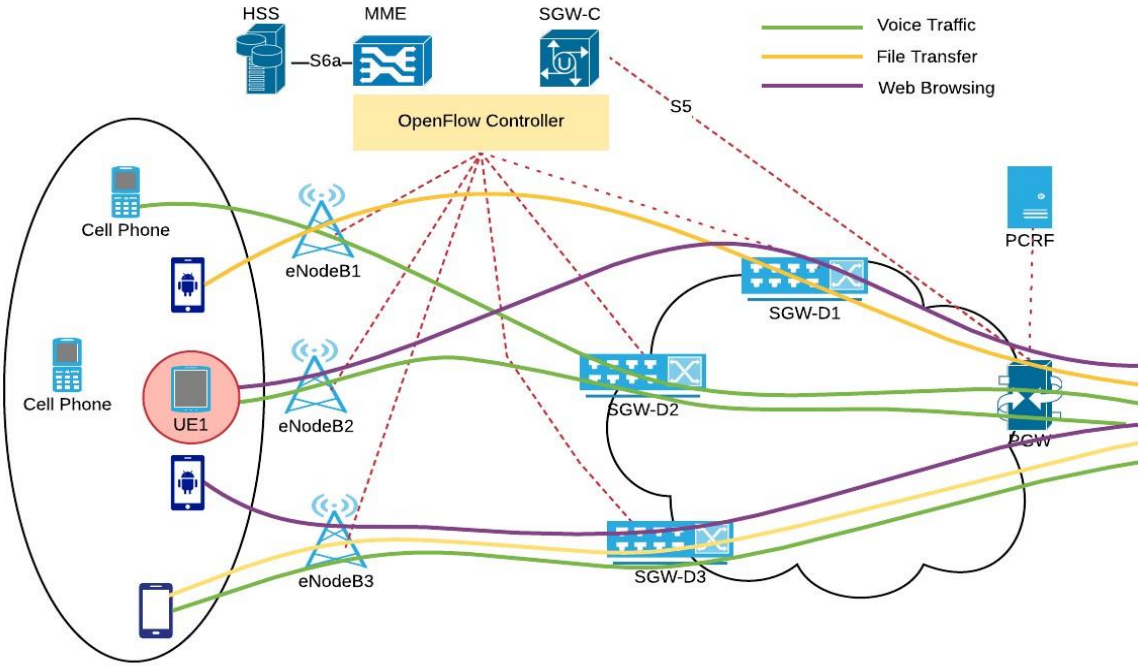


Figure 5.9: Load-Balancing Solution proposed by [4]

In SBMCNA, the concept of load awareness offered by the SDN controller is utilized to implement the load-balancing mechanism. The solution attempts to reduce the number of Packet-In-Messages sent to the controller and the number of control messages sent by the controller as a response to the Packet-In. Therefore, network transport devices are equipped with a local SDN agent that has the capacity to maintain a list of pre-defined traffic classes. This list is temporary and follows the same rules of

OpenFlow Entries, since it has an idle and hard timeout and can be modified or deleted based on the controller requirements.

In the UE-attachment procedure, the centralized controller computes the routes that satisfy a set of requirements based on users' policies and profiles. Then it configures the forwarding local agent with a predefined list of traffic classes. Therefore, different UE sessions that belong to the same traffic class are handled locally based on the instructions provided by the controller without the need to send the first packet to the controller. Also, the monitoring application is used to properly and efficiently collect fine-grained statistics from the network devices. The monitoring application has a process that is responsible for building a historical profile for each user. This profile incudes the current usage, the user's active hours, and mobility patterns, among others. The load-balancing application utilizes the statistics obtained by the monitoring application to offer fair distribution of the network traffic between the different SGW-D devices in the same domain. If one of the SGW-Ds suffers from congestion, the controller seamlessly moves some of the UEs to another SGW-D to provide better resource distribution without increasing the signalling load. If the application needs to move some of the UE sessions to another SGW-D, then the load-balancing application is programmed to use the historical profile built by the monitoring application to choose the best UEs (normally the one that rarely changes it location, especially at this particular time of the day), then it moves some of its sessions to different paths through different SGW-Ds. In this case, the load-balancing application will assign UE session to a different SGW-D only if there is no other solution and the UE profile is used to pick the best UEs for the job.

5.4 Network Model

The network topology used in our simulation is shown in Figure 5.10. It consists of the core network, access network and the InternetHost. The core network comprises of an OpenFlow controller and three enhanced OpenFlow FDs that are capable of GTP tunnelling. The core network adds realistic delays that simulate the time required for a packet to travel from the InternetHost to the access network and vice versa. Specifically, 8ms for the S1-U interface, 4ms for the S5 interface and 8ms for the SGi interface.

The EUTRAN consists of three OpenFlow capable eNodeBs and a set of UEs, where each eNodeB is equipped with an omnidirectional antenna with 40 dBm transmission power, 18 dB antenna gain, 5 dB noise figure and 2 dB cable loss. The RLC layer at the eNodeB is configured with the Unacknowledged Mode, with a fixed PDU size of 40 bytes. A realistic channel model that considers both path loss and fading is used in our simulation. The Urban Macro Path Loss Model and the Jakes model for Rayleigh fading is used in all scenarios. The UEs are configured to use 26dBm transmission power and a linear mobility model with a speed of 1m/s.

The network is implemented in OMNeT++ version 4.2.2 utilizing its independently developed open source INET 2.3 library together with simuLTE and OpenFlow 1.3 extensions. OMNeT++ is run on the Windows 7 Pro operating system hosted by a Dell Precision Tower equipped with Intel Xeon E3-1246V3 / 3.5 GHz CPU with 8 GB of RAM. INET library and the aforementioned extensions provide flexible tools that are utilized in the creation of the simulation platform used to evaluate and quantify the efficiency and the performance of the proposed architecture. To evaluate the integration of SDN in EPC, a prototype implementation was developed that introduced several modifications, improvements and extensions to the base version of both simuLTE and OpenFlow 1.3 models, in order to create a simulation platform that offers complete tools to enable the validation of the system performance under different circumstances. These extensions include: the elements detailed in the next sub-sections.

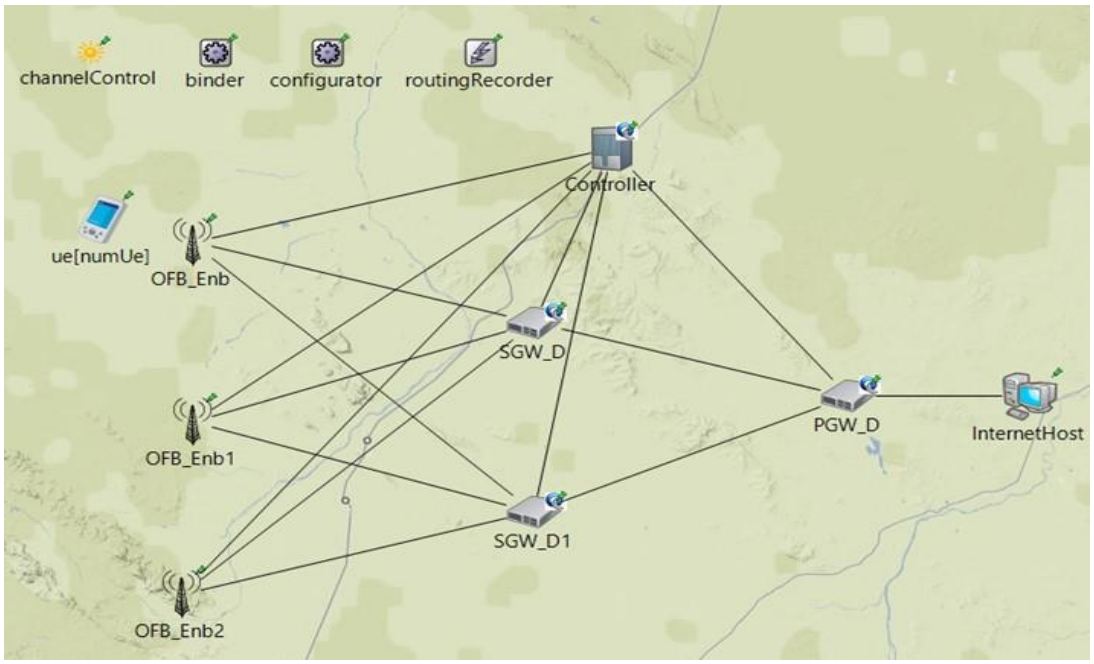


Figure 5.10: Network topology used in OMNeT++ simulation

5.4.1 NC Controller Module

Figure 5.11 illustrates the OMNeT++ module of the SDN controller node. It consists of an application layer, a network operation system and communication interfaces. The mmeApp module works together with the spgwApp modules to realize the network functions and services by sending instructions or requirements to the ControllerOS module to relay them to the networking components. The spgwApp module implements the control plane functionality of the SGW-C and PGW-C previously explained in Section 5.2.3.

The ControllerOS module is also responsible for extracting information from the network devices and communicating them back to the application layer. To achieve this, the ControllerOS module utilizes two framework applications, namely: topology discovery and monitoring applications, to extract and maintain network information. This information includes an abstract view of the network, statistics, and events that describe what is happening. Since HSS is not implemented in this model, the userTable module is used to store the UEs subscription profile and for simplicity, the UE authentication procedure is not implemented.

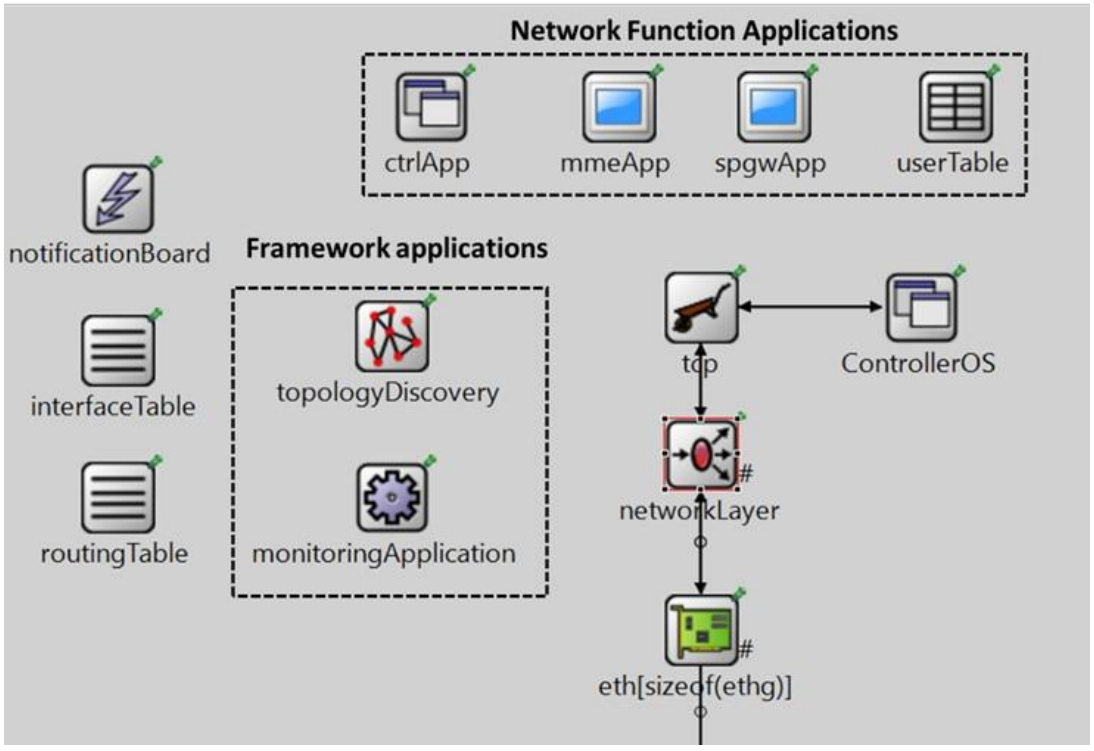


Figure 5.11: Controller Module in OMNeT++

The SDN controller module implements most of the communication interfaces, although not all of them have the accuracy that is specified by the standard. The OMNeT++ signal mechanism is used as a synthetic representation of the northbound interface between the ControllerOS and the application layer. The southbound interface has been modelled with high accuracy and real packets are built and exchanged between the controller and the FDs. The SDN controller model has a TCP connection with every FD in the network exclusively for the purpose of exchanging control and statistics messages.

OpenFlow protocol is used to realize the southbound interface by handling the communications between the individual network devices and the controller. The controller module can work in two operational modes and each mode represents the way the controller configures the data plane devices. The operational mode specifies the number and the type of messages that need to be sent to the FDs to setup the UE GTP tunnel. These modes are NORMAL, and ADVANCED respectively. The controller parameters specified by the ini file are used to specify the operation mode of the controller, for example in order to make the controller operate in the normal way (sends a sequence of Flow-Mod messages to setup the GTP tunnel), the below line of code is used in the ”ini” file.

```
**.onDemandController.operationMode="NORMAL"
```

In the same way to change the operation mode from NORMAL to ADVANCED the below line of code is used. In this mode the controller sends a User-DataPlane-Setup-Request message to each FD in the selected path to instruct it about how to handle the UE upcoming traffic.

```
**.onDemandController.operationMode="ADVANCED"
```

5.4.2 GTP-Capable-OpenFlow-Switch Module

The module depicted in Figure 5.12 represents an overview of the different components of the GTP-capable OpenFlow switch. It mainly consists of the OpenFlow plugin and data processing pipeline. Tunnelling in this implementation is realized through the

deCap-vPort and enCap-vPort modules that simplifies header's manipulation to support GTP encapsulation and decapsulation.

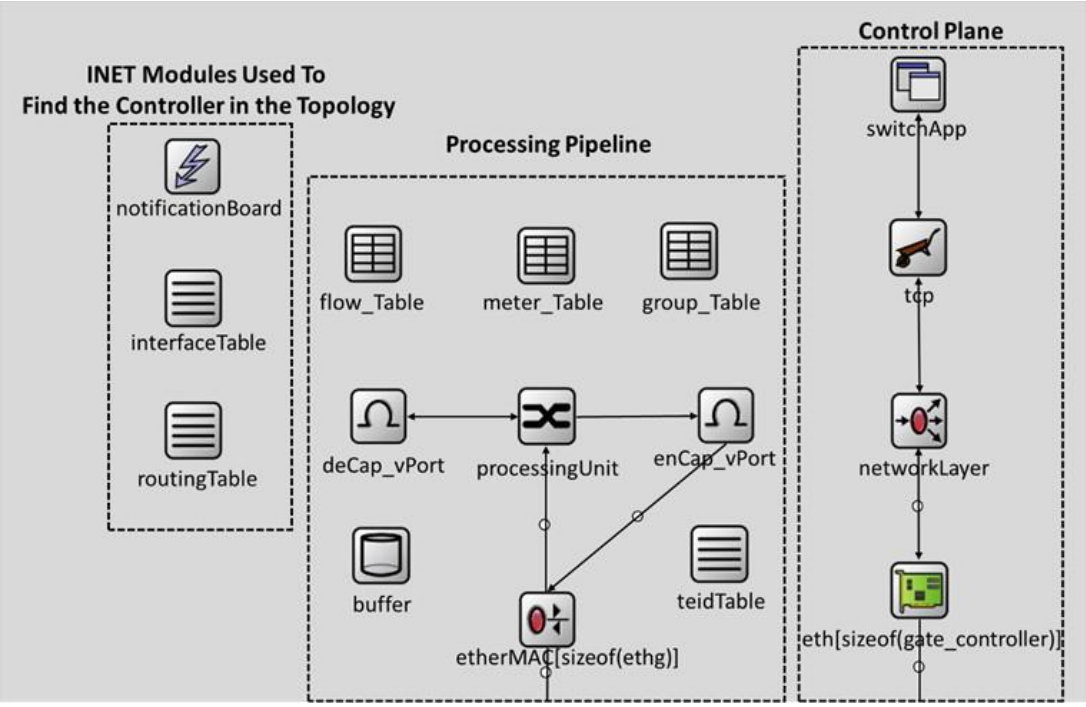


Figure 5.12: Implemented Model of a GTP capable OpenFlow Switch

5.4.2.1 OpenFlow Plugin

The switchApp module depicted in Figure 5.12 represents an extended version of the original OpenFlow plugin module that presented in [82]. It has a local program with a focus on the interpretation of the new OpenFlow messages and converts them to flow-entry. It is also responsible for providing the controller with aggregate statistics of the FD. Therefore, switchApp can handle both the standard OpenFlow control messages and the User-DataPlane-Setup-Request. The latter is a new message that proposed to carry information to setup UE uplink and downlink tunnels.

For example, in the service-request procedure, the controller configures the GTP tunnel by sending User-DataPlane-Setup-Request to the selected FDs. The message sent to the SGW-D contains the SGW-S1-TEID, SGW-S5-TEID, eNodeB-S1-TEID, eNodeB-IP-Address, PGW-S5-TEID, PGW-IP-Address, physical output ports, and QoS parameters. Upon the reception of this message the local program configures the TEID Table to add two entries that hold the virtual port encapsulation parameters. The Tunnel-Id used as a key in the TEID-table. Let us say that the TEID table has two entries, 100 and 200; the former to encapsulate the packet with eNodeB information

(TEID, IP-Address, UDP Port Number (always 2152 in our simulation)) and the latter to encapsulate the packet with PGW information. Then the local program installs two flow entries to map the traffic to the correct virtual port. The first entry matches traffic with SGW-S1-TEID to be sent to the virtual port 200 and the second entry matches traffic with SGW-S5-TEID to be sent to the virtual port 100.

5.4.2.2 Switch Data Processing Pipeline

The data message received by the physical port is passed up without any modifications to the processingUnit module, which implements the OpenFlow switch functionality on the data plane. The processingUnit is pre-configured with flow-entry that has the highest possible priority. This entry matched traffic that has UDP destination port equal to 2152 (GTP-U traffic) and sends it to the deCap-vPort. The returned packet from the deCap-vPort consists of the IP header with a transport-layer header plus payload; the GTP TEID is also written into the lower 32 bits of the tunnel-Id metadata. Another flow-entry matches the GTP-U tunnel utilizing the tunnel-Id metadata and sends it to the enCap-vPort.

The enCap-vPort operation is directly mapped to the information specified by the TEID-table. When the packet arrives at the enCap-vPort, it first extracts the output tunnel-Id from the control information attached to the packet, and then it looks up the tunnel header information in the TEID-table. If a match is found, then the packet is encapsulated with GTP utilizing the tunnel information of the TEID-table as shown in Table 5.1. If no such tunnel information is present, the enCap-vPort checks if the value of output tunnel-Id is equal to one of the physical ports; then the packet is encapsulated within Ethernet headers utilizing the information included in the control information attached to the received packet. Otherwise, the packet is forwarded to the controller with an error indication. The TEID-table consists of one or more entries. Each entry is composed of tunnel Id, TEID value, next hop address, next hop port number, and QoS parameter, as shown in Table 5.1.

Table 5.1: Main Components of TEID Entry

Tunnel Id	Next Hop TEID	Next Hop Address	Next Hop Port No	QoS Parameters	Physical Port No

5.4.3 OpenFlow-Based eNodeB

The OpenFlow-Based-eNodeB keeps the same radio protocol stack as specified by 3GPP standards leveraging the LTE NIC module from simuLTE [78], while its S1 interface is replaced by an OpenFlow switch. To handle UE authentication, authorization, and mobility management, a new set of signalling messages is adopted to exchange the necessary information during the UE operations. The main control messages included in the OMNeT++ model are Initial-UE-Message, Initial-Context-Setup-Request, Initial-Context-Setup-Response, Switch-Path-Request, and Switch-Path-Request-ACK. The new messages have similar information to those specified by the 3GPP standards but are structured using OpenFlow protocol.

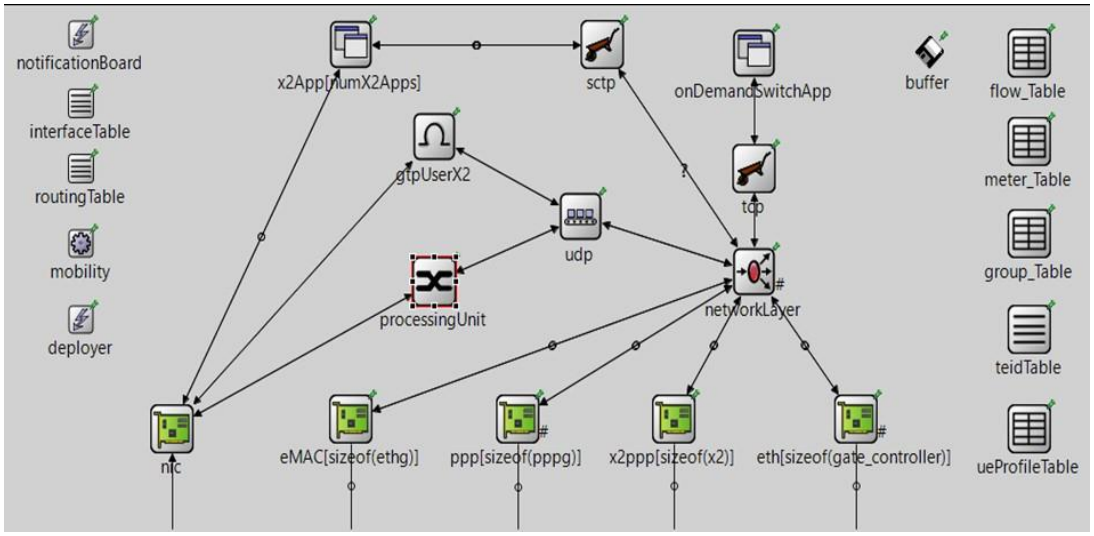


Figure 5.13: OpenFlow-based eNodeB Module Structure

Figure 5.13 shows the basic structure of the eNodeB module. It mainly consists of 4 parts:

- The first part represents the connection between the eNodeB and the network controller through the OpenFlow link. This part consists of 4 modules: namely, onDemandSwitchApp, tcp, networklayer, and ethernet Interface.

  The onDemandSwitchApp module represents the OpenFlow plugin. It is more advanced and complicated compared to the OpenFlow plugin specified by the OpenFlow 1.3 specification. This module is responsible for sending and receiving all the UEs messages related to the mobility and session establishment, modification and tear down. No authentication messages are sent or forwarded by this module because this feature is not implemented by either

the MME or the HSS. This module communicates with the controller using a TCP connection. The other three modules of this part are implemented by the INET library [79] and reused here to setup the TCP connection with the controller. These modules are explained in detail in Chapter 4 Section 4.5.1.1. The onDemandSwitchApp module has two operating modes: First is normal and second is advanced. The mode in which this module operates is specified by the eNodeB parameters specified in the ”ini” file. Currently the only difference between the two operating modes is that the advanced mode has the ability to intercept and read the User-DataPlane-Setup-Request message to configure the tables of the eNodeB module, while the normal mode cannot. The below line shows how to configure the eNodeB to use the normal mode.

```
**.onDemandSwitchApp.operationMode=”NORMAL”
```

Changing the word NORMAL to AVANCED will alter the onDemandSwitchApp behaviour to work in the advance mode. If the onDemandSwitchApp received a User-DataPlane-Setup-Request message while it is configured to use the normal mode, then the message will be dropped, and the simulation stop with an error message explaining the problem. This can happen if the controller is configured to use the advanced mode while the eNodeB is configured to use the normal mode. Figure 5.14 shows the onDemandSwitchApp operation in the advanced mode.

Figure 5.14 shows how the onDemandSwitchApp module of the eNodeB module handles messages coming from the controller through the TCP connection. This module also handles messages sent from the NIC module. These messages are sent by utilizing the OMNeT++ signal mechanism and handles by re-implementing the receiveSignal method ().

- The second part is the NIC module, which is built by the simuLTE model and modified by us to includes features required to support the OpenFlow interconnection. It implements the radio protocol stack. It has four connections, one to the air interface to exchange messages with the UEs, the others with the X2App, gtpUserX2, and processingUnit respectively. The NIC module also uses the OMNeT++ signal mechanism to exchange information with the switch’s OpenFlow plugin(onDemandSwitchApp). The NIC module use the

X2App connection to exchange control messages with its neighbour eNodeB over the X2 interface e.g handover request. The gtpUserX2 connection is used to send UEs data traffic during the handover. Finally, the processingUnit connection is used to send/receive traffic to/from the S1-U interface.

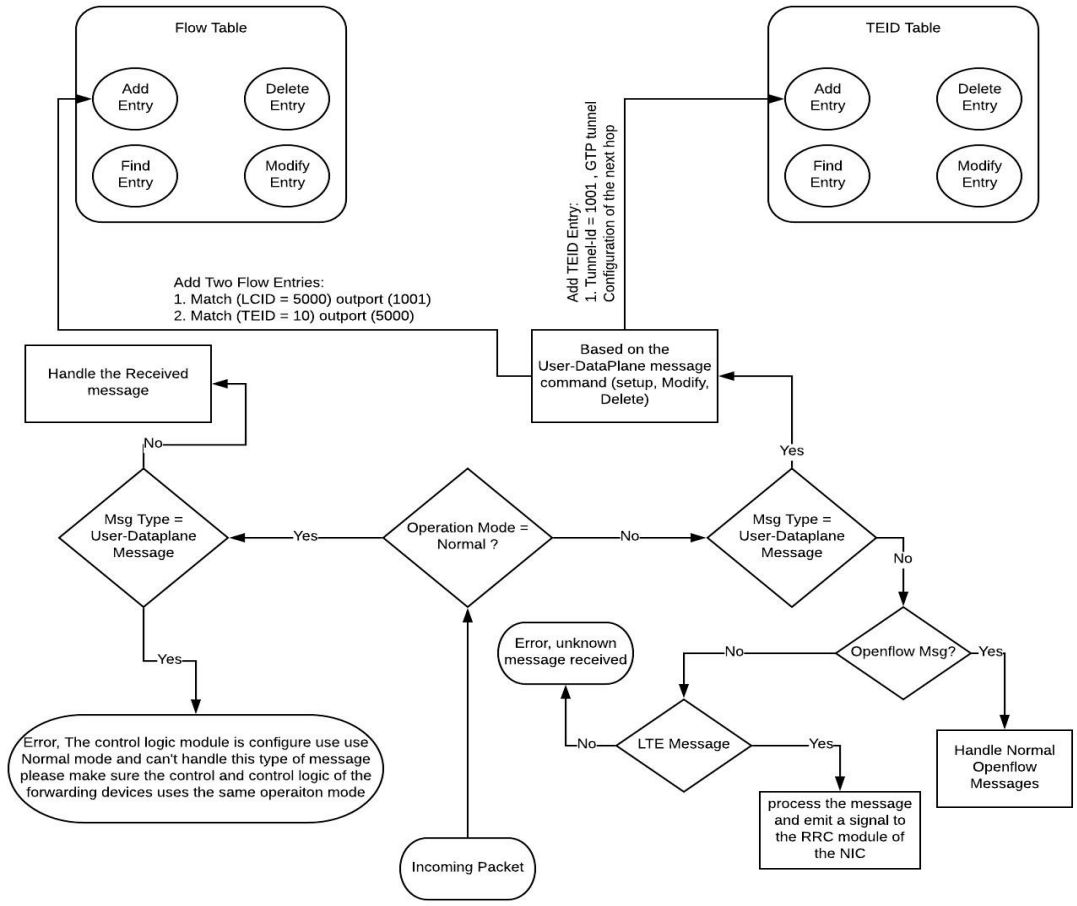


Figure 5.14: onDemandSwitchApp Advanced Mode Operations

The third part represents the connection between the eNodeB and it neighbour eNodeBs. This part consists of several modules that simulate the data and control plane to exchange data and control messages between the neighbouring eNodeB. Currently the module only has the control messages required to support X2 handover. The X2 control plane modules includes, X2App, sctp, networkLayer, and X2ppp Interface, while the data plane modules include, gtpUserX2, udp, networkLayer, and X2ppp Interface. These modules represent one-to-one mapping with the control and data plane protocol stack specified by the 3GPP specification. These modules are implemented by the simuLTE [78] and the INET library [79] and reused in our module to support the X2

communication between the neighbouring eNodeB. These modules are not mandatory, and one need to configure them only if there is a point-to-pint link between the eNodeBs in its simulation network. This section briefly describe the X2App and gtpUserX2 modules. The X2App module is consist of two blocks: namely, server and client and each eNodeB must have N of the X2Apps, where N equal to the number of X2 point-to-point links it has with the other eNodeBs.

For example, if the eNodeB is connect to three neighbouring eNodeBs, then it must have three X2Apps which is equal to the number of eNodeBs that it has X2 point-to-point link with. The server block is used to send x2 control messages and the client block is used to receive X2 message from the other eNodeBs. As previously message X2 control message sent over SCTP session and that's why the X2App module has a connection to sctp module. The later helps setup the SCTP session for the X2 control communication. The gtpUserX2 is responsible for encapsulating the packets coming from the INC module with GTP header before passes it to the other networking modules to add the required headers before sending it to the next hop. At the same time, this module decapsulate packets coming from the UDP module and deliver it to the NIC module.

- The last part is responsible for handling the data traffic sent and received to/from the S1-U interface. In our module, this part is implemented by the processingUnit module. The latter is a is a modified version of the open flow processing module explained in Chapter 3 section 3.4.2.1B. It is a compound module that has multiple virtual ports to help with the encapsulation and decapsulation of the received packets. Two virtual ports are connected to the UDP module one to decapsulate the received packet by removing the GTP header and send the inner packet to the processing pipeline to find the correct mapping between the TEID and the LCID. The basic structure of the processingUnit module is shown in Figure 5.15. The downlink traffic coming from the UDP module is passed through the fromUdp gate to the deCap-vPort sub-module where GTP header is removed and the inner IP packet with the TEID value attached as a control object is forwarded to the pipeline module. In the pipeline module, the TEID value is used to lookup the forwarding table and the matched entry should specify the LCID (used in simuLTE as a part of NIC

module to uniquely identify a session with in the system. In simuLTE this value is generated by the PDCP module, but the module is modified to expect receiving the value attached with the receiving packet). The pipeline attaches the LCID value to the packet and send it through the toNic gate to the NIC module. The latter sends the packet through the correct radio link to the UE.

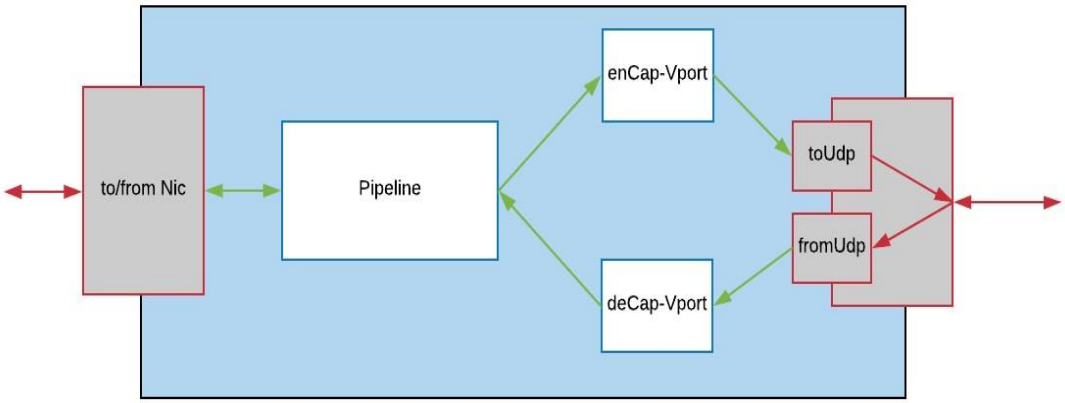


Figure 5.15: Internal Structure of the ProcessingUnit module of the eNodeB Node

In the same fashion, uplink traffic is passed through the fromNIC module to the pipeline module. The latter expect to receive an IP packet with a LCID value attached to the packet as control object. The LCID used to lookup the flow-table and the output port of the matched entry (represent the vPortId) is attached to the packet before sending to the enCap-vPort module. In the enCap-vPort the vPortId is used to lookup the TEID table to fund the correct GTP configuration and based on the result and IP packet is encapsulated with GTP header and sent to the UDP module through the toUdp gate and UDP and IP headers are added by passing the GTP packet through the networking modules until it reach the Ethernet interface where the layer 2 header information is added to the new packet before send it to the network.

5.4.4 Local Agent

Local program used as part of the module structure of both the FD and the eNodeB modules. The main structure of these modules depicted in Figure 5.12 and Figure 5.15 are extended to include OFAgent and ueProfileTable modules. The next sub-sections describe the OFAgent module structure and operations.

5.4.4.1 OFAgent Module

Structure of the OFAgent Module is defined by the ”ned” file of the OMNeT++ simulator. This file describes the module structure and defines its parameters and connection gates. It also, associate each parameter with default value if it necessary. Figure 5.16 shows the basic structure of the OFAgent module. It has two gates to connect the module with the OpenFlow plugin module. These gates are used to send and receive messages to/from the OpenFlow plugin module. The names of these gates are swAppIn and swAppOut. Messages send by the OpenFlow plugin module should be received by and only by the swAppIn. In the same fashion messages sent to the OpenFlow plugin module should be sent through the swAppOut gate.

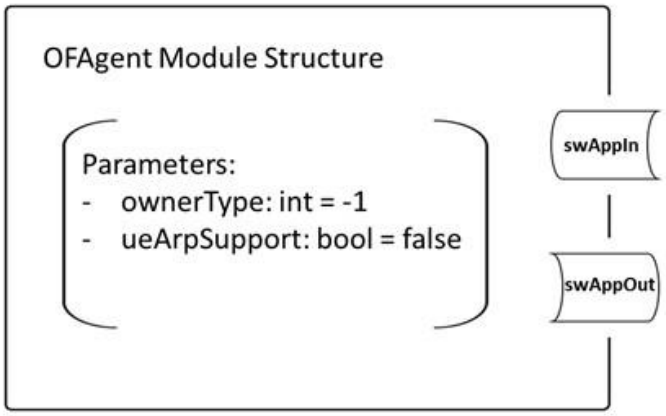


Figure 5.16: Agent Module Structure

In OMNeT++ [69], all simple modules are implemented as subclass of the cSimpleModule. Although the cSimpleModule defines many of the simulation-related functionality, but it cannot perform anything useful by itself. Therefore, it is mandatory for the subclass module to redefine one or more of the base-class (cSimpleModule) virtual methods. The OFAgent redefine the functionality of the handleMessage() and initialization() methods to properly initialized the module and handle the received messages. The OFAgent handles both control and data plane messages. Control messages received through the swAppIn and handled by the handleMessage() method. Data plane message emitted by the ProcessingUnit module through OMNeT++ signal mechanism. In order to receive these signals, OFAgent register itself as a listener in the initialization phase of the simulation.

To be a listener for a specific signal, it is required to be sub-class of the cListener. The latter is do-nothing implementation suitable as a base class for other listeners.

Therefore, the OFAgent module is implemented as sub-class of both the cSimpleModule and cListener modules as shown in Figure 5.17. It redefines and implements its own version of receiveSignal() method to handle the data plane messages sent through the signal mechanism. Furthermore, it can access all tables in the FDs compound module. This includes the flow-tables, group and meter tables, TEID and ueProfileTables. The OFAgent defines a unique identifier to each table. In the initialization method, the module gains access to each table and assign value to it identifier. The identifiers then are used to refer to the tables at different stages of the simulation.

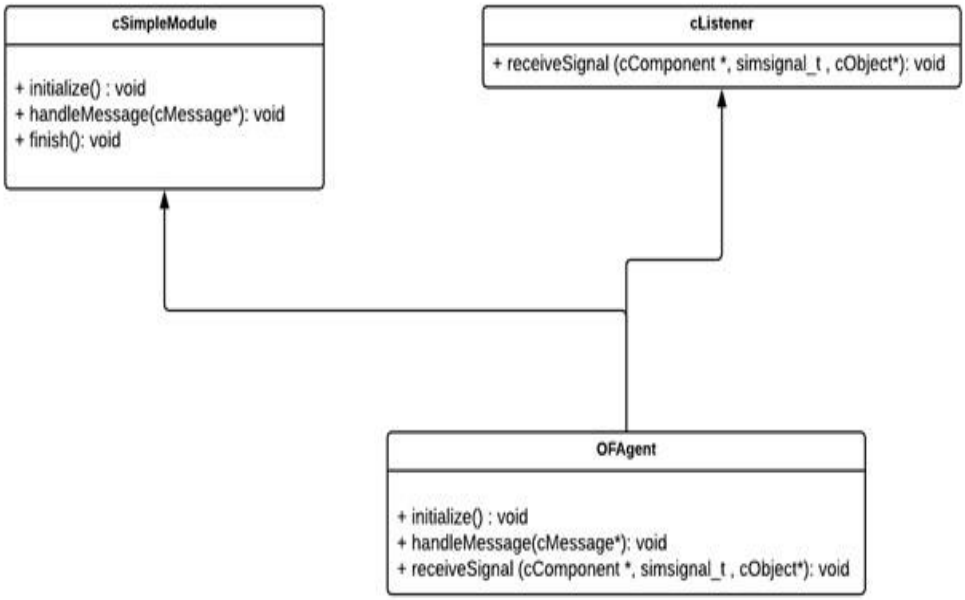


Figure 5.17: OFAgent class tree structure

OFAgent module overwrite handleMessage() and receiveSignal() methods to implement its own version of handling the messages received from the OpenFlow plugin and the processingUnit modules respectively. As shown in Figure 5.18, upon receiving a message, the OFAgent module check the source of the message to differentiate between the control and date plane messages. The messages received from the OpenFlow plugin (switchApp) module classified as control messages and handleMessage() method is called to process the received control message. If the message received through OMNeT++ signal mechanism, then the message classified as data message and receiveSignal() method is called to deal with the received message.

5.4.4.2 Control Messages

The same implementation concept is used in all our modules. That is the handeMessage() method works only as a decision maker, which only inspect the received message and based on the value of a specific field, a more specialized method is called to handle the message properly. In this case, the message type is used to identify the received message. Currently the module only has a handler for the OFPT UE PROFILE MOD message but it can be extended to support many more features. If a message received by the OFAgent module with a different message type, the simulation is stopped with a description about the error to explain that the module can't support the received message. If the type of the received message is OFPT UE PROFILE MOD, then the handleUeProfileModMessage() is called to process the received message. The handleUeProfileModMessage() uses the command field attached to the received message to specify the correct operation to perform. The method expects the command value to be one of the following: OFPUPC ADD, OFPUPC MODIFY, or OFPUPC DELETE and based on the command it configures the ueProfileTable to adding a new entry, modify or delete existing one. The upper part of Figure 5.18 shows the procedure used by the OFAgent to handle control plane message.

5.4.4.3 Data Plane Messages

The processing pipeline of the standard OpenFlow FD module has been modified. Now the OFAgent is responsible for the NO-MATCH-FOUND request signal instead of the OpenFlow plugin module. Upon receiving the NO-MATCH-FOUND signal the OFAgent module obtain the received packet from the buffer module and lookup the ueProfileTable using either the source or the destination IP address of the received packet. If no match is found, then Packet-In message is sent to the controller through the OpenFlow plugin module. Otherwise, header information of the packet is match against the traffic template list associated with the ueProfile. The same as before if NO-MATCH-FOUND Packet-In message is sent to the controller through the OpenFlow plugin module. If the packet headers match one of the traffic template list, then a new flow-entry is added to the flow-table. The header of the received packet is used as a matching field combined with the instruction attached to the traffic template is used to define the flow-entry components.

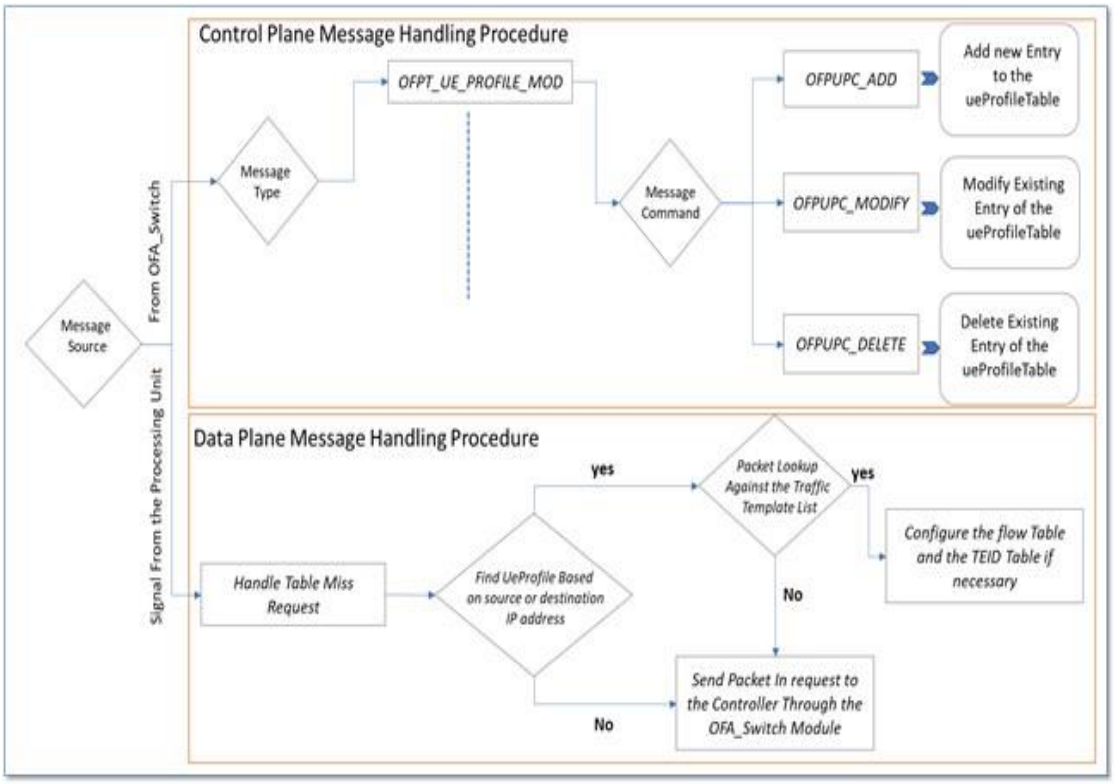


Figure 5.18: Control and Control Plane Operations of the OFAgent module

5.5 Simulation Setup and Results

In order to study and analyse the system performance of both proposals, multiple experiments that employed different numbers and types of applications are used. Each experiment lasted for 100s and was replicated five times with different seeds to exclude simulation artifacts and to obtain mean and confidence intervals over the different repetitions. At the start of each experiment, the UEs were randomly placed within a square of a given size at a maximum distance of 100-meter from the eNodeB and moved linearly at the speed of 1 m/s. The common ”ini” configuration between all the simulation scenarios is shown below.

```
record-eventlog = true
result-dir = Result
sim-time-limit = 100s
Repeat = 5
seed-set = $repetit
```

5.5.1 Applications

Three types of applications are used in these experiments. These application works like a traffic generator in some modules (e.g UEs) and traffic sink in another. In the first experiment, PING application is used and in the second experiment TCP and UDPVideoStream applications are used. More specifically TCP application is used in scenario 1 of experiment 2 and UDPVideoStream application in scenario 2 of the same experiment. The next sub-sections explain in detail these applications.

5.5.1.1 PING APP

An application module implemented by the INET library [79]. This application used only as a traffic generator. The module generates and sends ICMP echo request messages. At the same time, it calculates and records the RTT and packet loss of the received ICMP echo reply. Several parameters included in the ”ned” file of the module to specify the sending time, the time between the ICMP echo request messages and the address of the destination node. The startTime, sendInterval, destAddr parameters are used to fill these roles. A sequence number is attached to every request packet sent by this module and it expect that the reply packets arrive in the same order. For example, if reply packets with the sequence number (e.g 1, 2, 3, 5) received by the Ping application module, then the missing packet (number 4 in this example) is counted as lost. When the module receives a packet with a sequence number less than the sequence number of the last packet it successfully received it will be counted as out-of-sequence arrival. Therefore, the actual packet loss equal packet Loss minus the out-of-order packets.

5.5.1.2 TCP Application

Multiple TCP applications are available in the INET library such as TCPBasicClientAPP, TCPEchoAPP, TCPGenericSrvApp, TCPSessionApp, TCPSinkApp, TCPSrvHostApp and TelentApp. All these applications are built as child class of the TCPBasicBase. The latter represent a class the includes all the common functionality and operation shared by all these types of TCP applications. In the simulation, the TCPSessionApp is used only as a traffic generator and TCPSinkApp as a sink application. Therefore, this section briefly explains these applications and readers are advised to read the INET manual [79] for more information about the other types of the TCP applications.

- **TCPSessionApp**: is used as traffic generator application that open a single TCP connection to send a given number of bytes to the destination node. The application has multiple parameters that specify:
  - The local and remote socket configuration [IP address: port number], this is specified by the localAddress, localPort, connectAddress and connectPort parameters.
  - Traffic characteristics, these include when to open and close the session, the size and the type of the data need to be sent. The tSend and tClose parameters are used to specify the opening and closing time of the TCP session and sendBytes and the dataTransferMode parameters are used to specify the size and the type of the data need to be sent. Currently dataTransferMode can be set only to one of the following values ”bytecount”, ”object” and ”bytestream”.
- **TCPSinkApp**: simple TCP application that is programed to accepts any number of incoming TCP connections, and discards whatever arrives on them. This application measure and maintain the number of received packets and the end-to-end delay of each packet it receives.

5.5.1.3 UDP Video Stream APP

An application that consist of two complementary modules: namely, UDPVideoStreamCli, and UDPVideoStreamSrv. The former works a traffic sink and the latter works as a traffic generator. The client module sends a single packet to the server module to request a video stream. The application starting time, server IP address and the port is obtained from the startTime, serverAddress, and serverPort parameters. When a stream request packet is received by the UDPVideoStreamSrv module, it uses the videoSize parameter to draw a random video stream and start sending the stream to the client. The UDPVideoStreamSrv sends packets with size equal to packetLen parameter every sendInterval until videoSize is reached. The module works as CBR traffic generator if the packetLen and sendInterval is set to constant values. The UDPVideoStreamCli records several statistics about the received UDP stream. This includes the statistic of received packets and received bytes and statistic of end-to-end delay of incoming packets.

5.5.2 Signalling Experiment

This experiment focuses on measuring and evaluating the control-signalling messages sent by the controller to:

- Setup the GTP tunnel for the UEs to send data to the InternetHost.
- Perform SGW failover procedure.

This experiment validates the performance of two GTP implementation methods. In the first method, which is proposed by [37], the controller is responsible for the configuration of the enhanced OpenFlow FDs. The controller configures the FD processing pipeline by installing flow-entries according to users' policies and profiles. To accomplish this task, the controller sends a sequence of signalling control messages that include Flow-Mod messages and TEID-table configuration messages (GTP-Tunnel-Setup-Request), while in the second method, the OpenFlow plugin of the FDs is enhanced to support extra features and functionality. These include the ability to intercept, reads a new message that added to the OpenFlow 1.3 protocol known as User-DataPlane-Setup-Request message (carries information about both the uplink and downlink tunnels). The OpenFlow plugin uses the information of the received message to configure the FD flow and TEID tables. For simplicity, these methods will henceforth be referred to as Method 1 and Method 2. This experiment is divided to two scenarios.

5.5.2.1 Bearer Setup Signalling Scenario

This scenario simulates a network with 100 UEs connected to OFB_Enb. These UEs starts an access bearer setup procedure to setup the GTP tunnels before sending the PING packets to the InternetHost. The simulation is repeated 10 times. In the first run, only 10 UEs are used and 20 UEs in the second run and so on till the last run where 100 UEs are used. In these simulation runs each UE used a single PING application to send a small message (40B every 1s) to the InternetHost. The "ini" configuration of the PING application is shown below.

```
#========== PING Apps Configuration ==============
**.numUe = $numUEs=10..100 step 10
**.ue1[*].numPingApps = 1
**.ue1[*].pingApp[*].stopTime = 100s
**.ue1[*].pingApp[*].destAddr = "InternetHost"
**.ue1[*].pingApp[*].packetSize=40B
**.ue1[*].pingApp[*].sendInterval = 1s
**.ue1[*].pingApp[0].startTime = uniform(0.1s,0.9s)
**.ue1[*].pingApp[*].stopTime = 99s
```

The first line tells OMNeT++ [69] to run the simulation 10 times with different number of UEs, the second line specify that each UE must has only one PING application, the other lines configure the PING app to pick a random time between 0.1s and 0.9s after the start of the simulation to start sending a PING packet with the size of 40B every 1s to the IP address of the InternetHost. The last line tells the application to stop sending the PING packet at time 99s which is only 1s before the end of the simulation. This scenario measured and analysed the number of control-signalling messages sent by the controller to setup the GTP tunnel to forward the PING messages from the serving eNodeB to the InternetHost. Figure 5.19 illustrates the measured number of control messages required to setup the end-to-end GTP tunnel between the eNodeB and PGW-D for the aforementioned methods.

Figure 5.19 presents the performance results related to the two methods described previously. As the figure shows, Method 1 leads to poor performance as this method requires more control messages to setup a single GTP tunnel and is highly likely to produce more signalling loads in the system. It makes perfect sense that the performance of the Method 2 is dominant because the advanced OpenFlow plugin of the FDs takes care of the processing pipeline configuration, which reduces the number of operations and control messages handled by the controller. In Method 2, the controller only sends a single User-DataPlane-Setup-Request message that includes information specified by the MME and SPGW applications to each FD in the selected path and the OpenFlow plugin handles the remaining operations.

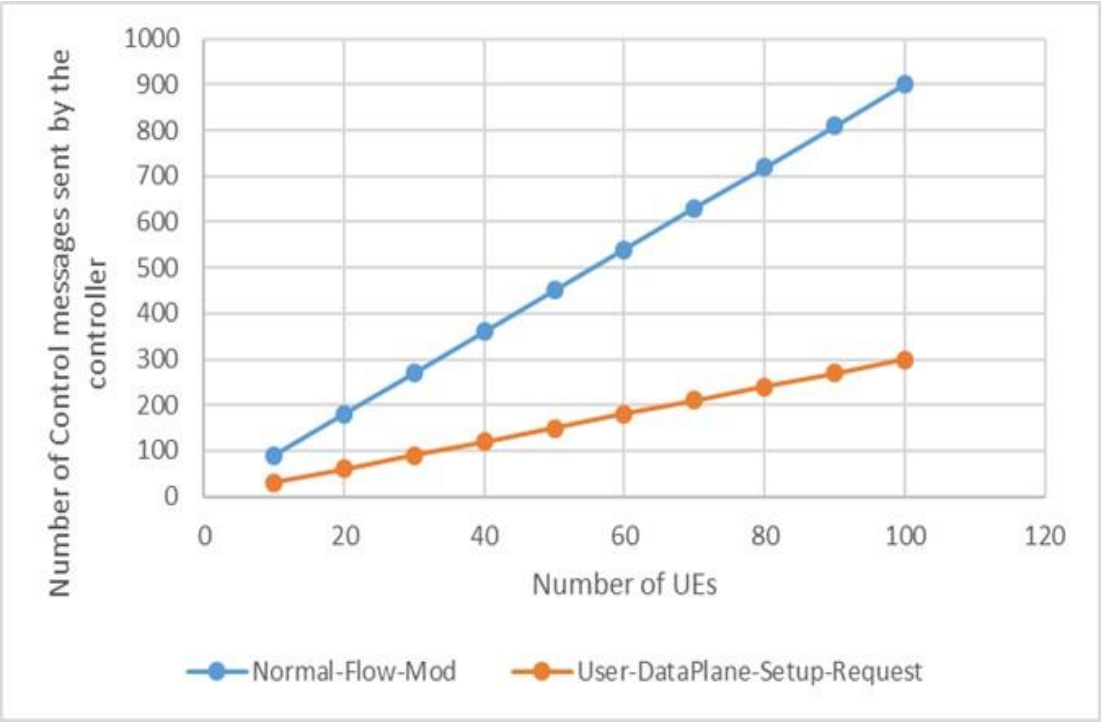


Figure 5.19: Number of Control Messages Sent by the Controller vs. Number of UEs Running a Single Application that Sends PING Messages to the InternetHost

5.5.2.2 Serving Gateway Failover Scenario

The aim of this simulation scenario is to evaluate the system performance during SGW failure. This scenario uses the same network setup and configuration of scenario 1 where UE traffic is distributed between SGW-D1 and SGW-D2. The scenario focuses on measuring the control-signalling sent by the controller to overcome the failure of the SGW-D1. Link failure is used to simulate the SGW-D1 failure because there is no direct and easy way to cause node failure in OMNeT++ [69]. To do that several parameters are added to the monitoring application of the SDN controller to specify the time to cause the link failure.

```
**.Controller.monitoringApplication.linkFirstEnd = "OFB_Enb"
**.Controller.monitoringApplication.linkSecondEnd = "SGW-D1"
**.Controller.monitoringApplication.linkDisableTime = 6 s
**.Controller.monitoringApplication.failOverIncluded = true
```

These lines are used to cause a link failure between OFB_Enb and SGW-D1 after a 6s from the start of the simulation time. More specifically, the first two lines specify the link that need to be disabled by the monitoring application. The third line specify the time of the failure.

In this case, 6s after the start of the simulation time. The last line makes the monitoring application notify the controllerOS module about the link failure. The monitoring application is statically configured to send a link failure notification after 0.5s from the time it causes the link failure, which mean the controllerOS will be notified at 6.5s. The monitoring application is programmed using C++ to reads these parameters either from the ”ini” file. If these parameters are not specified by the ini file, then the monitoring application will use the default values.

The default values disable this feature and makes the monitoring application works as normal without any extra task to perform. In both scenarios the spgwApp module is configured to use weighting factor to perform load distribution between the SGW-D1 and SGW-D2. 1:3 metric is used in the selection algorithm, which means that the spgwApp module assigns one UE to SGW-D1 and 3 to the SGW-D2 and repeat the process again. The ”ini” configuration to make the spgwApp module use the weighting factor is shown below.

```
**.Controller.spgwApp.selectionMode="WF"
**.Controller.spgwApp.keyNode="SGW_D1"
**.Controller.spgwApp.selectionMetrics="1:3"
```

This configuration tells the spgwApp to uses weighting factor for the load distribution, assign the first incoming request to SGW D1 and use 1:3 metric. In 3GPP SGW failover procedures previously explained in Section 5.3.2, the MME starts an access bearer release procedure and waits for the UE start a new access bearer procedure to assign them to a new functional SGW. Unlink 3GPP procedure, our procedure aims to provide faster recovery time without the need to release the access bearers and waits for the UEs request. This is done utilizing the SDN concept and the uses of centralized TEID allocation. This scenario compares and evaluates the performance of the two GTP implementation methods in term of the control message required to perform the tunnel redirection from one SGW-D to another. Figure 5.20 shows the number of control-signalling sent by the controller to transfer the UEs served by the SGW-D1 (failed SGW) to SGW-D2. The Restoration procedure includes installation of new flow-entries in the SGW-D2 and modifications of the TEID table of the PGW-D and OpenFlow capable eNodeB to change the virtual Port destination IP address and physical output port to forward UEs traffic to the new SGW-D.

Figure 5.20 shows the number of control messages that are needed to be sent by the controller to move different numbers of UEs from SGW-D1 to SGW-D2, as a response to SGW-D1 failure. Two things can be observed from the simulation outcome: First, the number of UEs served by the failed SGW is between 3 and 25, which makes since considering the 1:3 weighting factor metric. Second, the difference between the two implementation methods in terms of signalling loads is minimal. This result makes sense, because the number of control messages generated by the controller to change the forwarding behaviour from one SGW-D to another for each UE are 5 in the Method 1 and 3 in Method 2. Specifically, in Method 1, the controller sent 2 Flow-Mod messages and 1 GTP-Tunnel-Setup-Request to configure the processing pipeline and specify the virtual port parameters of the newly selected SGW-D, then it sent 1 GTP-Tunnel-Modify-Request to both the PGW-D and the serving eNodeB to send the traffic to the IP address of the new SGW-D, as the TEID will be the same. Conversely, in Method 2, a single User-DataPlane-Setup-Request message is sent by the controller to the newly selected SGW-D and the OpenFlow plugin of that devices handles the transformation of the received packet to flow-entries and specify the virtual port parameters. At the same time, a User-DataPlane-Modify-Request message is sent to both the PGW-D and the OFB_Enb to modify the TEID table and change the Next hop IP address of the tunnel to the IP address of SGW-D2 instead of SGW-D1.

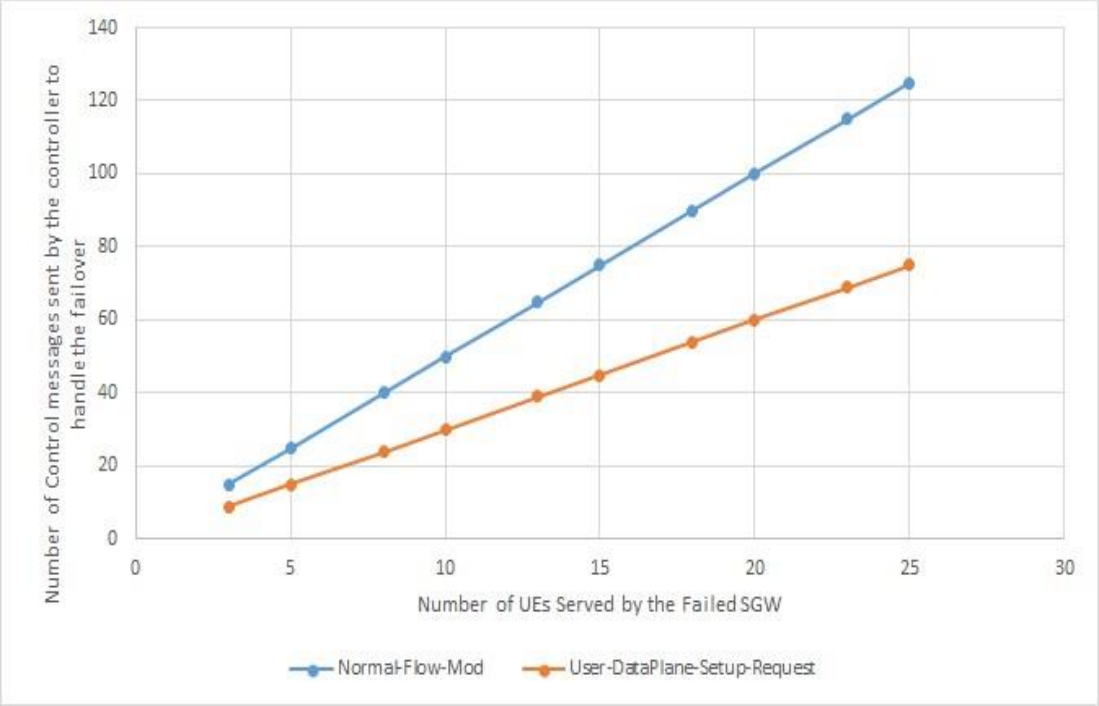


Figure 5.20: Number of Control Messages Sent by the Controller to Handle the SGW-D Failover with respect to the Number of UEs Served by the Failed SGW-D

5.5.3 SDN Agent and Load balancing Experiment

This experiment focuses on:

- Demonstrate the advantage of the SDN in providing better load distribution.
- Show the benefit of extending the OpenFlow FDs to include a local agent in term of reducing the signalling load and the centralized processing of packets.

Two scenarios are used to measure the required statistics. The first scenario, measured the control messages sent by the controller and the second scenario, measured the load distribution within the FDs of local SGW pool. More information about the scenarios step and the obtained result is presented in the next sub-sections. The simulation time of this experiment is also 100s with the same repetition number of Experiment 5.5.2.

5.5.3.1 Multiple sessions Bearer Setup Signalling Scenario

The simulated network of this scenario consists of number of UEs ranging between 10 and 100 connected to OFB_Enb. These UEs pick a random time after the start of the simulation to start the initial bearer setup procedure and after that each UE start two TCP session with the InternetHost. The ”ini” configuration of TCP sessions between the UEs and the InternetHost is shown below.

```
#Traffic between the Internet and the UE(InternetHost → Ue )
#Transmitter
*.ue1[*].tcpApp[0..1].typename="TCPSessionApp"
*.ue1[*].tcpApp[0].connectPort=1000+ancestorIndex(1)
*.ue1[*].tcpApp[1].connectPort=1000+( $ numUEs + ancestorIndex(1))
*.ue1[*].tcpApp[0..1].sendBytes=1GiB
*.ue1[*].tcpApp[0..1].active=true
*.ue1[*].tcpApp[0..1].tOpen=uniform(0.1s,0.9s)
*.ue1[*].tcpApp[0..1].tSend=uniform(1s,1.9s)
*.ue1[*].tcpApp[0..1].connectAddress="InternetHost"

#Receiver
*.InternetHost.tcpApp[*].typename="TCPSinkApp"
*.InternetHost.tcpApp[*].localPort=1000+ancestorIndex(0)

**.tcpApp[*].tClose=-1s
```

The above configuration is used to configure each UE to have two TCP applications. After the start of the simulation these applications pick a random number between 0.1s, and 0.9s to open TCP session with the InternetHost. Then it picks another random number between the 1s and 1.9s to start sending 1Gigabyte of data to the InternetHost.

Assuming that 10 UEs are used in the simulation and the InternetHost has the IP address 12.0.1.1, then the first application of UE1(0) will open session using the TCP socket [12.0.1.1:1000] and the second application will open session using the TCP socket [12.0.1.1:1011], this way the InternetHost will be able to differentiate between the TCP sessions coming from different UEs. In this scenario, the controller signalling messages to setup the initial access bearer is measured when:

- Native OpenFlow procedure is used to configure the FDs (Method 1).
- OpenFlow plugin of the FD is extended to handles mobile network specific operations (Method 2).
- FDs is equipped with a local agent (Method 3).

Considering that the focus of this scenario is the controller signalling messages, no load distribution is used and all the UEs traffic travel using the same path from the UEs to the InternetHost. More specifically, the UEs echo request packet is travelled through the air Interface to OFB_Enb to SGW-D to PGW-D to the InternetHost and return traffic are transferred using the same path in the reverse direction. Figure 5.21reports the signalling loads for the aforementioned cases when the number of connected UEs ranges from 10 to 100.

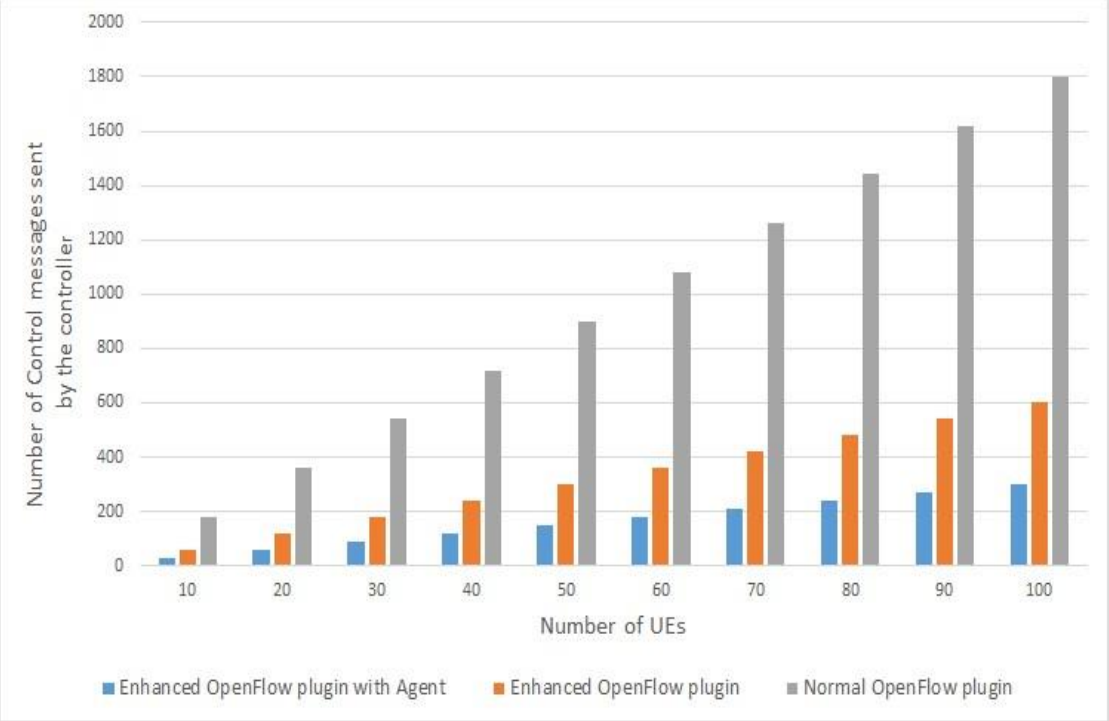


Figure 5.21: Number of Control Messages Sent by the Controller when the FDs have i) Normal OpenFlow plugin, ii) Enhanced OpenFlow plugin, iii) Enhanced OpenFlow plugin with Agent

Figure 5.21 shows the results obtained from experiment 2 scenario 1. The grey lines represent the control message sent by the network controller to the FDs to setup the access bearers for the UEs when the simulation network is configured to use Method 1, while the orange and blue lines represent the control messages when the simulation network is configured to use Method 2, and Method 3 respectively. The following can be observed from the results:

- Both Method 2, and Method 3 reduce the signalling loads and contributes to reducing the network congestion, as congestion may occur if too many data packets are needed to be sent to the controller to create a reactive rule.
- Both Method 2, and Method 3 extends OpenFlow protocol to uses specific messages that is built to handle a specific operation of the mobile network (User-DataPlane family messages that include setup/modify/delete messages) but still Method 3 outperform Method 2 because in Method 2 the first packet of any new flow needs to be sent to the controller for processing.
- Considering that the traffic generated by the UE applications falls under the same traffic class specified by the pre-defined traffic classes list. Method 3 provides the best system performance and adding the SDN agent to the FDs reduces the signalling load significantly. In this case, very few control messages need to be sent or received by the controller as shown in the blue lines.
- If the UE applications generate traffics the belong to a traffic class that is not included in the local agent list, then the FD will forward the first packet of the session to the controller for further processing and expect User-DataPlane-Setup-Request message as a response. In this case Method 2 and Method 3 will behave in the same way and fellow the same procedure.
- With two applications per UEs the signalling load required to setup the data plane forwarding path in Method 1 is very high even higher than the outcomes result of experiment 1 scenario 1.

5.5.3.2 Load-balancing Scenario

This scenario simulates a network of 100 UEs connected to an eNodeB. Two simulation suns with different load distribution methods are used in scenario. Weighting Factor is used in the first run and network load is used in the second run to distribute the downlink traffics between two different SGW-D devices. Namely, SGW-D1 and SGW-D2.

Network topology used in this scenario is shown in Figure 5.10. In this network each UEs is equipped with UDPVideoStreamCli application. This application is configured to pick a random time after the start of the simulation to send a VideoStream request message to the UDPVideoStreamSrv application, which is in the InternetHost node. The InternetHost is configured to run multiple UDPVideoStreamSrv applications that equal to the number of the UEs. The UDP port number is used to differentiate between the applications. Upon receiving the VideoStream request message, the InternetHost start sending the video stream to the UE in question. The ”ini” configuration used to specify the load distribution mechanism is shown below.

```
——————WeightingFactor————————-
**.Controller.spgwApp.selectionMode="WF"
——————NetworkLoad—————————-
**.Controller.spgwApp.selectionMode="LOAD"
```

The first line is used in the first run to distribute the network traffic between the SGW FDs using Weighting Factor and the second line is used in the second run to use the network load as the mechanism to distribute the traffic. Only one of these configurations can be used in a single run and if both lines are used, then the first one is used and the second is just ignored. Figure 5.22 shows a close approximation of how the system will handle load distribution when weighting factor and network loads are used as two mechanisms to select the SGW-D that should handle the UEs traffic. Considering that the scenario simulates downlink traffic from the InternetHost to the UEs. The SGW-D1 load is measured as the number of bits per second send over the link between the SGW-D1 and the eNodeB. The same method is used to measure the load of the SGW-D2. In this simulation scenario, the monitoring application is configured to get the FDs stats every 5 ms and the UEs requests are sent sequentially between simulation times of 1.0 s and 2.0 s. This simulation environment allows the load distribution application to Presley distribute the downlink traffic between the SGW FDs (SGW-D1 and SGW-D2).

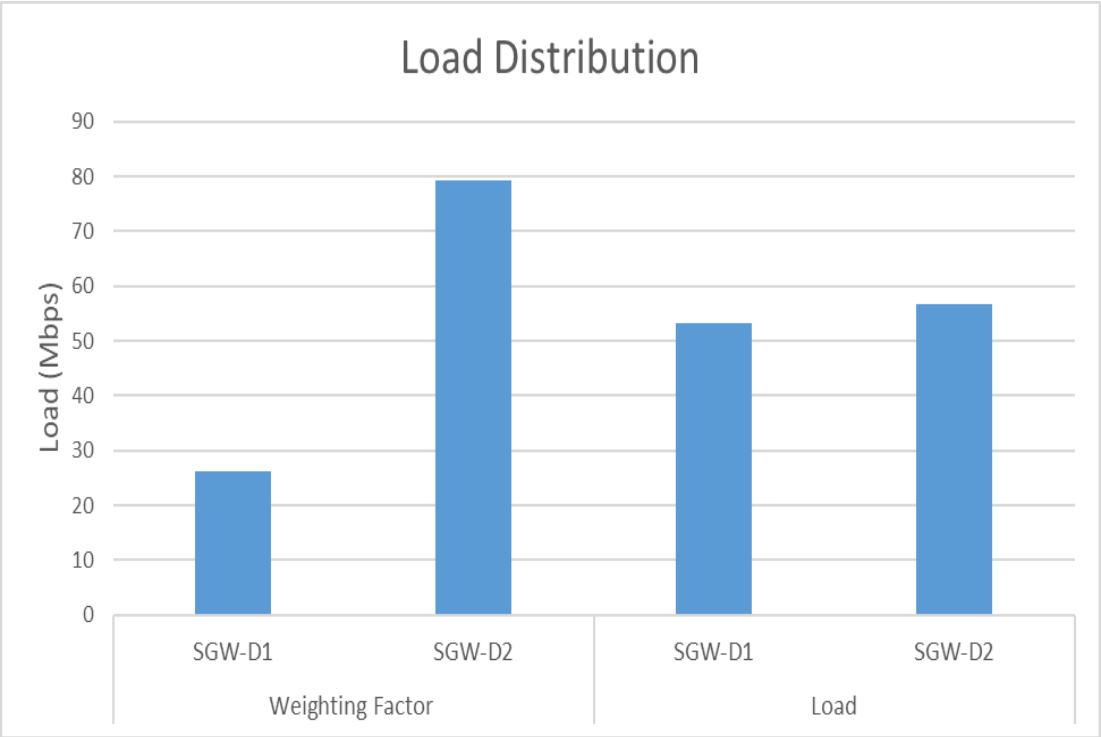


Figure 5.22: Load Distribution between the SGW-Ds in the Local Pool for both Weighting Factor and Network Load Selection Mechanism

The first insight that one can extract from Figure 5.22 is that the load handled by SGW-D2 is much greater than the load handled by SGW-D1 when weighting factor is used as a selection mechanism. In fact, this result is logical considering that a 1:3 metric is used in the selection algorithm. Changing the weighting metrics, to 1:2 or even 1:1, leads to a better load distribution. It is important to notice that a weighting-based algorithm follows a configured pattern without any knowledge about the network status, which may lead to unfair distribution of network loads, as shown in Figure 5.22. Conversely, load-based selection will always provide a fair distribution that aims to maximize resource utilization.

5.6 Summary

This chapter has presented Software-Based Mobile Core Network Architecture. Specifically, the architecture inherent SDN characteristic to provide an abstraction layer separating the control plane from the underlying data plane. Three procedures are described and analysed in detail, including Initial attachment and Initial bearer setup procedure, network resiliency, and load-balancing. The system performance when the FDs have local agent is measure and compare with two distinctive GTP tunnel

implementation methods. The results show that signalling loads are significantly reduced when the FD is equipped with a local agent. The experiments also show that, utilizing SDN introduces flexible and programmable aspects to the mobile core network to provide better load distribution and faster recovery time during network equipment failure. The system performance still can be enhanced in term of the data plane forwarding by removing the GTP tunnel. These approaches will be investigated in the next chapters.

CHAPTER 6

6 Software Defined Evolved Packet Core

6.1 Introduction

The current cellular network suffers from inflexible and expensive equipment, complex control plane protocols, and vendor specific configuration interfaces [32] [84]. The growth of mobile users' data traffic requires a significant re-design of the network's data and control plane infrastructures. The motivation is to simplify the mobile network deployment and operations by providing a programmable system that allows operators to deploy a new service or tune an existing one in a simple, flexible manner by separating the control plane and data plane.

Chapter 5 illustrates how the control plane and data plane separation can help enhance the network performance in terms of signalling load and better recovery time for some of the operations like SGW failure recovery, thanks to the centralized control management application used to provide centralized allocation of the TEIDs of the UEs GTP tunnels. Forwarding the UEs traffic in tunnel make it very difficult for the network operators to deploy services near to the UEs (in the backhaul network).

Therefore, the aim of this chapter is to demonstrate the ability of SDEPC to enhance the system performance in terms of end-to-end delay and packet loss. SDEPC utilizes SDN/OpenFlow technology to eliminate the need for GTP tunnelling [6][9], and uses path based MPLS tagging for the traffic forwarding and services differentiation. The SDEPC is built, tested and validated using an Open Source simulator (OMNeT++) by extending our work on an OpenFlow 1.3 model [82].

6.2 SDEPC Architecture

SDEPC architecture is shown in Figure 6.1, where the control plane is incorporated in a logically centralized SDN controller while the data plane forwarding is handled by the SDN transport network. The transport network consists of a set of core and ingress/egress nodes where the eNodeBs and the FDs connected to the PDN represent the ingress and egress nodes respectively and the rest are core nodes. Unlike the current EPC implementation where the PGW is responsible for IP address allocation, NAT, firewall and packet inspection [85]. In SDEPC the network functionality runs as a separate software application on top of the controller. This set-up contributes to providing an easier upgrade process of individual services without the need to upgrade the entire physical device. The MME and PCRF are replaced with software applications that provide similar functionality. eNodeB signalling messages have been restructured and formatted to be sent to the SDN controller using OpenFlow protocol.

The use of GTP-U tunnelling has been completely replaced by an SDN transport network to provide additional flexibility and optimal usage of the resources. MPLS tags have been used for the traffic forwarding through different paths from the eNodeBs to the PDN and the tags also carry the QoS identifier to provide service differentiation. Unlike LTE, which uses a centralized user plane network that requires all the traffic to traverse to the edge of the network where the PGW is usually located (even if some data traffic is going to/from local application servers), e.g. enterprise cloud servers, SDEPC helps in providing more efficient data plane forwarding. Simulation used to establish the actual improvement in system performance in terms of scalability, efficiency (bandwidth utilizations) and performance (end-to-end delay) using different network configurations. The proposed architecture can use NFV [17] as one, but not necessarily exclusively as the means of its implementation.

6.2.1 Architecture Description

This section briefly describes the basic building components of the SDEPC architecture. This includes the control components which represented by the SDN controller and data plane components represented by set of OpenFlow FDs and OpenFlow-enabled eNodeB. The next sub-sections describe these components in more detail.

6.2.1.1 SDN Controller

SDN Controller provides a programmatic platform for controlling the data plane FDs [3]. It has a set of supervisory applications, such as (i) network topology; (ii) network monitor; (iii) resource optimization apps, which are responsible for maintaining a global view of the network topology, network resources and distributing the loads across different links in the network.

The EPC network functions also run on top of the controller and contain multiple applications, such as (i) mobility manager; (ii) Network Address Translation (NAT); (iii) firewall; (iv) routing; (v) PCRF; (vi) the subscriber's profile. The latter maintains accurate information about the subscriber such as the user identification, address, and service subscription status; it also stores user subscribed QoS information, such as the maximum allowed bit rate or allowed traffic class. By ungrouping and moving network roles from individual network elements e.g. PGW, the SDEPC can reduce the overall latency by providing better routing.

The network topology application utilizes queries primitives and a graph database to maintain a global view of the network topology, while the monitor application maintains up-to-date network statistics and tracks topology changes. This data can be used by other applications to deliver their services, such as routing and resource optimization applications.

The monitoring application uses OpenFlow primitives such as OFP-Table-Stats-Request, OFP-Flow-Stats-Request, OFP-Port-Flow-Request, OFP-Queue-Stats-Request, OFP-Group-Stats-Request, and OFP-Meter-Stats-Request to acquire this information from each device. From the acquired information, the SDN controller builds detailed statistics about the type of traffic and the usage of each UE in order to apply charges and retain control over each UE based on the network operator's management policies and the subscriber's entitlement.

6.2.1.2 Forwarding Devices

This is represented by several OpenFlow switches, each consist of a flow, group, meter tables and the means to talk to the control plane that resides on a remote controller using the OpenFlow Protocol. A flow-table contains multiple flow entries; each flow-entry consists of:

- Matching rules that are composed of a set of ingress ports and L2/L3/L4 header fields, which may be variously wildcarded or masked.

- List of one or more instructions attached to each match rule.
- Counters for collecting statistics about the flows. A group table can be used for load balancing, fast failover, multicast/broadcast and to apply the same instruction to multiple flow entries, while the meter table is used to support QoS.

6.2.1.3 eNodeB

eNodeB represents the point of interaction between the access and core network in the SDEPC architecture. In the proposed architecture, the eNodeB keeps the same functionality and radio protocol stack as specified by 3GPP standards, while the S1 primitives are mapped to OpenFlow messages.

To handle UE authentication, authorization, and mobility management, new signalling messages have been adopted to be sent over the OpenFlow link between eNodeB and the EPC. These new messages have similar information to those specified by the 3GPP standards but are structured using OpenFlow. An SDN Agent is installed in each eNodeB and is responsible of informing the SDN controller about the radio resource allocation and its list of UEs that are receiving a better signal from the neighbour eNodeBs. This way the controller has a global view of the resource usage and the currently used handover and cell coverage parameters, in order to determine if it is necessary to implement load balancing in the access network.

Figure 6.1: SDEPC network

6.2.2 Architecture Operations

In SDEPC, the topology discovery application builds a graph database from the network nodes and links. The monitoring application tracks the network loads and utilization as well as the UE usage. It builds detailed statistics about the different nodes and UEs in the network. These statistics are used for charging and are also used by the routing application to specify the MPLS tags that represent a path between the ingress and egress nodes, allowing the controller to provide fair distribution of the network resource. The routing applications use Yen´s algorithm [86] to calculate multiple shortest paths between the ingress/egress nodes and assign an MPLS tag to each path. Unlike the conventional MPLS implementation where the label is locally significant, in the present implementation the controller assigns a single tag for the path from start node to the end node of the path. Then the controller conducts and installs OpenFlow rules in the entire core FDs to make a forwarding decision solely based on the received packet MPLS tag. The core nodes utilize the MPLS LABEL and MPLS Traffic Class (TC) fields to match the incoming traffic. The former is used to specify the output port leading to the next hop, while the latter is used to enforce complex QoS to provide a service prioritization between the received traffic by utilizing meter table together with per-port queues. The controller can either install these rules at the network initialization stage or when it receives the first packet from the edge nodes depending on the operator's requirements.

- **User attachment procedure**: starts with an Attach-Request being sent by the UE to the serving eNodeB. The latter sends an OFP-Initial-UE-Message that encapsulates the user Attach-Request to the controller, and upon receiving the message, after successful authentication, the controller replies to the serving eNodeB with the UE assigned IP address as shown in Figure 6.2.
- **Uplink traffic**: as Figure 6.2 shows, first the radio bearer is setup, then the UE sends the flow to its serving eNodeB. Upon receiving the first packet by the serving eNodeB. The latter realizes that it does not have a flow-entry programmed on its flow-tables to handle this traffic. Therefore, a Packet-In message is sent from the eNodeB to the controller requesting instructions to handle this type of traffic. The controller uses (i) the packet header information to identify the traffic type; (ii) the PCRF rule and user profile to determine the QoS Parameters. Then it determines the path that this packet and all upcoming

packets related to the same session should take through the network. The session type and network loads are considered in the process to determine the best path. The controller then installs a new flow-entry in the eNodeB to push UE identifier and Forwarding Path MPLS tags, and the QoS parameters for the session. The QoS parameter is a one-to-one mapping between the QoS Class Identifier (QCI) and the MPLS TC. All of the transport devices forward the traffic based on Forwarding Path MPLS tag using pre-installed flow-entry. Before the traffic reaches the egress node the Forwarding Path MPLS tag is popped and sent to the egress node. At the egress node, the UE identifier MPLS tag is popped; NAT/Port Address Translation (PAT) is performed, and the traffic is forwarded normally.

- **Local uplink traffic between two UEs**: when the controller receives the first packet of the session traffic, it will instruct the eNodeB to push the MPLS tag for the path between the source and destination eNodeB. Instead of tunnelling the data all the way to the edge of the network (PGW) utilizing the GTP tunnelling, simply to send it back to a UE in a neighbour eNodeB or even in the same eNodeB (used in current 3GPP standard), the controller in SDEPC offers network function abstraction to move some of the PGW functionality away from the network boundary and closer to the access network. This process reduces the unnecessary burden on the core network by providing a more efficient, less bandwidth consuming traffic forwarding mechanism that contributes to better latency for the UE-2-UE traffic.
- **Downlink procedure**: is similar to the uplink explained previously where the first packet of the traffic is sent to the controller, which checks if the traffic is destined to a UE in the ideal state. If yes, then it uses the EPC control plane application to specify the location of the UE, sending a paging message to the UE to change its state to connected and setup the radio bearer with the eNodeB. The controller uses the session information and the operator configuration to install a new OpenFlow rule to push an MPLS tag that specifies the path and the QoS level for the traffic to reach the serving eNodeB where the tag is popped and the normal OpenFlow lookup is performed to map the session to a radio bearer.

- **Handover procedure**: SDEPC provides the same handover process of a 3GPP LTE network with a slightly different way of handling the Path-Switch-Request by the controller to redirect the downlink traffic to the target eNodeB.

  In SDEPC the X2 interface is used only to exchange control messages between the source and target eNodeBs. Once a handover decision is made by the source eNodeB based on the UE measurement reports that inform it about the UE received signal strength/quality of the serving and neighbour eNodeBs, a Handover Request message is sent to the target eNodeB. The latter performs admission control and sends an OFP-Path-Switch-Request message to the controller. The controller updates the subscriber profile record to reflect the new location of the UE and sends an OFP-Flow-Mod-Message to the edge FD to tag the UE traffic with a new MPLS tag, which routes the traffic to the new serving eNodeB. Then the controller configures the flow-table the target eNodeB by sending OFP-Flow-Mod-Message. After that the target eNodeB sends a handover acknowledgement back to the source eNodeB that issues a handover command to the UE to detach from the source eNodeB and attempts to connect to the target eNodeB. The main differences between this procedure and the 3GPP-X2-handover are:

  - UE downlink traffic is immediately redirected to the target eNodeB that is responsible for buffering the data during the radio connection reconfiguration and synchronization. When the UE is successfully connected to the target eNodeB, the buffered data is forwarded to the UE.
  - no control messages are required to be sent by the target eNodeB to release the network resource at the source eNodeB because the resource is released after the timer associated with OpenFlow flow entries is expired [43].

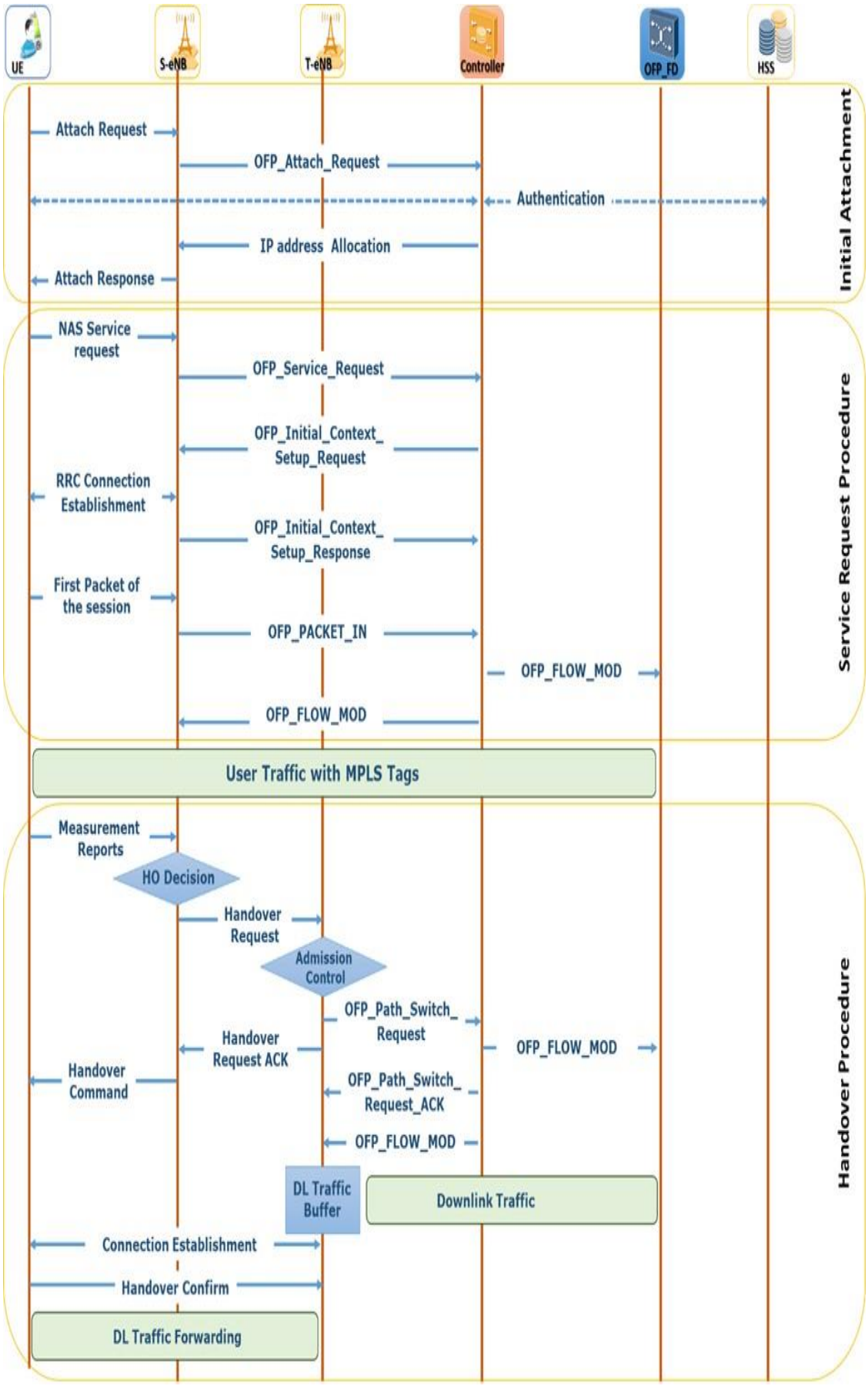


Figure 6.2: Simple Software Defined EPC Operations

6.3 Simulation Study of SDEPC Architecture

SDEPC architecture is implemented on OMNeT++ and modelled by a combination of OpenFlow 1.3 [82] and SimuLTE models. The user plane transport network is a set of FDs, while the UE and eNodeB nodes are imported from the SimuLTE [78]. To implement the SDEPC architecture in OMNeT++ it was necessary to address:

- SimuLTE module does not support control plane signalling messages.
- the restructure and encapsulation of the eNodeBs signalling messages in order for them to be sent using OpenFlow protocol to the controller. An extension is required to support the vast majority of the signalling messages between eNodeB and EPC. The proposed extension provides a simple conceptual model that represents a good basis for further extensions as well as the development of a complete tool for the future
- Introduce a few extensions to OpenFlow protocol to allow the mapping between the radio bearer and flow-entry in the flow-table

To address these issues, support for several control plane functionalities was added that are necessary in order to handle a UE session, establishment/teardown and intra-LTE handover which consists of:

- An ideal radio access control plane, achieved by exchanging data structures between the eNodeB and the UEs using the sendDirect() method without consuming link resources and errors.
- Addition of all of the control plane messages for UE session management in the EPC (for simplicity, NAS authentication messages were not implemented).

Also, the Initial-UE-Message, Initial-Context-Setup-Request, Initial-Context-Setup-Response, Path-Switch-Request, and Path-Switch-Request-Ack are restructured to be sent over OpenFlow link to the controller. To adapt these messages to OpenFlow protocol the OpenFlow protocol was extended to send/receive and handle the LTE related control messages and adopted a new set of messages that are related to UE operations. All of the new messages began with OFP-The-Name-OF-The-Message. Furthermore, a new action was added to the OpenFlow protocol to allow the controller to program the eNodeB to map multiple flow entries to a radio bearer. The OFP-Action-DRbearer struct provides a data structure for encoding a new action to map a flow-

entry to a radio bearer. It contains fields describing the radio bearer type, the data structure length, and the radio bearer identifier.

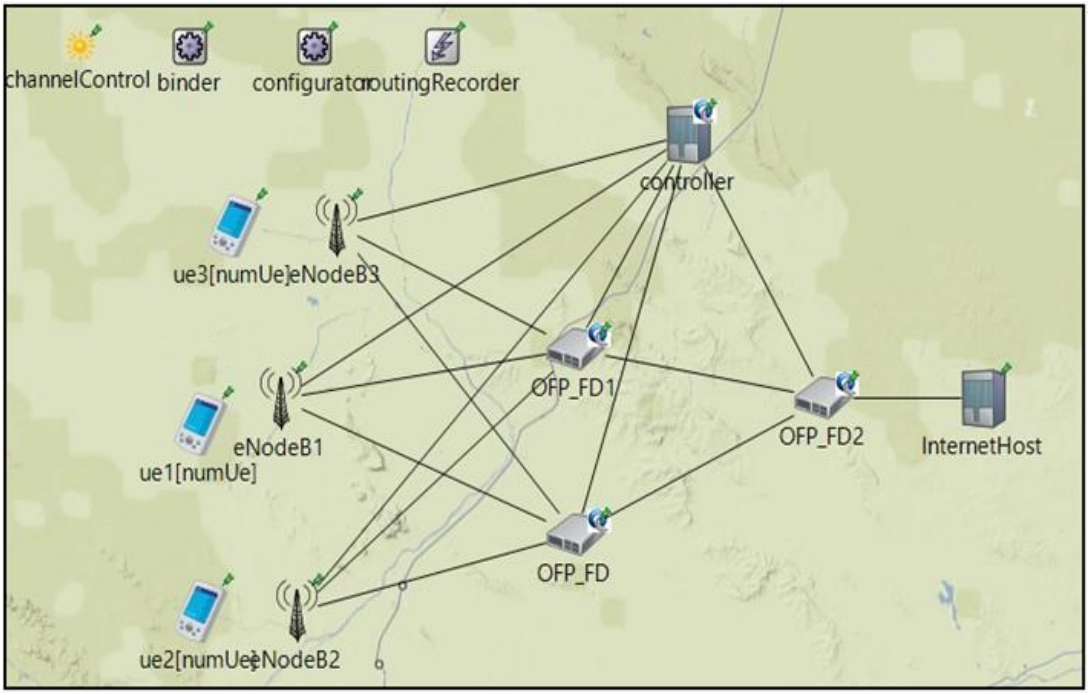


Figure 6.3: SDEPC OMNET++ Simulation Model

6.4 Simulation Results and Analysis

SDEPC has been simulated on a Dell Precision Tower equipped with an Intel Xeon E3-1246V3 / 3.5 GHz CPU with 8 GB of RAM, the Windows 7 Pro operating system, and OMNeT++ version 4.2.2, INET 2.3 with SimuLTE, and OpenFlow 1.3 extensions. Multiple experiments are used with different number of UEs and traffic types for both Uplink and Downlink. The simulated SDEPC network, shown in Figure 6.3, consists of: i) three eNodeBs; ii) several UEs uniformly distributed among cells with a few UEs sending/receiving traffic to/from the InternetHost that resides outside the SDEPC network; iii) SDN FDs that work as SGW/PGW data plane; iv) SDN controller. This is compared with the standard 3GPP LTE network that has the same number of eNodeBs and UEs. In our simulation experiments, eNodeBs employ MaxC/I scheduler and operates on 20MHz frequency band (100 Physical Resources Blocks available). The UEs moving linearly in a random direction with the speed of 1 meter per second (m/s) away from the eNodeB. The simulation lasted for 100s and replicated 5 times with different seed numbers to exclude simulation artifacts and to obtain mean and

confidence intervals over the different repeated simulations. The main simulation parameters are reported in Table 6.1.

Table 6.1: EUTRAN Configuration Parameters

Parameter	**Value**
Carrier Frequency	2 GHz
Bandwidth	20 MHz (100 RBs)
Mobility Model	Linear/Stationary
UE Speed	10/0 m/s
Path Loss	Urban Macro
eNodeB Tx Power	40 dBm
UE Tx Power	26 dBm
eNodeB Antenna Gain	18 dB
Noise Figure	5 dB
Cable Loss	2 dB

6.4.1 GTP Overhead Experiment

This experiment focuses on studying the effect of GTP overhead on the network utilization. In this simulation scenario, the UEs are connected to the access network ranging from 50 to 175 UEs distributed across 3 eNodeBs. The UDPBasicApp used as the traffic generator application, sends a packet every 5ms to each UE.

- INET 2.3 used in our simulation has several UDP applications to work as the UDP traffic generator or sink. This includes: UDPBasicApp, UDPSink, UDPEchoApp. The UDPBasicApp can work as a traffic generator or a sink, and when it is working like a generator, it sends UDP packets to the address of the destination node every given interval and expects the other end to run one of the following Applications UDPSink, UDPBasicApp, and UDPEchoApp. If the destination node is running one of the first two applications, then one-way traffic is simulated and a statistic like the end-toend delay is recorded by the receiver application (UDPSink, UDPEchoApp). Other statistics are also recorded by both the generator and sink application. These statistics includes the sending and receiving packets and packet loss etc. The UDPEchoApp works like the name suggest, it echoes the received packet back to the sender application. In this case two-way traffic is simulated and statistics like the RTT

is recorded with the other statistics that previously mentioned. destAddresses parameter of the UDPBasicApp model is the used to specify the way this model works. If it has an IP address or a name of the destination model, then the UDPBasicApp works as a traffic generator. Otherwise, the model works as a sink and discards the received packets after recording the statistics.

The simulation was run with two different packet sizes, 40B and 300B respectively. Figure 6.4 presents the measured link utilization related to both 3GPP-LTE and SDEPC.

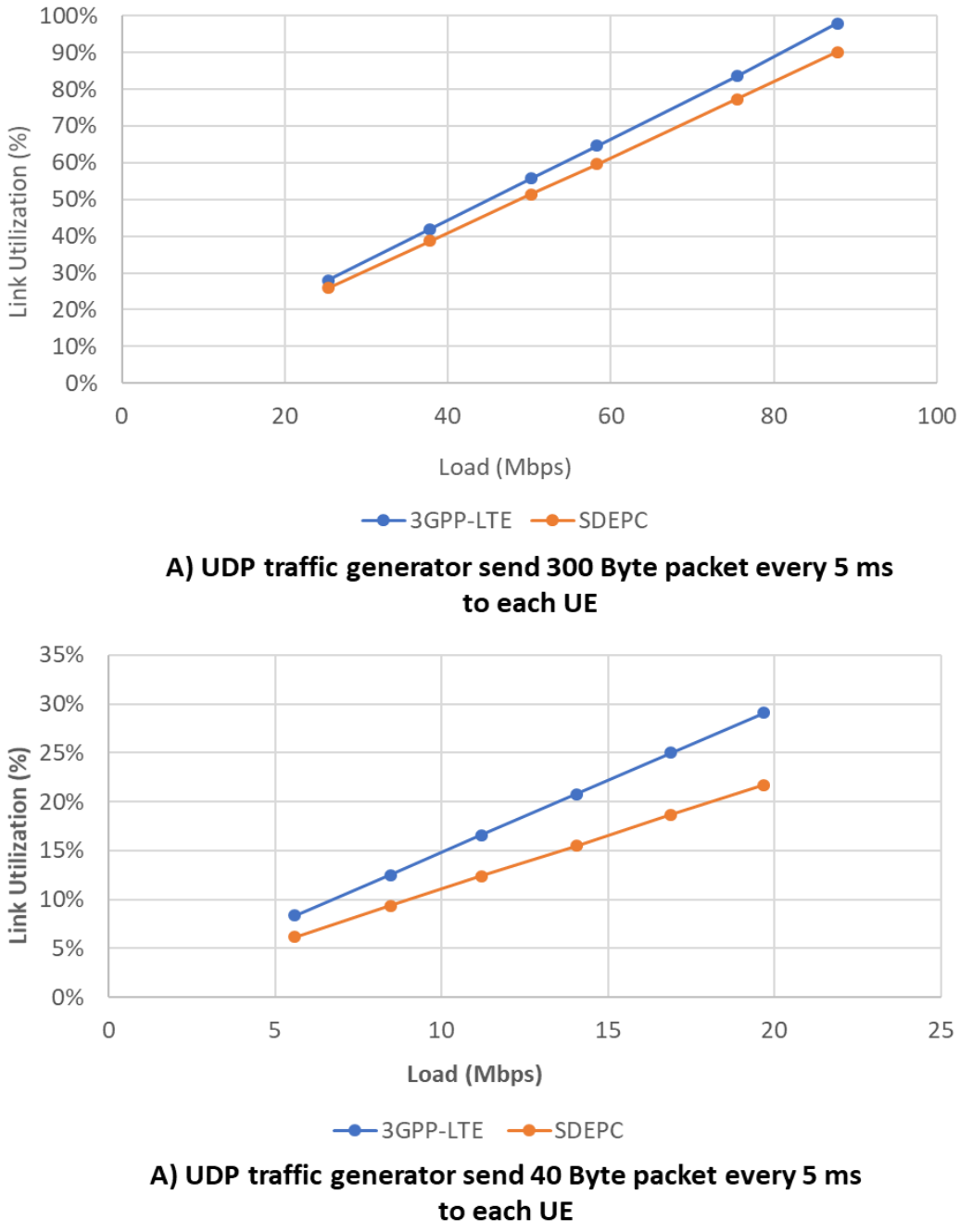


Figure 6.4: Load vs Link Utilization

Figure 6.4 shows that the SDEPC provides a better link utilization compared to the standard 3GPP-LTE. Moreover, the side effect of GTP encapsulation is more noticeable when the packet size is 40B. With 20Mbps loads, the overhead is about 9% when the packet size is 40B and less than 2% when the packet size is 300B. It worth noticing that with very large packet sizes, which is not considered in our simulation, IP fragmentation takes place and causes more overhead because of GTP encapsulation headers.

6.4.2 SDEPC Performance Evaluation Experiment

This experiment evaluates and analyses the impact of the extra headers added by GTP for each packet on the system performance by recording end-to-end delay, packet loss, and PGW queuing times for the different loads for both the SDEPC and the standard LTE networks. In this experiment, UDPVideoStream (VDoIP) is used as the traffic generator application.

The UDPVideoStream app is described in more detail in Chapter 5 Section 5.5.1.3. The Internet host uses UDPVideoStream Server application, which generates 1000 Byte packet every 5ms to each client request video stream. The load is changed by changing the number of UEs requesting the video stream from the InternetHost. It can be observed from the obtained results shown in Figure 6.5 that SDEPC provides a better performance than the 3GPP-LTE network. The average end-to-end delay, queuing time, and packet loss were used as metrics to validate the system's performance.

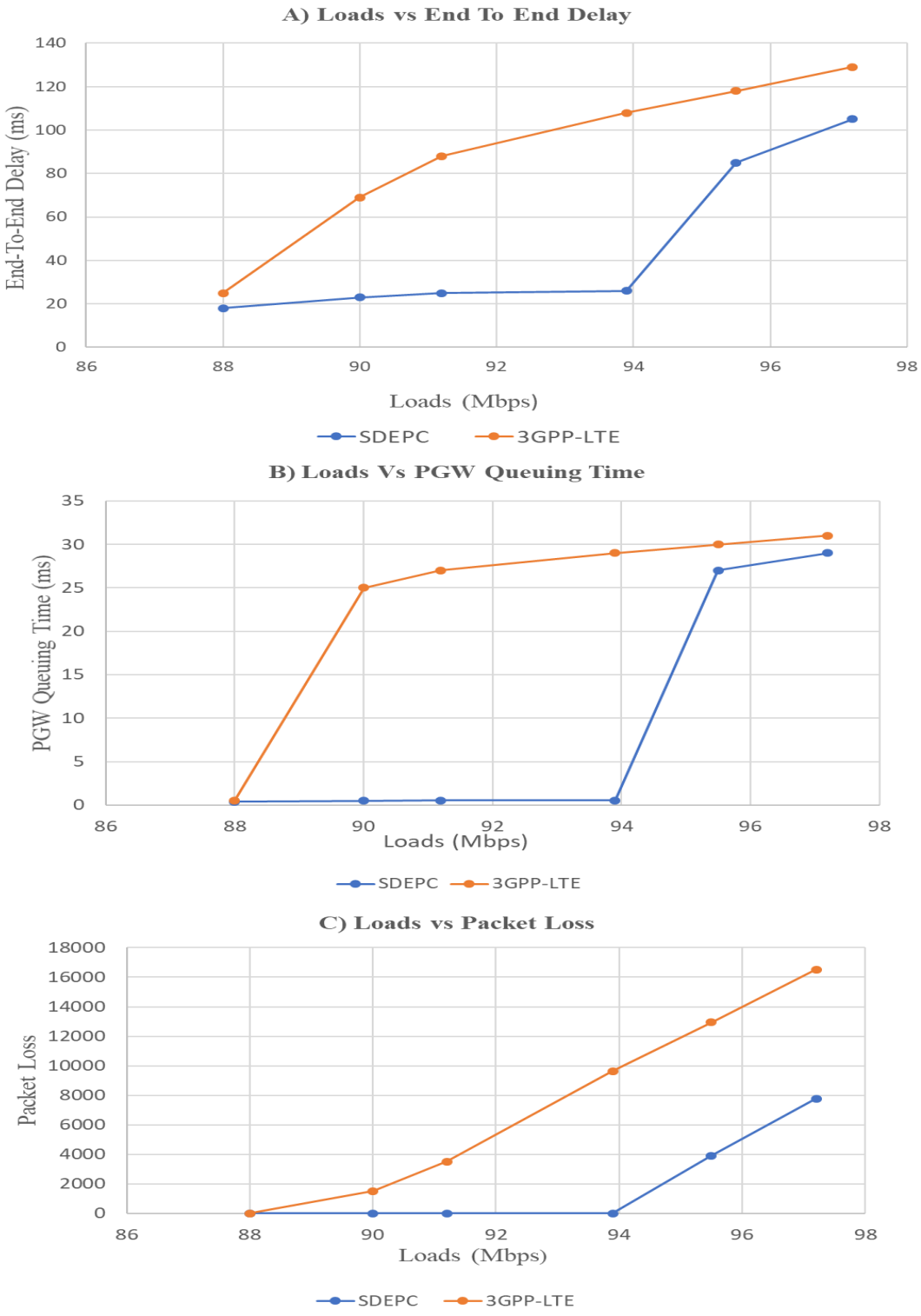


Figure 6.5: System performance comparison result

Figure 6.5-A illustrates the average delay experienced by the VDoIP packets during the simulations for the UEs. The results indicate that both systems provide a good performance with a minor advantage to the SDEPC until the load reached 90Mbps where the side effect of GTP tunnelling starts to appear, because the new headers added

by GTP (8-byte GTP, 8 byte UDP, and 20 byte IP headers) for each packet increase the loads above the interface threshold that increase the queuing time. This contributes towards a higher average end-to-end delay compared to the SDEPC where the overhead side effect appears only after the loads reached 95 Mbps.

Figure 6.5-B confirmed that 3GPP-LTE network queuing packets started when the loads reached 90 Mbps while SDEPC started after 95 Mbps which support the difference between the two systems in terms of end-to-end delay, as shown in Figure 6.5-A. Figure 6.5-C shows that the SDEPC started to drop packets only after the loads reached 95 Mbps, while the 3GPP-LTE started dropping packets when the loads reached 90 Mbps. In LTE, the packet loss ratio is 0.6% around 90 Mbps and continues to increase to reach 4% around 93.9 Mbps, while the packet loss in the SDEPC with the same loads was 0. After loads of 95.9 Mbps both LTE and SDEPC suffer from fairly high packet loss but SDEPC continues to provide better system performance in terms of packet loss ratio.

6.4.3 UE to UE Communication Experiment

This experiment focuses on measuring the system performance in the UE-to-UE case. Two tests are used in this experiment each with different traffic generator applications. In each test the simulation is run 5 times with different seed numbers and the average of the collected results is taken to reduce the simulation artifacts. In all tests the network topology is fixed and UEs are statically positioned in the simulation playground each near to its serving eNodeB to guarantee that both UEs have QCI 15 and the measured statistics are not affected by the wireless channel. In each test UE1(0) used a specific application to send traffic to the UE2(0).

- The first test uses PingApp as a traffic generator to send ICMP echo request from UE1(0) to UE2(0). The PingApp is configured to start sending traffic after 100ms of the simulation start time. The PingApp sends a packet with size 32 byte every 1s to the IP address of the UE2(0). In this run the RTT is recorded and measured.
- The second test uses simpleVoIPSender in the UE1(0) and simpleVoIPReceiver in UE2(0). The simpleVoIPSender application starts sending VoIP packet after 100ms of the simulation start time. The application sends a packet with the size of 40B every 20ms for the duration of weibull(1.423 s, 0.824 s), then changes its state to the silence and stops sending packets for weibull(0.899 s, 1.089 s).

In the simpleVoIPReceiver application both the end-to-end delay and Mean Opinion Score (MOS) are measured and recorded. MOS is a value between 1 (Poor) and 5 (Excellent), representing a human user's opinion of the voice quality. It is computed by using International Telecommunication Union (ITU) E-Module utilizing the packets end-to-end delay, jitter and packet loss ratio for the voice calls.

- Voice traffic is simulated in OMNeT++ by using VoIP application which consists of two models namely simpleVoIPSender and simpleVoIPRecevier, the former used as a traffic generator while the latter used as a sink. The simpleVoIPSender module generates constant bitrate traffic that supports talkspurts. The sender can be in either a talk or silence state. The talkspurtDuration and the silenceDuration parameters are used to specify the time spent by the sender in each state. In the first state the simpleVoIPSender model works like a CBR source and sends sequence of packets every configurable time interval to the simpleVoIPRecevier model at the destination node over UDP. The talkPacketSize and packetizationInterval parameters are used to specify the packet size and the sending interval. Packets generated and sent by this model do not have actual voice data. The senders change its state to silence without any explicit signalling after the talkspurtDuration is passed and stay in this state for the time specified by the silenceDuration parameter. In this state no packets are sent by the simpleVoIPSender model. The simpleVoIPRecevier works as a sink application for the traffic generated by the simpleVoIPSender model. When the simpleVoIPRecevier receives a packet, it records several statistics that includes the end-to-end delay and the MOS.

The same runs are performed with the standard EPS network (simuLTE [78]) using the same simulation configuration and the two results are compared and plotted to illustrate the performance of both systems.

Figure 6.6 illustrate the mean RTT comparison and Figure 6.7 shows time vector of the RTT of both systems. It is clear that the mean RTT of the SDEPC system is lower than the standard EPS system, because in EPS the ICMP echo request packet travels from the source (UE1(0)) through the serving eNodeB to the EPC network and back again to the service eNodeB of the destination (UE2(0)) till it reaches the destination. While in the SDEPC the centralized control plane allows a local break out to the traffic at the aggregation points to ensure the best route for the traffic.

Figure 6.7 shows a time vector for the RTT of each ICMP echo packet for both systems. It is very noticeable that the first and 51 echo packets have a higher RTT than the others. The reason for that, the first packet is sent to the controller based on a predefined rule that instructs the eNodeB to send traffic with destination address that matches the local IP addresses pool to the controller. Also, in our simulation all dynamic OpenFlow rules have a hard-timeout of 50s and because of that the forwarding rules are deleted, and the upcoming packet must go to the controller again to install a new rule to forward the traffic. This behaviour can be further enhanced by deploying a better mechanism to specify the rules hard and idle timeouts.

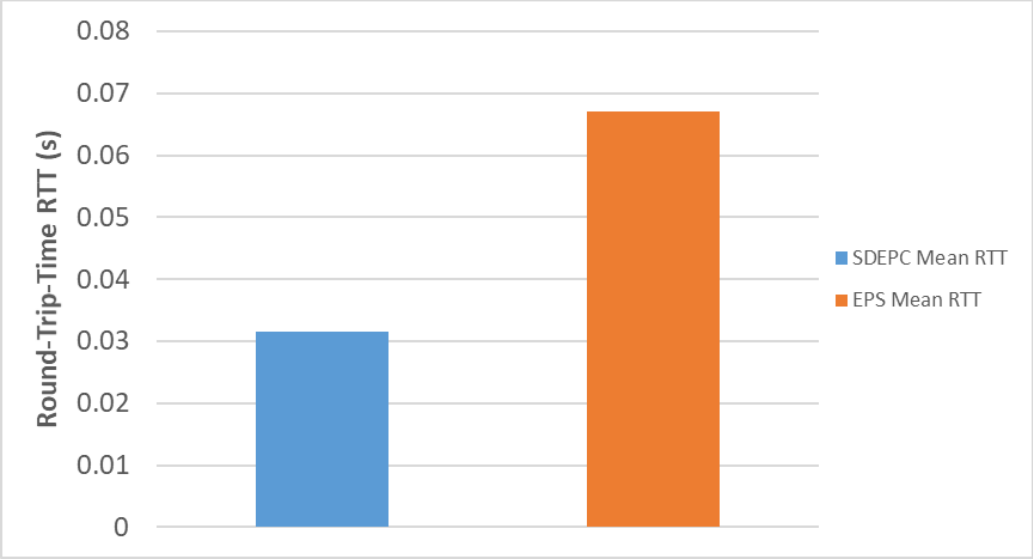


Figure 6.6: EPS vs SDEPC Mean RTT

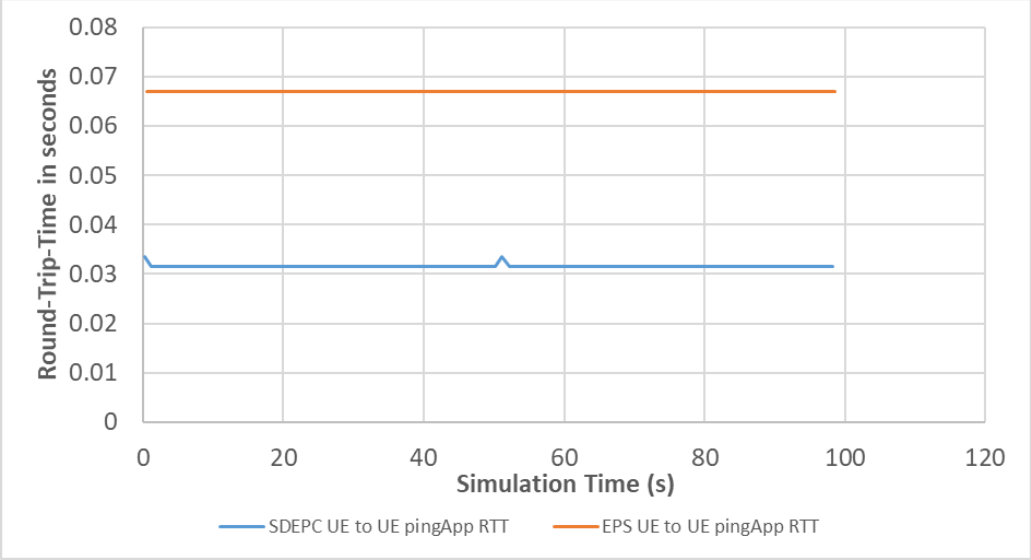


Figure 6.7: EPS vs SDEPC RTT Vectors

The same network with the same configuration is used in the second test, only the application is changed to VoIP. In this test, end-to-end delay together with the MOS are used as the metric to evaluate the system performance. Also, this test measures the effect of the slightly long processing time of the first packet on the user experience during the voice call. Figure 6.8 shows bar charts of the recorded end-to-end delay of both systems. This chart plots the minimum, maximum and the average end-to-end delay.

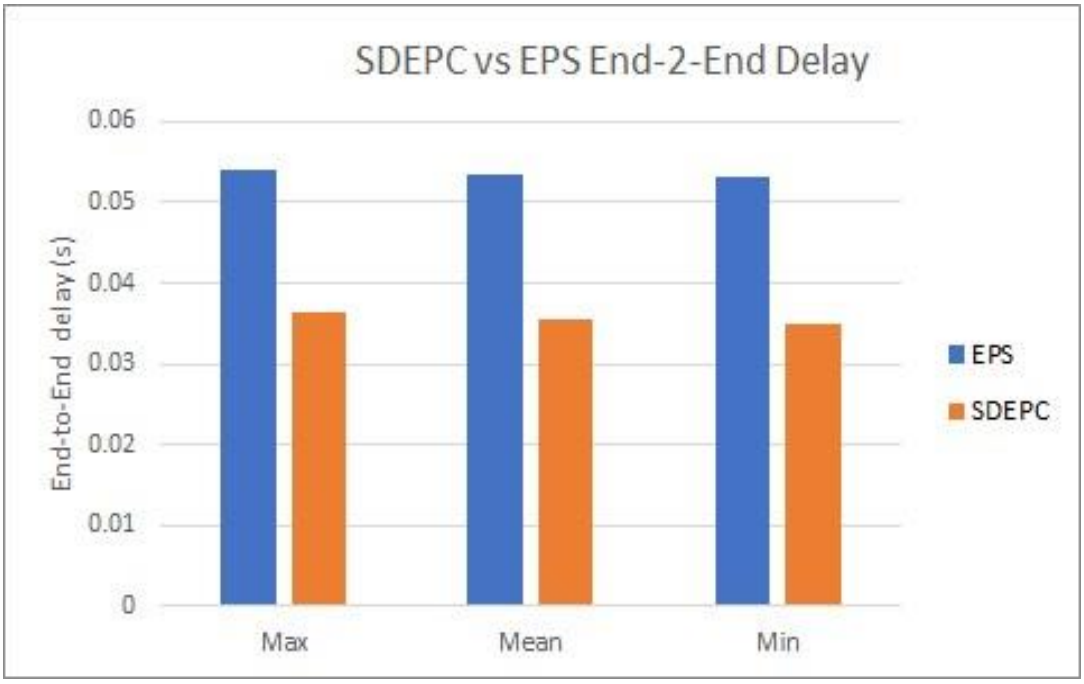


Figure 6.8: VoIP Traffic - end-to-end Delay comparison of EPS and SDEPC

The results obtained from this test show the same behaviour of the PING test and SDEPC outperforms the standard EPS because of the optimization in the UE-to-UE traffic routing offered by the centralized control plane. Also, time vector for the end-to-end delay and MOS of the VoIP traffic throughout the simulation time is measured and presented in Figure 6.9, while Figure 6.10 shows the time vectors of measured MOS. The obtained results are used to evaluate the effects of the first packet and the packet after deleting the forwarding rule from the switches due to the hard timeout timer.

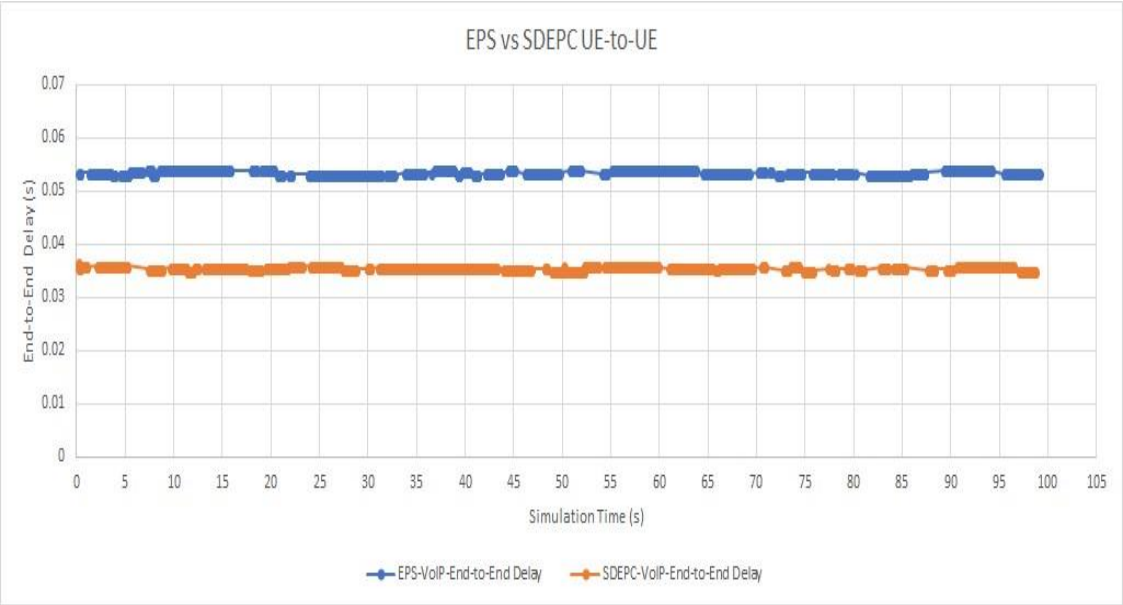


Figure 6.9: VoIP Traffic - end-to-end Delay Time Vectors

The end-to-end delay time vectors of both systems show that the time duration of the talk and silence states is different. At the same time, it is noticeable that in the SDEPC system the first packet and the packet after 50s simulation have a slightly higher delay time but with minimum impact on the overall end-to-end delay of the system. To further investigate if this delay causes a problem to the user experience during this call, MOS is captured and recorded as a time vector during the whole simulation time.

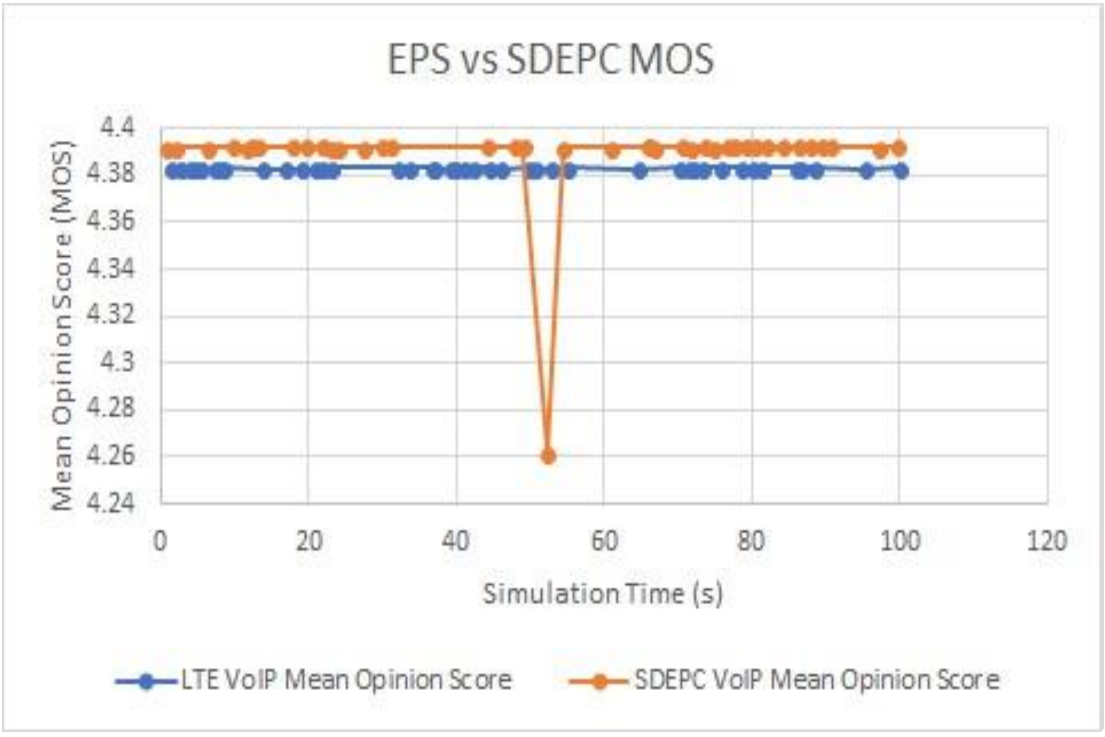


Figure 6.10: VoIP Traffic - EPS vs SDEPC Mean Opinion Score

It can be observed that both systems provide a very good MOS throughout the simulation time, this is because both systems are not loaded and only two UEs are simulated. The SDEPC score a slightly higher value than the standard LTE. The Mean Opinion Score value is calculated using the E-Model that considers packet loss and delay together with jitter in its computation. In this test both systems do not have any packet losses or concerning jitter. The packet delay of both systems drives the MOS outcome. Therefore, the SDEPC score a higher MOS value than the standard LTE because the measured end-to-end delay is better as shown in Figure 6.9. Moreover, deleting the forwarding rules at time 50s of the simulation changes the measured MOS from 4.39 to 4.26 but this change does not affect the user experience at that moment since both values are within the rage of very good experience. Furthermore, the extra delay of the first packet has no noticeable effect of the reported MOS considering the measured MOS for the first packet is exactly 4.39169 while the MOS of the flow upcoming VoIP traffic is ranged between 4.39196 and 4.39243.

6.5 Advantages and Disadvantages

This presented SDEPC architecture utilizes SDN to increase network programmability and flexibility. SDEPC eliminates GTP tunnelling which helps reducing the overhead of the extra headers added by the GTP for each packet and provides better routing without the need to tunnel all the traffic to the PGW to forward the traffic. Using SDN with MPLS path-based traffic forwarding boosts system performance in terms of end-to-end delay and bandwidth utilization.

Nevertheless, the main benefits are obtained from new services such as offloading of selective traffic to a distributed cloud solution and better in-network cashing solutions. At the same time, SDEPC operates without GTP, which reduces the system interoperability with legacy networks and requires an innovative solution to provide some sort of backward compatibility. Therefore, the next section describes the modifications required to SDEPC architecture to operate with and without GTP, which guarantees the backward compatibility with legacy networks.

6.6 Backward Compatibility Support

The network topology is shown in Figure 6.13. It has several differences from the network explained in Section 6.2.1. This includes: i) heterogenous backhaul network;

ii) standalone MME device; iii) modified OpenFlow FDs to work as SGW-D and PGW-D.

- **backhaul network**: consists of two sub-networks; the first sub-network is a normal traditional network while the second is represented by set of OpenFlow switches distributed on the boundary of the first sub-network as shown in Figure 6.13. OpenFlow switches are used after the mobile core and access networks, and these switches are configured by the SDN controller to route the received traffic through the normal network to the switch on the other end. The OpenFlow FDs of the backhaul network is implemented to support two operating modes. In first operating mode, OpenFlow FD utilizes its normal implementation that always send the first packet of each flow to the controller, while in the second operating mode, each FD is equipped with a local agent. The Agent is used to offload some of the controller operations to data plane nodes and reduce the signalling messages between the controller and data plane nodes. The agent is configured once and only update when the controller thinks it is better to alter the network forwarding behaviour. This approach helps reduce the number of packets that needs to be sent to the controller and reduce the processing time of the first packet. Section 6.6.3.2 compare the performance of these operating modes in term of the delay of the first packet and the effect of the distance between the controller and the OpenFlow FDs.
- **MME entity**: keeps its S1, S11 and S10 interfaces to interact with the legacy network, at the same time it can interact with the SDN controller using RestFul API. As previously mentioned, the path selection is done by the SDN controller through a specialized application that considers the current network loads before specifying the best path to handle the traffic.
- **SGW-D and PGW-D**: is implemented by utilizing a modified version of OpenFlow protocol. The next sub-section explains in detail the structure and the design of the OMNeT++ model of these data plane devices.

6.6.1 SGW and PGW Data Plane Forwarding Device

In OMNeT++ nodes are simulated by a combination of modules that are connected together by means of links. Each module consists of structure and behaviour parts. The former is implemented in ned file while the latter is implemented in C++. modules communicating with each other by the exchange of messages. Figure 6.11 shows the

structure of the FD. Each internal module is associated with a C++ class that represents the behaviour of that particular module. All the modules in OMNeT++ are implemented as an extension to the cSimpleModule. The latter represents an abstract class with methods to initialize and finalize the module, it also has method (handleMessage) to handle the messages received by the module. Any module built as extension to the cSimpleModule must haves at least an overwrite implementation of the Initialization and handleMessage methods. Normally the Initialization method initialize the module by reading the module configuration, also in the initialization access to the node tables is obtained if necessary. The handleMessage implements the way to handle the received message by each individual module.

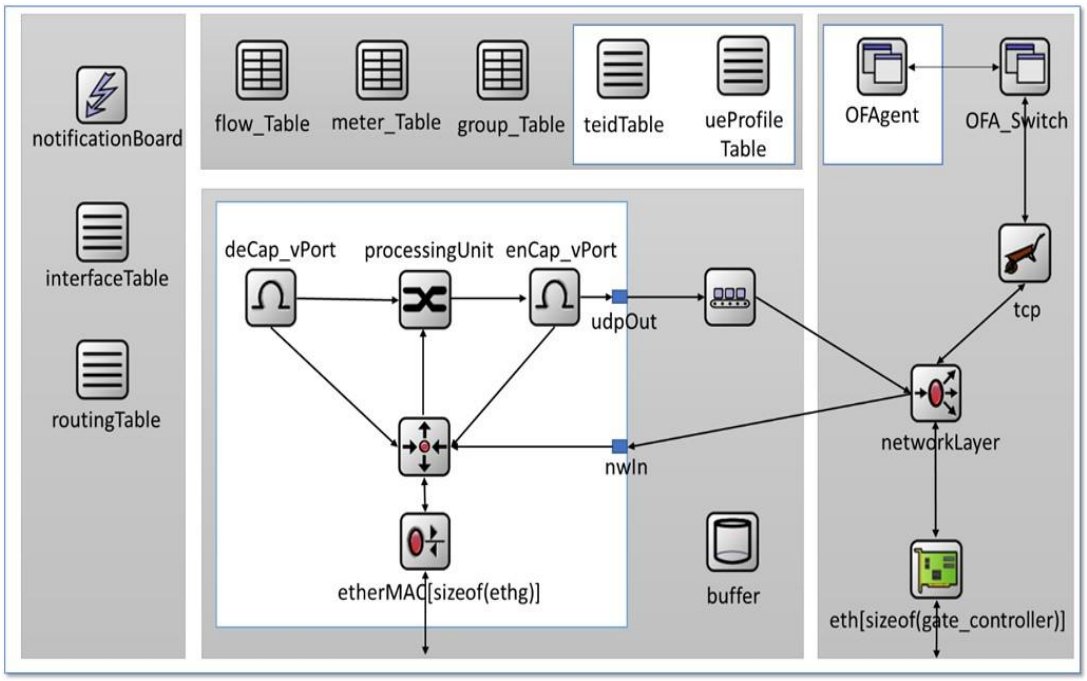


Figure 6.11: SGW and PGW Data plane FD module

The UML representation of the FD module is shown in Figure 6.12. This section briefly describes the relationship and the interaction between the classes. The class Entry is an abstract class that implements do-nothing methods and it is required from all the sub-classes to re-implement these methods to redefine the method´s operation. FlowEntry, MeterEntry, GroupEntry, and TeidEntry are implemented as sub-classes of class Entry. A simplified view of attributes and methods is used due to the space limitation. Below a brief description of these sub-classes.

- **FlowEntry class**: has one to one relationship with the FlowStats class. The later includes the flow matching fields, the InstructionSet with a basic statistic about

the number of packet and bytes handled by the FlowEntry and it includes the duration of the entry as well.

- **GroupEntry class**: has one or more OFBBucket. The later contains a bucketStats and action list. The structure of the action list class is omitted in Figure 6.12 but is fully explained in Chapter 3 Section 3.4.2.1B. The bucketStats class maintains statistics about the number of packets and bytes handled by each bucket.
- **MeterEntry**: is designed by following the same philosophy. As shown in Figure 6.12, each MeterEntry consists of one or more meter-bands, and each meter-band has meter statistics.
- **TeidEntry**: is the last sub-class of the Entry class. It has several parameters that represent the GTP tunnel configuration.

In the same way FlowTable, GroupTable, MeterTable, and TeidTable are implemented as subclasses of the abstract Table class. The later includes an empty implementation to Add, Modify, Delete, and lookup an entry in the table, where each sub-class implements its version of these methods. The structure, operation, and relationship of each one of these subclasses is described below.

- **FlowTable class**: has FlowEntryList that consist of one or more FlowEntry. It also performs basic operations to add, modify, lookup, and delete FlowEntry from the FlowEntryList. The class can be accessed by other classes such as the OFA_Switch, OFAgent, and the ProcessingUnit.
- **GroupTable and MeterTable**: have the same behaviour described in the FlowTable, in a sense that the GroupTable has a GroupEntryList that consists of one or more GroupEntry. The same with the MeterTable, it has a MeterEntryList that consists of one or more MeterEntry. Both classes implement their version of the Table operations methods and accessed by OFA_Switch, OFAgent, and the ProcessingUnit.
- **TeidTable**: has a TeidEntryList that consists of one or more TeidEntry, normally configured by the OFA Switch and used by the enCap-vPort class to lookup if the packet needs to be sent through a GTP tunnel.
- **ueProfileTable**: has ueProifleList that consists of one or more ueProfile entries. This class is accessed and configured only by the OFAgnet.

The most important behaviours of the FDs are performed by the OFA_Switch, ProcessingUnit, and the OFAgent classes. The OFA_Switch plays the role of a

simplified OpenFlow plugin. It is mainly responsible for the interaction with the SDN controller and is based on the controller instruction OFA_Switch to configure the FDs tables; it also produces and send statistic messages to the controller upon request. The processingUnit fills the role of the fast data plane processing entity. It basically forwards the received packet based on the controller instructions specified by the FD tables. Differently the OFAgent is involved in both data and control plane operations. The control plane refers to the interaction with the controller and the configuration of the FD tables. In this switch design, the OFAgent module works like an addon that can be turn on and off by using the FD parameters.

6.6.1.1 Operating Mode

The SGW and PGW data plane FDs are implemented to support three different operating modes. The first one is to work in reactive mode while the second is to work in proactive mode and the last one is to work in semi-reactive mode. The OFAgent module is turned off in the first two operating modes and only used in the semi-reactive mode. This section briefly describes the FD process the incoming traffic in each operation mode.

- **Reactive Mode**: in this operating mode, when a frame is received the processingUnit class handles the packet by performing a lookup on the flowTable and if a match is not found, then the received frame is inserted in the buffer and NO-MATCH-FOUND signal is emitted by the processingUnit. The OFA_Switch class registers itself as a listener to this type of signals and upon receiving the signal, the frame is obtained from the buffer and a Packet-In message is sent to the controller. The message may contain the whole frame or just headers information depending on the sendCompleteFrame parameter specified by structure of the OFA_Switch module. In our test, this parameter is set to false and only the frame headers are sent to the controller. The frame is kept in the buffer while the OFA_Switch is waiting for the controller´s response. Upon receiving the controller instructions through a Flow-Mod-Message, the OFA_Switch creates a new FlowEntry and configures the FlowTable to add this new Entry. At the same time, the OFA_Switch module sends a FRAME PROCESS signal to the processingUnit to process the buffer frame. In this operating mode, the FD only asks for instructions when it receives a frame and does not know how to handle it.

- **Proactive Mode**: in this mode, the FD flowTable is configured to handle the traffic even before receiving it. As previously mentioned, this mode reduces the handling time of the first packet of each flow, but at the same time introduces extra entries in the flowTable even when there is no traffic. One way to optimize this behaviour is by utilizing an algorithm to predict the behaviour of different traffic classes to assign a hard_time out to remove the flow-entry after a specific time has elapsed. This solution may introduce high signalling load if the algorithm did not correctly predict the right time to remove the flow-entry. If the flow-entry is removed while the actual traffic is still coming to the FD, the procedure previously explained in the reactive mode needs to be performed to acquire a new instruction from the controller.
- **Semi-Reactive Mode**: in this mode, the OFAgent module is turned on and used to keep the controller instructions provided during the UE initial attachment procedure. In the initial attachment procedure, the controller sends a Ue-profile-mod-message to the selected FD. The message is first received by the OFA_Switch, which simply forwards the message to the OFAgent. The latter creates a new ueProfile that includes the UE information and a list of traffic classes with the way to handle them. Then it configures the ueProfileTable to add the new ueProfile to the table. At the same time, if the serving eNodeB is a standard eNodeB, then the controller also sends a GTP-Tunnel-Setup-Request-Message to the SGW-FD to provide the FD with the capability to setup an on-demand GTP tunnel with the serving eNodeB. Similarly, for the ue-Profile-Mod-Message, the message is first received by the OFA_Switch. Upon receiving the message, the OFA_Switch creates a new TeidEntry that contains all the GTP tunnel information and configures the TEID table to add the entry to the table. At the same time the controller notifies the MME about the SGWFD GTP tunnel configuration. The MME then sends the information to the serving eNodeB in the normal S1AP initial-context-setup-request-message. Upon receiving the first packet of a new flow, the processingUnit will not find a match in the flowTable because the switch is working in a semi-reactive mode. NO-MATCH-FOUND signal is then emitted by the processingUnit, but unlike the normal implementation of the FD, the OFAgent is the one responsible about this signal not the OFA_Switch module.

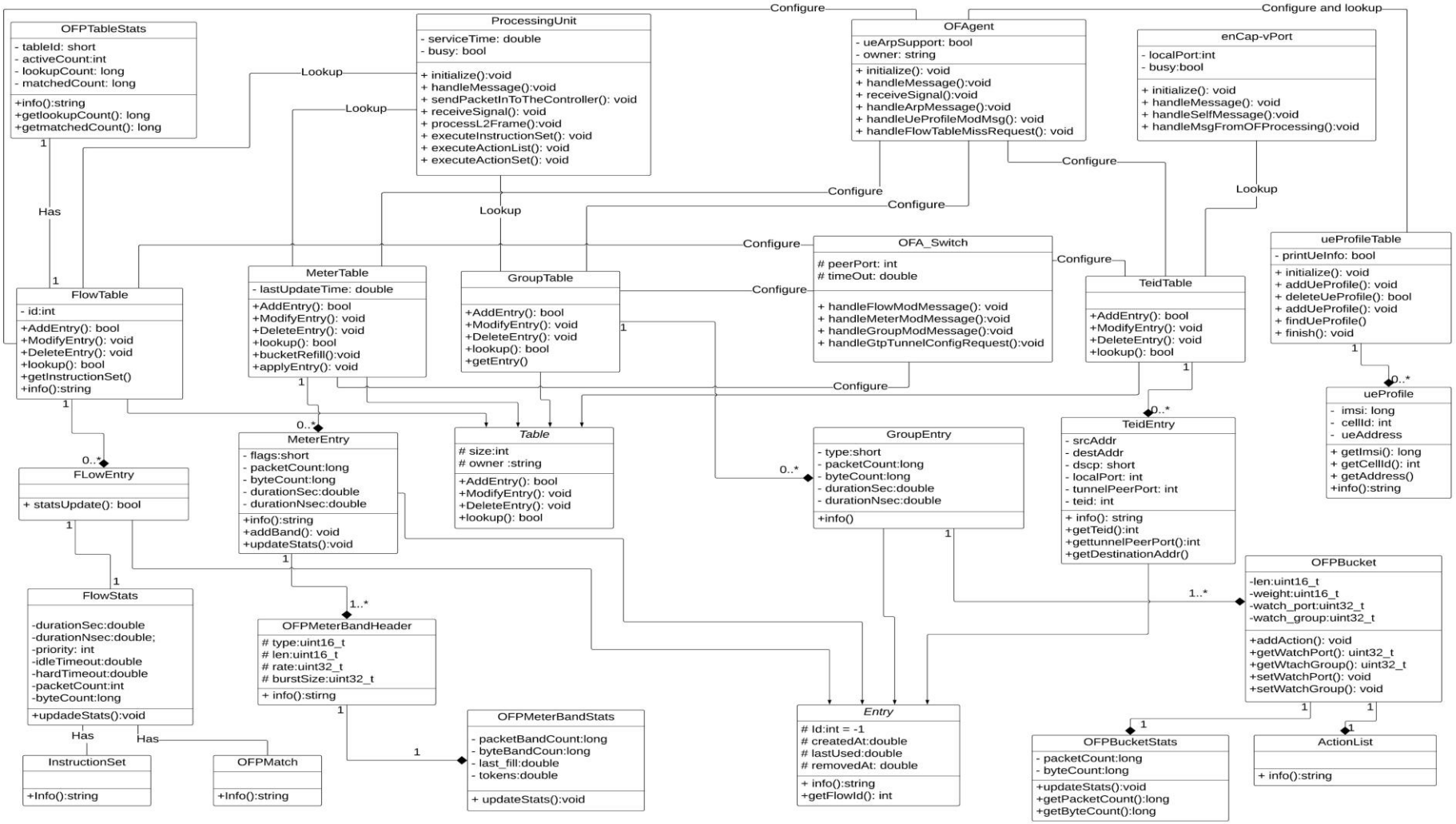


Figure 6.12: UML representation of the SGW and PGW FD Modul

6.6.2 Simulation Network

The logical topology of the network used in our scenarios is shown in Figure 6.13. It mainly consists of several building blocks, that includes: i) one or more UEs; ii) two eNodeBs, the first one represents the standard LTE eNodeB while the second one is OpenFlow enabled eNodeB; iii) MME; iv) SDN controller; v) set of OpenFlow FDs; vi) InternetHost that includes an application that works as a traffic generator or sink.

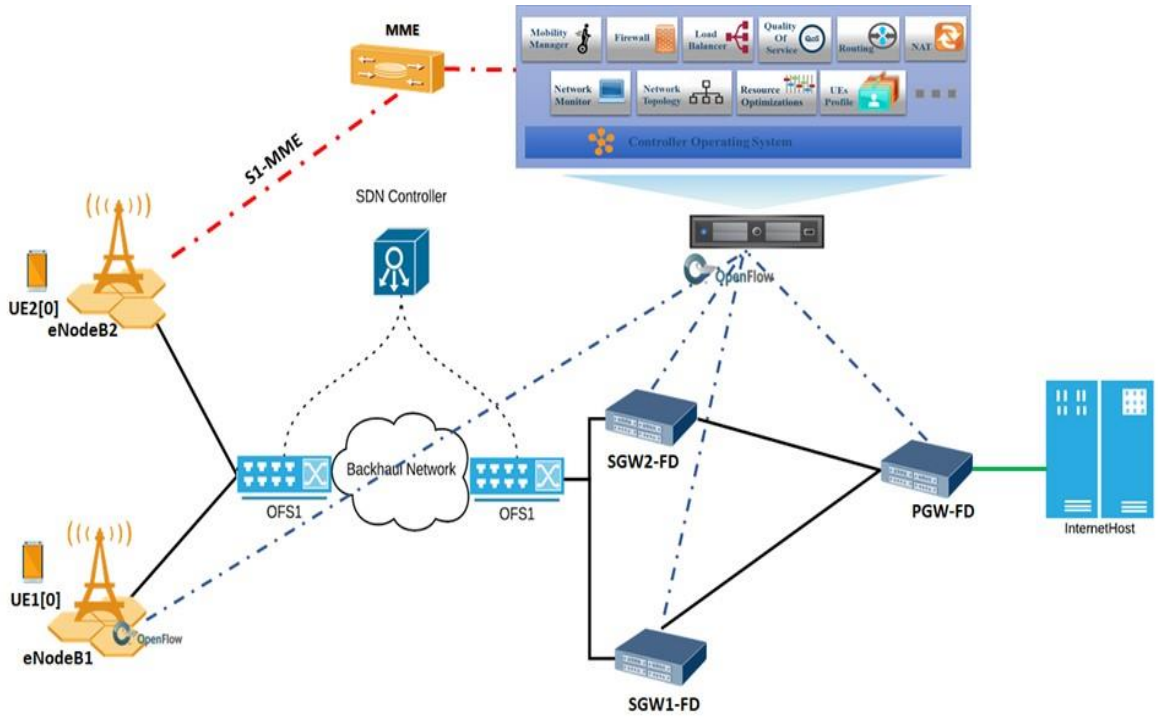


Figure 6.13: Logical Topology of the simulated network in OMNeT++

In our simulations, all the nodes are statically positioned in the simulation playground except for the UEs. The location, speed and movement direction are changed based on the experiment. As shown in Figure 6.13, UEs are connected to the eNodeB by means of the air interface, eNodeB1 is connected to the MME through the S1-MME over the SCTP connection, while eNodeB2 is connected to the SDN controller through the OpenFlow channel over TCP connection. The same as eNodeB2 all the FDs are configured to connect to the SDN controller using OpenFlow protocol. Finally, the MME is connected to the SDN controller through a simple connection that simulates the northbound interface.

In this simulation, the serving eNodeB of each UE is statically configured using the ini file. In the initialization step of the simulation, each UE reads the macCellId, and masterId parameters from the ini file to know it's serving eNodeB. Then they start the initial attachment procedure with its serving eNodeB by sending Attached-Request-Message using ideal RRC protocol (the message does not require a radio resource and is received without any delay as a result of radio channel. In order to achieve this the OMNeT++ sendDirect method is used between the UeRRC and EnbRRC models). Upon receiving the UE message by the serving eNodeB, the NAS payload together with eNodeB UE S1AP ID, Tracking Area Identifier (TAI), and GCI are attached to S1AP Initial-UE-Message and sent to the MME. It is worth mentioning that the MME selection procedure is not implemented in our model and only one MME can be used in the network simulation and it is required that the name of the module to be exactly (MME), otherwise the eNodeB S1AP module will not be able to locate the MME in the simulation to send the S1AP messages. If the serving eNodeB is OpenFlow capable then the Initial-UE-Message is sent to the SDN controller using OpenFlow over the TCP connection.

At the controller the OpenFlow header is removed and the Initial-UE-Message is forwarded through the northbound interface to the MME. In EPS, when the S1AP Initial-UE-Message is received by the MME the Gateways selection procedure is performed to select SGW and PGW required to establish the UE default bearer. In our architecture, the MME is not responsible for the Gateway´s selection. Instead the SDN controller performs this task utilizing the current utilization of the available FDs to pick the best one to handle the UE upcoming traffic. When the Initial-UE-Message is received from the OpenFlow capable eNodeB a copy is sent to the MME to update the UE profile. At the same time the SDN controller setup the path for the UE upcoming data plane traffic. The process includes identifying the best FDs to handle the UE traffic and configure the FDs to handle the UE traffic based on the UE profile. The FDs of SDN sub-network of the backhaul are equipped with an agent to:

- Reduce the signalling traffic between the SDN controller and the data plane devices.
- Reduce the delay of the first packet of each new flow.

To keep the programmability features offered by the SDN architecture, the agent is modelled to inherit OpenFlow characteristics, which allows the SDN controller to dynamically (re)configure the FDs agent. New messages have been added to the OpenFlow protocol to allow the SDN controller to communicate and exchange information with the FDs. Through these messages the controller can Add/update/delete the configuration of the FDs agent.

6.6.3 Performance Metrics and Evaluation Results

Two experiments are used to validate the operability of the proposed system. The simulation times of all tests are 100s and each simulation is repeated 5 times with different seed numbers, the average result is used as optimal result. In these experiments, a fixed simulated delay has been added to links between the individual entities. The link delay between the eNodeBs and the SGW-FDs is set to 5ms, and another 5ms is the delay of the link between the SGW-FD and the PGW-FD, while the simulated delay of the link to the internet between the PGW-FD and the InternetHost is set to 8ms. The air interface is accurately simulated, and packet delay is compliant to the 3GPP specification of the LTE network. simuLTE [78] is used to provide the air interface between the UEs and the eNodeBs. The EUTRAN parameters are summarized in Table 6.1, which is based on the 3GPP specification for the EUTRAN. This includes the used carrier frequency, bandwidth, eNodeB and UE transmission powers, noise figure, cable loss, and path loss model.

6.6.3.1 UE Handover in Heterogenous Network Experiment

This experiment is utilized to demonstrate the ability of the SDEPC system to operate with and without GTP by leveraging a smart centralized control plane application that allows traffic redirection to and from the GTP tunnel. In this experiment, the InternetHost uses the UDPBasicApp to send UDP traffic to the UE1(0). The application is configured to start after the simulation starts by 0.5s. The application sends a packet of size 400 Byte every 10ms to the IP address of UE1(0). A randomly generated UDP port number is used as the source while the destination port number is set to 5001. The UE1(0) uses the UDPSinkApp that basically deletes the received packet after it has measured and recorded the traffic statistics.

In this experiment, both UEs are equipped with OMNeT++ LinearMobility module that allows the UE to change its location during the simulation utilizing a configurable speed and angle. The mobility module of each UE is configured to move 10 meters every second in a straight horizontal line between eNodeB1 and eNodeB2.The number of packets sent by the InternetHost and the SGW-FD is used as a metric to validate the system operation. The SGW-FD uses port0 to send downlink traffic to eNodeB1 utilizing the GTP tunnel and Port1 to send the downlink traffic eNodeB2 utilizing layer 2 tunnelling. The packets sent by each port are recorded using a time vector and scalar. The handover time is also recorded together with the end-to-end delay.

The obtained results are shown in Figure 6.14, Figure 6.15, Figure 6.16, and Figure 6.17.

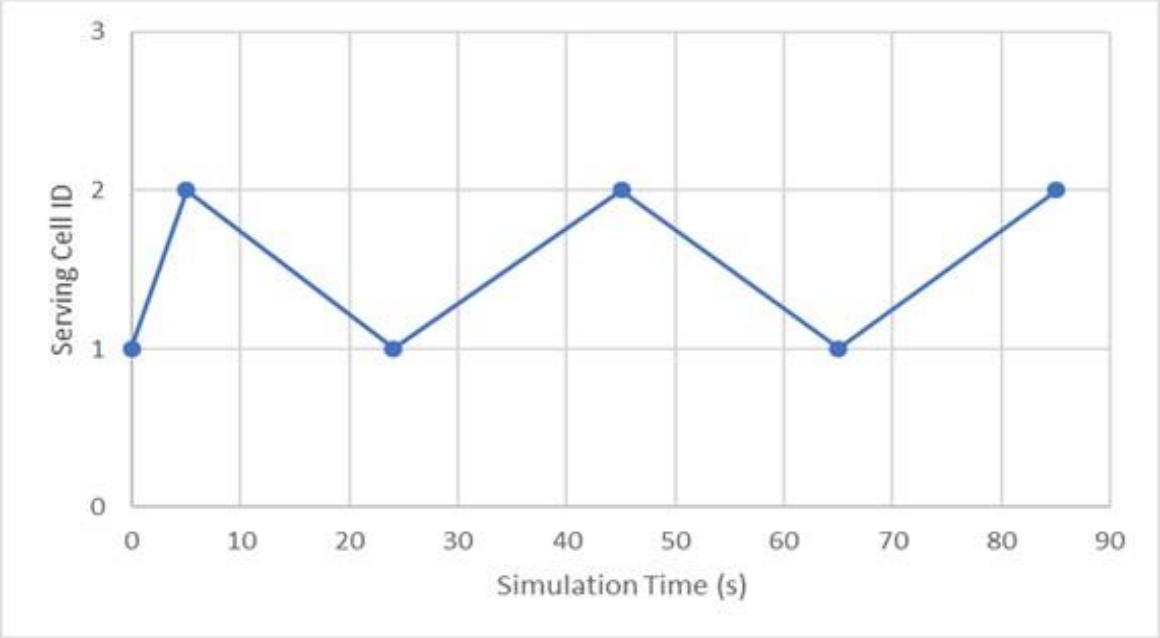


Figure 6.14: UE1(0) serving Cell Id though out the simulation

Figure 6.14 shows the handover times of UE1(0) during the simulation time. The handover time is signalled to the statistic listener after the target eNodeB sends the Switch-Path-Request message to the MME or the SDN controller to change to downlink traffic direction to the correct serving eNodeB. It can be observed that UE1(0) performed 5 handover operations between the two eNodeBs. In the beginning of the simulation UE1(0) starts the default bearer setup procedure using eNodeB1 then after 5s performed the first handover operation to change its serving eNodeB to eNodeB2. The other 4 handover operations are performed on simulation times 24s, 45s, 65, 85s respectively.

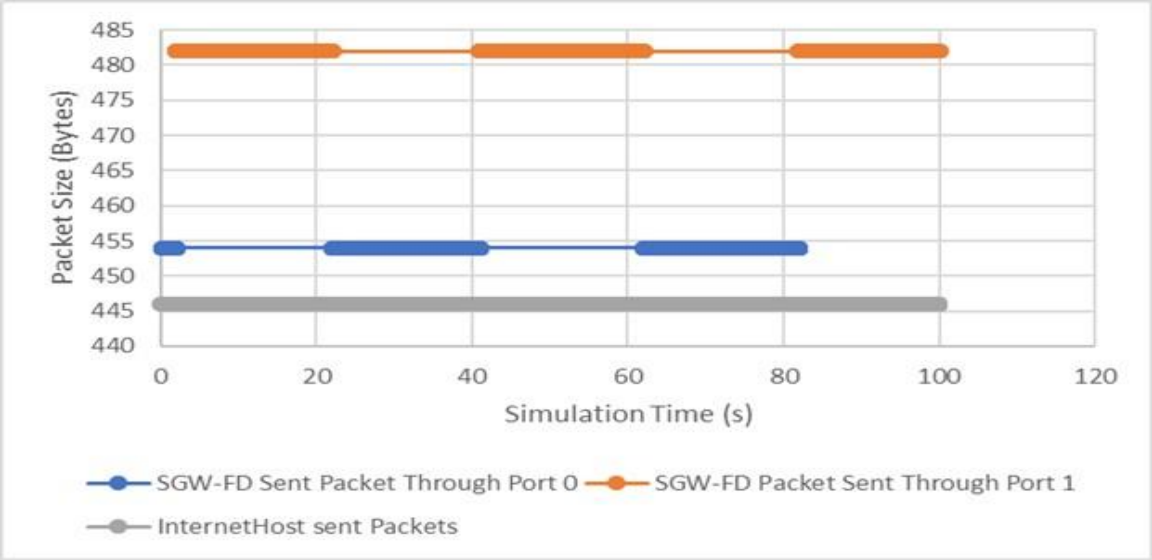


Figure 6.15: Packet sent by the InternetHost and the SGW-FD

Figure 6.15 shows the time vector of the traffic sent by the InternetHost and the SGW-FD. The grey line represents the packet sent by the InternetHost during the simulation time. The blue and orange lines represent the traffic sent through port0 to eNodeB1 and through port 1 to eNodeB 2, respectively.

It can be observed that in the SGW-FD the downlink traffics are bouncing between port 0 and port 1 and the time of changing of the forwarding port is in line with the handover time shown in Figure 6.14. After the simulation started the downlink traffic is sent through port 0 to eNodeB1 and around 5s the sending interface is changed to port 1, which changes the direction of the traffic to eNodeB 2, where the behaviour is continued till the simulation time reaches the second 24. At that moment, the sending port changed to port 0. The sending port is changed each time UE1(0) changes its serving eNodeB using the X2 handover procedure. The change in the sending port happened again around 45 s, 65 s, and 85 s. It is also worth noticing that the packet size sent through the layer 3 tunnel (GTP) is larger than the packet sent through the layer 2 tunnel. In fact, this is due to the extra headers added by GTP to each packet to route the packet through the network backhaul to the serving eNodeB. The GTP tunnel statistics is recorded and plotted in Figure 6.16. While Figure 6.17 shows the end-to-end delay recorded by the UE1(0) during the simulation time.

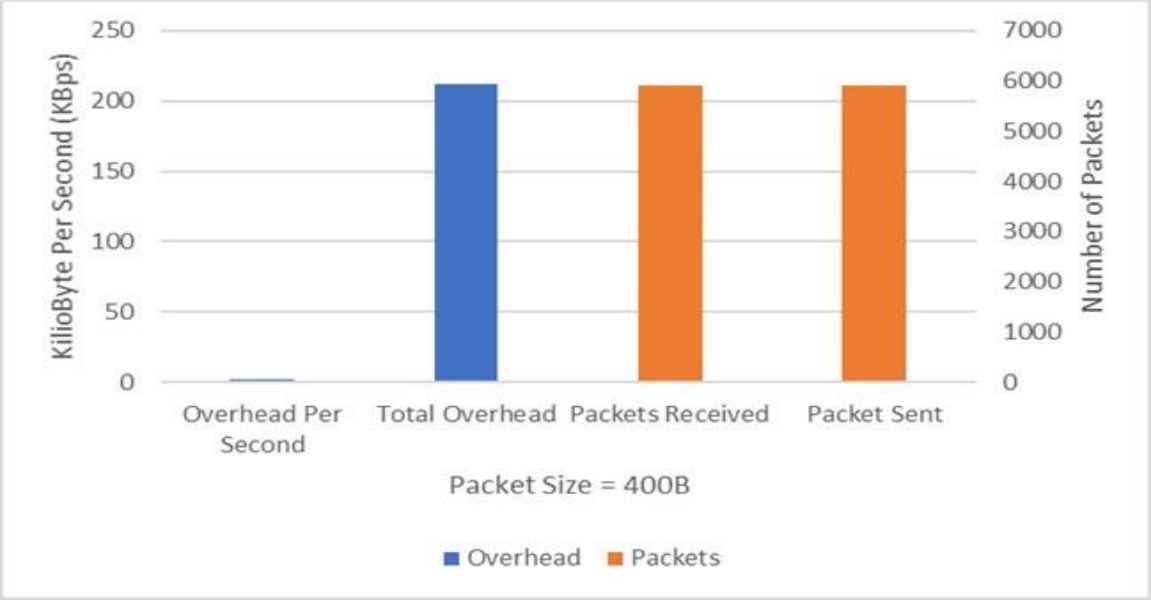


Figure 6.16: GTP tunnel statistic When the Packet size equal to 400 Byte

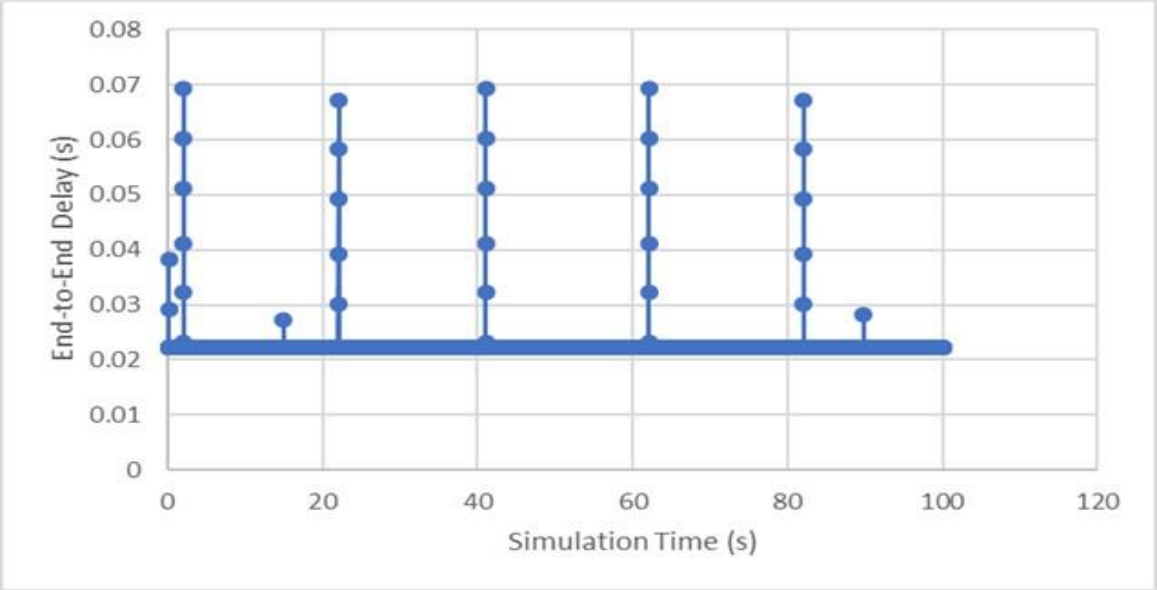


Figure 6.17: SDEPC UE1(0) end-to-end delay

It can be observed that the end-to-end delay measured by UE1(0) has several spikes that represents an increase in the end-to-end delay. It is obvious that this extra delay is due to the buffering together with the X2 transmission delay. Bear in mind that the time of the delay spikes is the same as the handover times shown in Figure 6.14. The first spike is due to ARP resolution and first packet processing. The link delay of the connection between the PGW and internet Host is set to 8ms and in our simulation an ethernet link is used to

connect these two nodes. When the UDPBasicApp send the first chunk of data to the IP address of the UE1(0). it passes through the UDP and network modules. At the network layer ARP procedure is performed to resolve the MAC address of node advertised public IP address of the UE. In reality, this will not be the case and the first spike will not be visible. Overall the system performance is still very good, and this extra delay caused by the handover operations did not affect the system performance considering that the peak delay reported by UE1(0) is 69ms which is still a very good end-to-end delay and will not affect the user experience.

6.6.3.2 SDN Controller Location Experiment

In this experiment, two runs are used to measure and compare the system performance. In the first run, the OpenFlow switches of the backhaul network are equipped with an SDN agent while the second run uses standard OpenFlow switches as specified by the OpenFlow 1.3 specification. In this experiment the end-to-end delay of the first packet and the distance of the backhaul SDN controller from the switches are used as metrics to evaluate the system performance. The end-to-end delay of the first packet of downlink traffic from the InternetHost to 10 UEs connected to eNodeB1 is shown Figure 6.18. For simplicity, run 1 and 2 will be referred to as network 1 and 2 respectively.

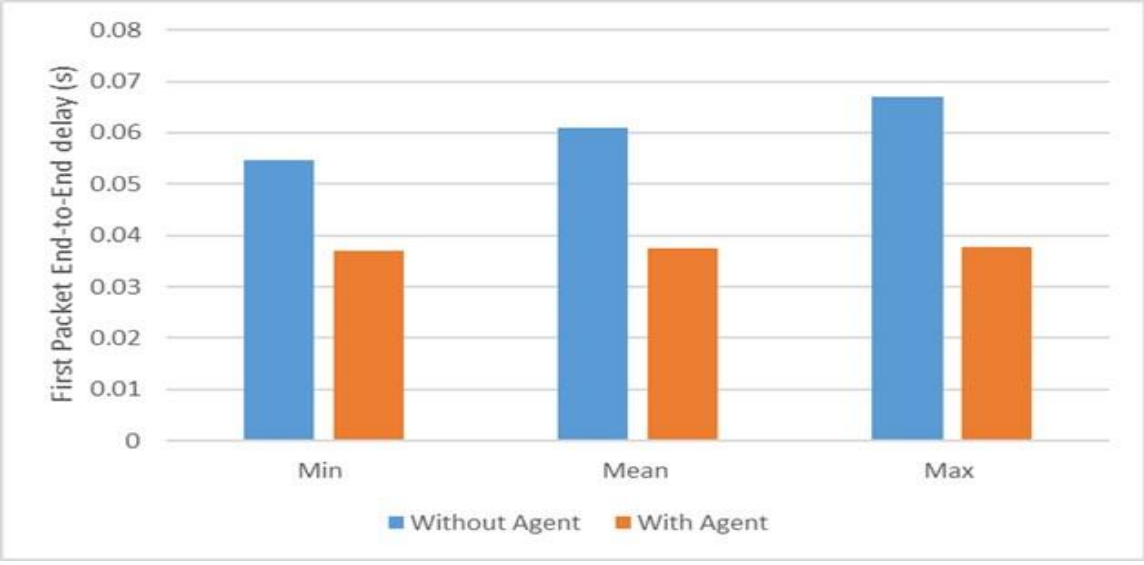


Figure 6.18: end-to-end delay of downlink traffic first Packet

It can be observed from the obtained results that the delay of the first packet recorded by all the UEs is less with network 1. In network 2, the first packet of each flow is buffered, and headers information are sent to the controller for processing. Upon receiving the controller instructions, the buffered and the upcoming packets of the flow are forwarded using the installed rules.

Meanwhile, the FDs of networks 1 has an agent that is preconfigured by the controller to correctly forward the received traffic to the correct serving eNodeB based on the header information of the received packet. This helps the OpenFlow switches to handle the received packet locally without contacting the controller which contributes in reducing the buffering and processing time. To investigate the effect of the controller distance from the OpenFlow switches on the delay of the first packet of downlink traffic sent by the InternetHost to UE1(0) that is connected to eNodeB1, multiple simulation runs are used to measure the delay of first packet when the controller is 172, 263, 316, 458, 525, 662 Kilometres away from the OpenFlow switches (OFS1 & OFS2 in our simulation).

In this test the delay of the link between the OpenFlow switches and the SDN controller is calculated by utilizing the geographical distance between the two nodes. The delay of the fibre cable used in the simulation is calculated by dividing the geographical distance between the nodes by the propagation speed for optical fibre. The propagation speed of the fibre cable used in our simulation equals to the speed of light divided by the 1.5 which represent the refraction index of the fibre cable as shown in equation 1.

$$Linkdelay = \frac{distance}{\left(\frac{speedoflight}{1.5}\right)} \quad (6.1)$$

The Link delay of connection between the OpenFlow Switches and the controller using the aforementioned distances is equal to 0.00086s, 0.001315s, 0.00158s, 0.00229s, 0.002625s, 0.00331s respectively. The orange line shown Figure 6.19 represents the delay of the first packet when the FD has a local agent while the blue line represents the delay of the first packet when the FD does not have a local agent.

As expected in Network 2, the delay of the first packet increased proportionally with the distance of the controller and in Network 1, the first packet delay is not affected by the increased distance between the OpenFlow switches and the SDN controller. In Network 1, the agent of the OpenFlow switches is preconfigured by the backhaul controller. Therefore,

the distance between the switches and the controller has no effect on the delay of the first packet. Differently, in Network 2 the first packet is sent to the controller for processing and because of that the distance between the backhaul OpenFlow switches and the controller has a clear effect on the delay of the first packet.

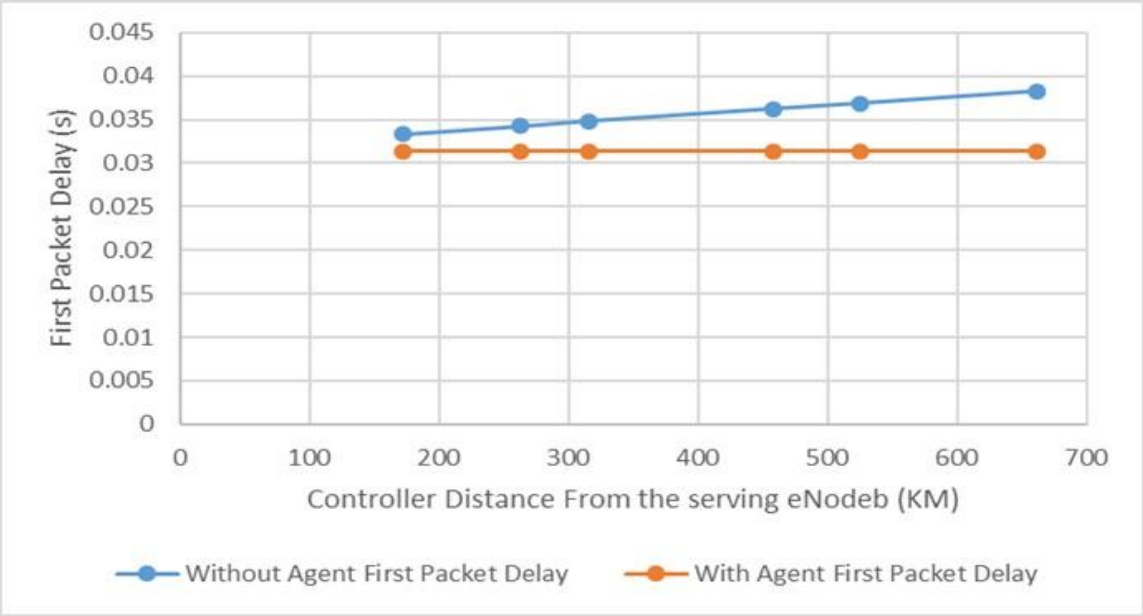


Figure 6.19: First packet delay with respect to the controller location

6.7 Summary

In this chapter, SDEPC architecture is presented and the OMNeT++ simulation model is described. Several experiments are conducted to compare the performance of the SDEPC and the standard LTE networks. The link utilization, end-to-end delay, packet loss, queuing time is used as metrics to evaluate the performance of the systems. Removing GTP proposed by the SDEPC simplifies the data plane network and increase the network flexibility which contributes to enhancing the network performance in term of end-to-end delay especially with the congested network. At the same time, removing GTP introduce backward compatibility issue with the existing mobile network architectures. Therefore, modification to the SDEPC architecture is presented to support data forwarding with and without GTP. OpenFlow FDs enhanced with a local agent to further improve the system operation and reduce the number of interactions between the controller and the FD and help reduce the number of installed flow entries in each FD. Two experiments are

conducted to validate the operate-ability of the proposed architecture. The next chapter shows how removing GTP help to deploy new services in the backhaul network and easy and simple manner.

CHAPTE 7

7 Software Defined Selective Traffic Offloading

7.1 Introduction

The current advances in the technologies being deployed in mobile networks are not able to cope with the unprecedented traffic amount being traversed across the networks due to the increasing number of smartphones and tablets and so on, where delay is still a common issue facing network operators and developers, especially when it comes to time sensitive applications such as online gaming applications [87], since these applications require real time communications or near real time. If you encounter delays in online games, then it will affect badly on these applications especially they require high level of QoS. One of the main reasons for ongoing delay issue is related to the architecture of the current mobile cores and the fact that each packet in the mobile network has to travel relatively long distance before even reaching to the gateway nodes (Packet Data Network Gateway (PGW) nodes in LTE/EPC nomenclature), because the gateways are mounted in centralized distribution, which will create a hierarchical routing causing the delay issue mentioned earlier.

There are some approaches that are proposed to overcome this issue such as various offloading strategies, including offloading to data centres inside the mobile provider core network footprint [11], however these approaches suffer from difficulty in deployment in real networks due to the fundamental complexity of mobile architectures. Assuming a cloud-based infrastructure distributed in a near proximity from the mobile access network which allows the operators to selectively offload some of it delay sensitive traffic. This

Chapter describes three network designs to implement cloud offloading. These solutions implemented with 4G network but can be also adopted in 5G network as well. All three solutions have similar network design that is composed of three sub-networks namely, Access, backhaul and core networks. The backhaul network consists of two parts: i) access; ii) aggregation. The main components of the selective traffic offloading solutions are implemented in the access part of the backhaul network. The solutions mainly consist of a Traffic-Offloading-Switch and a SDN controller. Based on the solution, slight modifications maybe required to the standard mobile network entities. More specifically, the first cloud offloading solution only requires a slight modification to the MME, while the second solution requires modification to both the eNodeB and MME to improve the processing delay of the first solution. The last solution is different than the first two solutions because it is based on the idea of GTP tunnel elimination.

7.2 Solution 1 Network Design

The aim of first solution is to minimize the change required to the standard LTE network and supports selective traffic cloud-based offloading operations [12][14]. The only change required in this design is to the MME entity.

In this solution the MME has a connection to the SDN controller to obtain network statistics and notify the controller to perform the traffic offloading. The MME has been modified to track and keeps the GTP-U tunnel information between the eNodeB and the SGW, which includes information like the TEID and IP addresses of both the eNodeB and SGW, as well as the UDP source and destination port numbers. The SGW TEID and IP address are obtained from the Create-Session-Response message sent to the MME by the SGW, while the eNodeB TEID and IP address are obtained from the S1AP Initial-Context-Setup-Request-ACK message. After obtaining the GTP-U information and if the UE is eligible for traffic offloading, a Traffic-Offload-Notification message is sent by the MME to the SDN controller as shown in Figure 7.1. The UE is considered as eligible for traffic offloading, either if it is registered by the network operator, or if an offload request is received from the server that is providing the service to the UEs. The main new elements in this network design is the SDN controller and the Traffic-Offloading-Switch, although

these elements are used in all three solutions, the functionality and the features offered by them are different in each solution.

In this design, the GTP between the eNodeB and the SGW is kept the same and the SDN solution works with the MME to perform the traffic offload to the cloud infrastructure. This solution specifies the traffic that should be offloaded and configures the offloading switch to extract the traffic from the GTP-U tunnel and send it to the cloud, and the offloading switch is also configured to return the traffic coming from the cloud back to the GTP-U tunnel again. These operations are happening without the eNodeB and the SGW being aware of the traffic redirection operation.

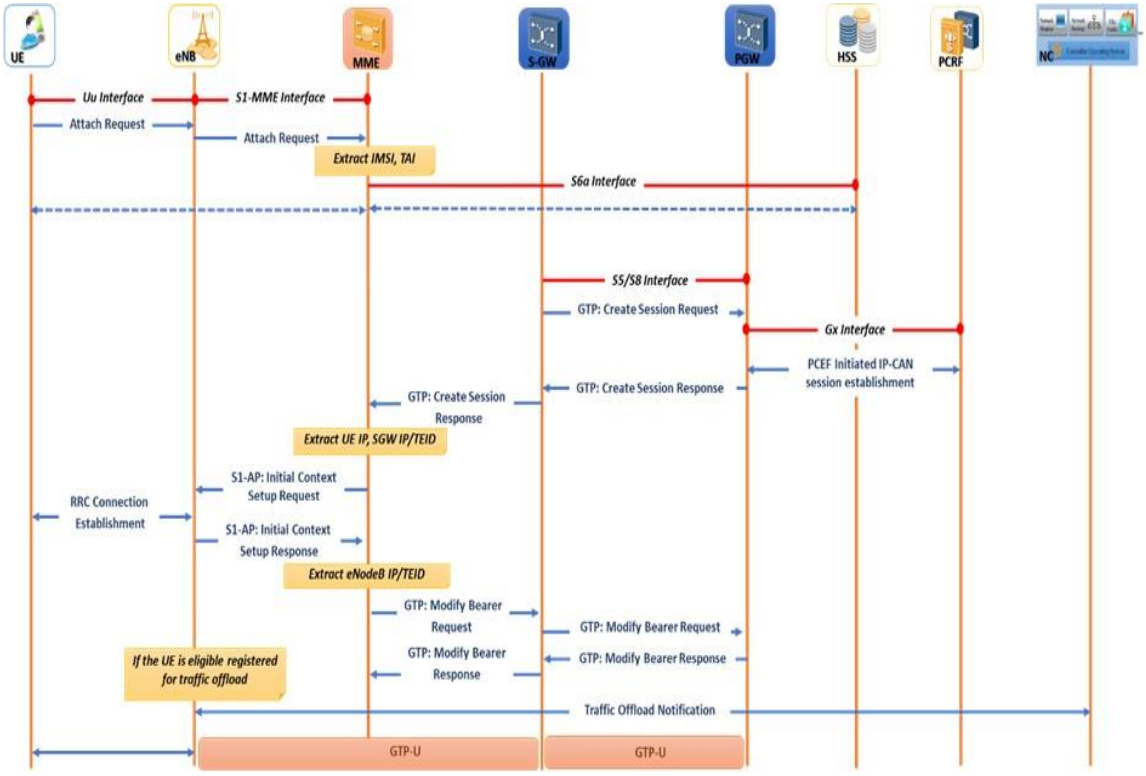


Figure 7.1: Solution 1 - Traffic Offload Setup Call Flow Sequence

7.2.1 Traffic Offloading Switch Implementation in OMNeT++

The Traffic-Offloading-Switch (TOS) is modelled in OMNeT++ as an extension to our work in [82] that provides a standard implementation of Openflow 1.3 switch. Basically, the TOS module inherits the structure of the standard Openflow 1.3 switch by leveraging the modularity and the object-oriented structure offered by OMNeT++. Therefore, most of the switch functionality is reused in our module of the Traffic-Offloading-Switch. The

main difference between the Traffic-Offloading-Switch and the standard OpenFlow 1.3 switch is the ability to perform GTP tunnelling.

The Traffic-Offloading-Switch is modelled to support GTP operations by extending several modules of OpenFlow 1.3 switch. Precisely, the extensions introduced to the flow processing pipeline and the OpenFlow plugin entities. The processing pipeline has been heavily modified considering that the Traffic-Offloading-Switch processing pipeline offers more features compared to the standard 1.3 switch processing pipeline. The module is converted to a compound module that includes several simple modules. Precisely, the module consists of the processingUnit, mux, enCap-vPort, Inspection-vPort and deCap-vPort modules. The processingUnit represents a modified version of the open flow processing module of the standard OpenFlow 1.3 switch module that basically inherits the main operation of the open flow processing module and adds extra features that are necessary and required to perform the GTP encapsulation and decapsulation. To ensure accuracy and reliability, the UDP and network layer modules offered by the OMNeT++ INET library have been reused to add the layers required to perform GTP encapsulation. Since the transport layer (UDP) and the network layer are already modelled and tested by the INET library it is unnecessary to program the same functionality in the enCap-vPort module. Therefore, the enCap-vPort module only adds the GTP header and the other layers are added by the INET's UDP, network and eMAC modules. The OpenFlow plugin is modelled by the OFA Switch To module that inherits the functionality offered by the OFA Switch module of the standard OpenFlow 1.3 switch and adds extra features.

The new features are used to allow the Traffic-Offloading-Switch to understand the OpenFlow extensions messages. Moreover, a new module has been added in the Traffic-Offloading-Switch known as the TEID-table. This table is used by the TOS in order to keep and maintain the GTP tunnel configuration. This section summarizes the changes made to the standard building blocks of OpenFlow 1.3 switch module. Moreover, a brief description of the new components structure, design and functionality is provided. The Traffic-Offloading-Switch shown in Figure 7.2 shares the same structure of the standard OpenFlow 1.3 switch.

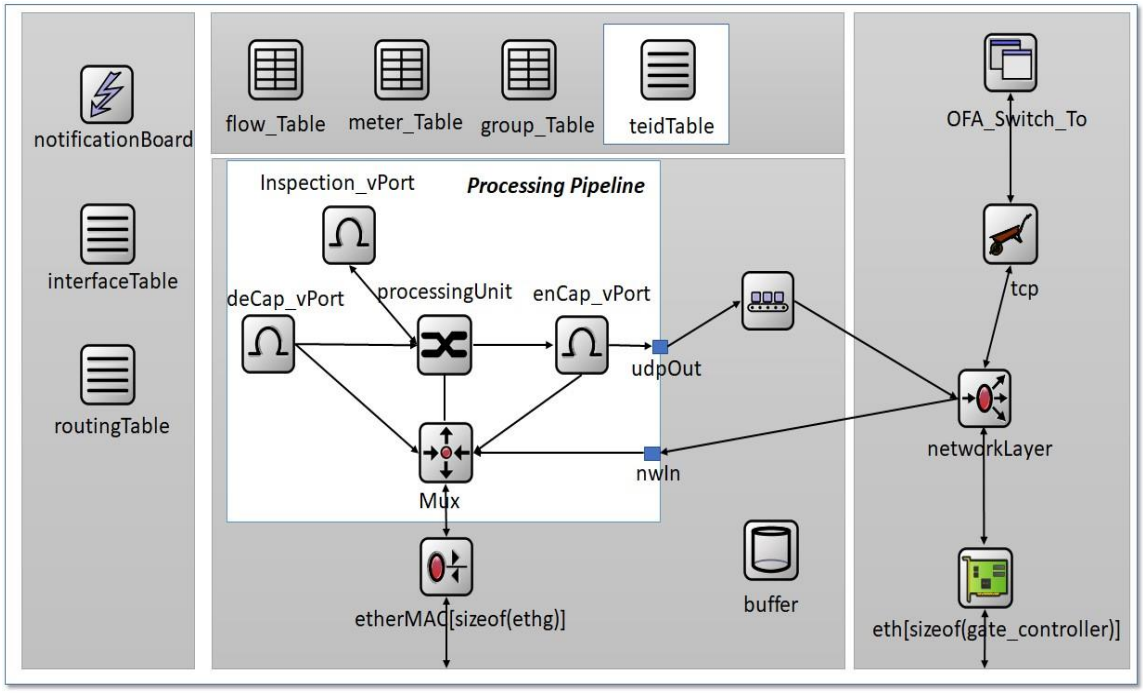


Figure 7.2: OMNeT++ module of the Traffic Offloading Switch

7.2.1.1 TEID Table Module

This is modelled as a OMNeT++ simple module that works as a container for the GTP tunnels information. The same as the flow-table, the TEID-table consists of one or more TEID entries, where each entry consists of two fields, namely the tunnel-Id which works as an Identifier to differentiate between the table entries.

The second field is the GTP tunnel configuration parameters, which includes information like the TEID and IP address for both the eNodeB and SGW, source and destination UDP port number, and the number of the output physical port. The TEID-table module is used by other modules in the Traffic-Offloading-Switch, specifically the OFA_Switch_To and the enCap-vPort modules. The latter is a simple module that is part of the switch processing pipeline compound module that is explained in Section 7.2.1.3.3.

7.2.1.2 OFA_Switch_To Module

This part of the offloading switch is also modelled as OMNeT++ simple module and as previously mentioned it is modelled as an extension to the OFA_Switch module by inheriting its functionality. It adds more features that are required to implement the traffic

offload to the cloud. This section describes the main changes and the added features and functionality, which includes the process of using the TEID-table by this module.

The OFA_Switch_To module represents the point of interaction with the SDN controller using OpenFlow protocol over TCP (Transport Layer Security (TLS) is not implemented). Most of the features offered by this module are inherited from the OFA_Switch module of the standard OpenFlow 1.3 switch, but at the same time, this module offers a set of new functionalities that are mainly related to GTP operations. Considering that GTP is not supported by OpenFlow, an extension to the protocol to support the required operation for the traffic offloading is implemented. This includes a set of new OpenFlow messages that has been defined to support GTP tunnel configuration operation (add/modify/delete). Furthermore, the GTP-Tunnel-Setup-Request, GTP-Tunnel-Modify-Request, and GTP-Tunnel-Delete-Request messages have been defined and structured to be used as part of OpenFlow message exchanges between the controller and the data plane switches to support the aforementioned GTP tunnel operation.

The OFA_Switch_To module has the required tools and functionality to receive, read and understand the new messages. Based on the type of the message the TEID-table is configured by the OFA_Switch To module leveraging the information of the received control message. The extensions to OFA_Switch module follow the same structure and operation steps defined by the OpenFlow switch 1.3 standard. When a control message is received from the controller through the TCP connection, the OFA_Switch_To module is responsible for handling the message and response to the controller if it is necessary. The message type is checked first and based on the type the correct method is called to handle the message for example if the GTP-Tunnel-Setup-Request is received, and then handleGTPTunnelSetupRequest method is called to handle the message. This method reads the control message and obtains the tunnel-Id value, which is used to check the TEID-table entries. If no match is found in the table, a new TEID-entry is added to the table. If a TEID-entry with the same tunnel-Id exists in the table, then an error message is sent back to the controller. This message includes the type and the code of the error as specified by the OpenFlow standard. Therefore, the error type set is to OFPET GTPTUNNEL STEUP FAILED, and the code field is set to OFPGTC TUNNEL EXISTS. These two parameters

are used to inform the controller that the switch already has a TEID-entry with the same Tunnel-Id specified by the message sent by the controller.

The same procedure is used with the other types of the GTP-Tunnel messages. Leveraging these messages, the controller can setup a new GTP tunnel, update or delete an existing tunnel. For example, GTP-Tunnel-Setup-Request message is used to configure the offloading switch at the beginning and when the UE changes its location while it is still using the offloading cloud services, the GTP-Tunnel-Modify-Request message is used to update the tunnel configuration with new parameters that reflect the GTP tunnel information with the new eNodeB.

7.2.1.3 Processing Pipeline

In the Traffic-Offloading-Switch, the processing pipeline is modelled as a compound module. It consists of several simple modules, namely mux, deCap-vPort, enCap-vPort, and processingUnit modules and each one of them is described in next sub-sections.

A. Mux Module

It is a very simple module that is only concerned about sending the traffic to the correct physical port. In our module, the mux has a bidirectional connection with each physical port through the eMAC module, and at the same time it has unidirectional connections (equal to the number of physical ports) with the processingUnit, enCap-vPort, deCap-vPort and the network modules known as ToOfprocessing, fromEnCap, fromDecap, fromNetLayer respectively. Incoming packets are sent to the processingUnit module without any modification through the ToOfprocessing connection. In our design the mux uses one to one mapping between the physical port number and the ToOfprocesisng port number. For example, packets received on physical port number 1 are sent by the mux to the processingUnit module through ToOfprocessing connection number 1, and after processing the packet in the module the packet is sent to the mux again through either the enCap-vPort or deCap-vPort or the network module. This is based on the operation performed by the processingUnit module. Similarly, if the packet is received through connection 2 then it is sent through physical port 2.

B. ProcessingUnit Module

This module is responsible for most of the OpenFlow processing operations, which includes packet headers extraction, flow-table lookup, and flow-entry instruction execution. This module inherits the structure of the open flow processing module of the standard OpenFlow 1.3 switch presented in [82]. Extra functionality and operations have been adopted in this module to support the required feature to perform offloading operation. The module is physically connected to both the enCap-vPort and deCap-vPort modules, and it has a vector of connections to receive the traffic from the mux module.

C. enCap-vPort Module

This module has been added to the processing pipeline to include the support for GTP encapsulation in the Traffic-Offloading-Switch. This module has three connections that are known as fromOfprocessing, toMux, and toUDP connections. The first one represents a unidirectional connection to receive traffic from the processingUnit module while the second connection represents a vector of unidirectional connections to send the traffic received from the processingUnit to the physical port without adding the GTP headers. The last connection is used to perform the GTP encapsulation operation. When a packet is received by the enCap-vPort module, it can be sent to either the mux through the toMux connection or to the UDP module through the toUDP connection.

The process starts by removing the control info object from the received packet. This object is attached and contains information like the tunnel-Id, Physical port number, and layer 2 headers information. The tunnel-Id is obtained and interrogated, and if it is marked as unspecified, then the packet is sent to the mux through the connection that equals to the value specified by the physical port number. Before sending the packet through the toMux connection, the layer 2 header is added to the packet using the information provided by the control info object. If the tunnel-Id has a valid value, then the TEID-table is checked and inspected to find a matching entry. If a match is found, then the tunnel configuration parameters are obtained, and the packet is encapsulated with GTP header and sent to the UDP module to add the extra layers information that includes UDP, network and layer headers.

D. deCap-vPort Module

This module is also connected to both the mux and the processingUnit modules. It has a unidirectional connection with the processingUnit module to receive the traffic and a vector of connections that equal to the number of the physical ports to send the traffic to the cloud after GTP decapsulation. When a packet is received by this module, the control info object is extracted to obtain the value of the outport field, then the packet is decapsulated and the inner packet is extracted and sent to the cloud through the mux module. To send the packet to the mux module the outport value obtained from the control info object is used to send the packet through the correct connection that makes the mux forward the packet to the correct physical port.

7.2.2 Control Plane Operations

This section describes the process of configuring the Traffic-Offloading-Switch to match a specific uplink traffic and extract it from the GTP tunnel, then send the inner packet to cloud-based infrastructure that are distributed near to the access network to reduce the latency and improve bandwidth utilization in the core network.

As shown in Figure 7.3, the process starts with the MME. It maintains the bearer information of the UE traffic that is considered eligible for cloud offloading. Then the bearer information is sent to the SDN controller in a form of Traffic-Offload-Notification message. In this message the MME includes information like the GTP-TIED and IP address of the SGW, with the TEID and IP address of the eNodeB, and the UE IP address is also included in the notification message along with other parameters. Upon receiving the notification message, the cloud offloading application of the SDN controller handles the message by extracting and obtaining the S1-U tunnel information, and this information together with the UE IP address is used to configure the Traffic-Offloading-Switch to offload the UE traffic sent over this bearer to the cloud.

To Configure the Traffic-Offloading- Switch, the controller first sends a GTP-Tunnel-Setup-Request message to the switch, in which the controller includes the tunnel-Id together with the GTP tunnel configuration parameters. When the switch receives this message, it configures the TEID-table, and this process involves adding a new TEID-entry consisting of two fields, namely: the tunnel-Id (entry identifier), and the tunnel

configuration parameters. The latter consists of TEID and IP address of the eNodeB, SGW IP address, and the source and destination UDP port numbers together with the output port number are also included in the structure of the tunnel configuration parameters.

The second step performed by the controller is the flow-table configuration. Therefore, the sequence of Flow-Mod messages is sent to Traffic-Offloading-Switch to configure the flow-table to handle both the uplink and downlink traffic. Each Flow-Mod message has match-fields and instructions along with other parameters. Upon receiving the message, the Traffic-Offloading-Switch configures the flow-table by adding a new flow-entry that matches the traffic with the same header information specified by match-fields of the Flow-Mod message, and executes the instruction specified by the message on the matched traffic.

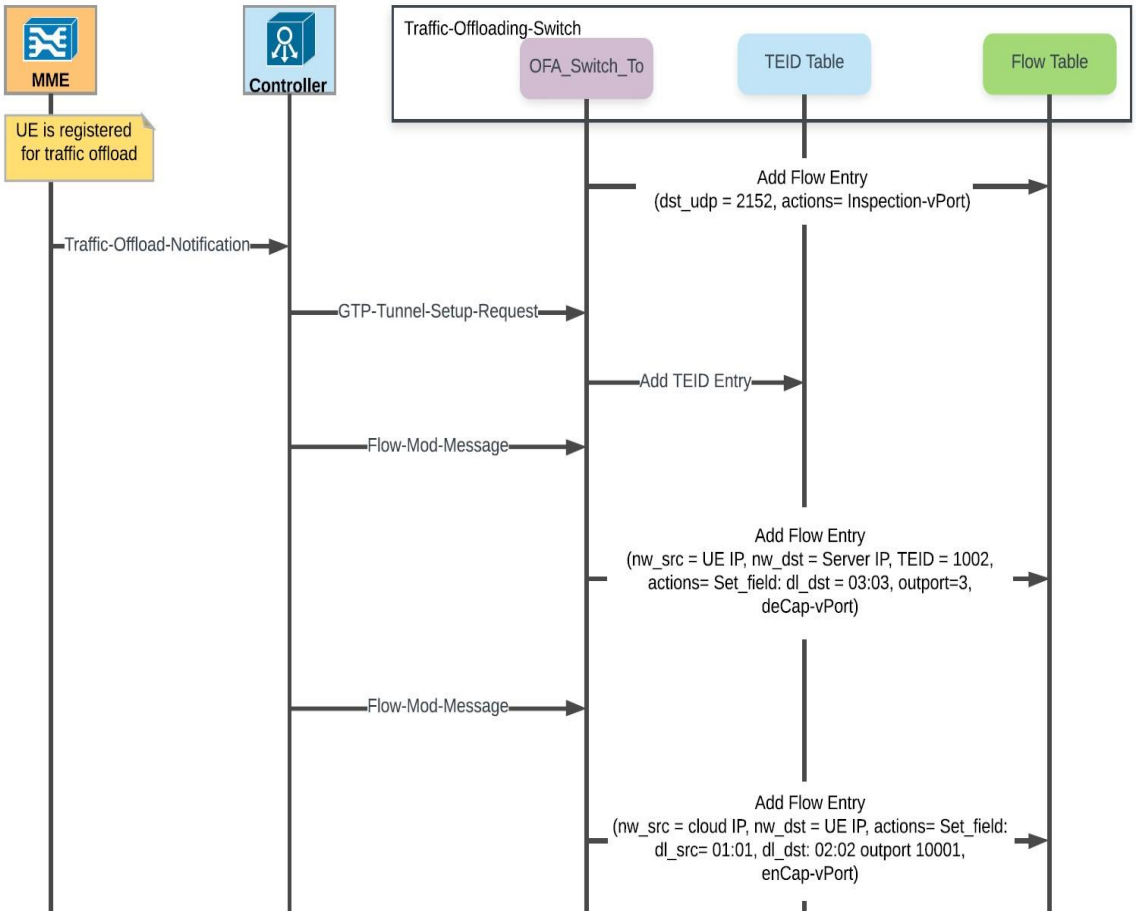


Figure 7.3: Solution 1 - TOS Configuration Procedure

Figure 7.4 shows a snapshot of the flow-table configured by the controller to handle traffic offloading operations.

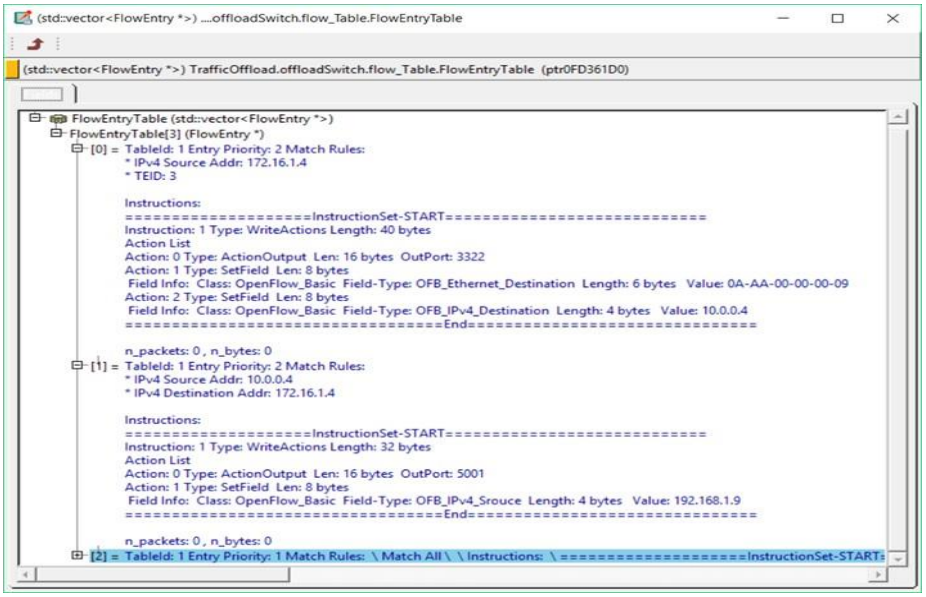


Figure 7.4: Solution 1 - flow-table of the TOS Module

Walkthrough example is the best and easiest way to fully understand the traffic forwarding operations. This section explains the control plane setup procedure by using the example shown in Figure 7.5.

In the beginning of the simulation the controller configures the data plane network to forward the traffic based on the Layer 2 information, also a static flow-entry is configured in each offloading switch to match traffic with UDP port 2152 to be sent to GTP inspection logical port that implemented as an internal part of the processing pipeline module. Figure 7.5 shows the network configuration after setup of the EPS bearer. The UE with the IP address 10.0.0.1/24 has GTP-U tunnel between the eNodeB1 and the SGW. The GTP-U tunnel has uplink TEID equal to 1002 (also known as the SGW TEID) and downlink TEID equal to 1003 (also known as the eNodeB TEID).

The eNodeB IP address is 192.168.2.1/24 with mac address equal to 00:00:00:00:02:02, while the SGW has the IP address 192.168.1.1/24 and mac address 00:00:00:00:01:01. In this tunnel UDP port 2152 is used as the destination port number and a randomly generated UDP port number used as the source for each side of the tunnel. After setup the tunnel, the MME sends a Traffic-Offload-Notification message to the controller as previously explained. In this example the message includes the UE IP address (10.0.0.1/8). The GTP-U tunnel information that includes 1003 and 192.168.2.1 as the eNodeB TEID and IP

address respectively, it also includes the 1002, and 192.168.1.1 as the SGW TEID and IP address respectively.

When the controller receives the Traffic-Offload-Notification message, the offloading application processes the message and obtains the important information and uses the OpenFlow interface to configure the offloading switch. The configuration of the switch starts by sending a GTP-Tunnel-Setup-Request message. Before sending this message, the offloading application generates a tunnel-Id for this UE from a configurable pool. In this example the tunnel-Id 5001 is used. The next step is to fill the tunnel configuration parameters struct with the required information, which is obtained from the MME notification message.

The configuration parameters that include the 1003 as the eNodeB TEID, 192.168.1.1 and 192.168.2.1 as the source and destination IP address respectively, UDP source and destination port numbers together with the output port are also included as a part of the tunnel configuration parameters. After acquiring all the message information, the GTP tunnel Setup Request message is built and sent to the offloading switch. After configuring the TEID-table of the offloading switch, the controller then sends two flow mod messages to add two flow entries that handle both the uplink and the downlink traffic. The first Flow-Mod message matches traffic that have 10.0.0.1/8 and 1002 as the source IP address and TEID respectively. The instruction in this message is set to send the packet to deCap-vPort and two set field actions to set the source and destination mac addresses to 00:00:00:00:02:02 and 00:00:00:00:03:03 respectively.

To summarize this flow-entry matches UE1 traffic that is sent over GTP-U of bearer 1. Then it decapsulates the packet, extracts it from the GTP tunnel and sends it to the cloud infrastructure. The second Flow-Mod message matches the traffic with source IP address that is equal to the cloud server IP address and the destination IP address equal to 10.0.0.1/8. The instruction included in this Flow-Mod message has two sets of field actions to specify the source and destination mac addresses, in this case 00:00:00:00:01:01 and 00:00:00:00:02:02 respectively. The output action specifies the tunnel-Id 5001 as the output port. The processingUnit module uses this information to fill the control info object fields before sending the packet to the enCap-vPort

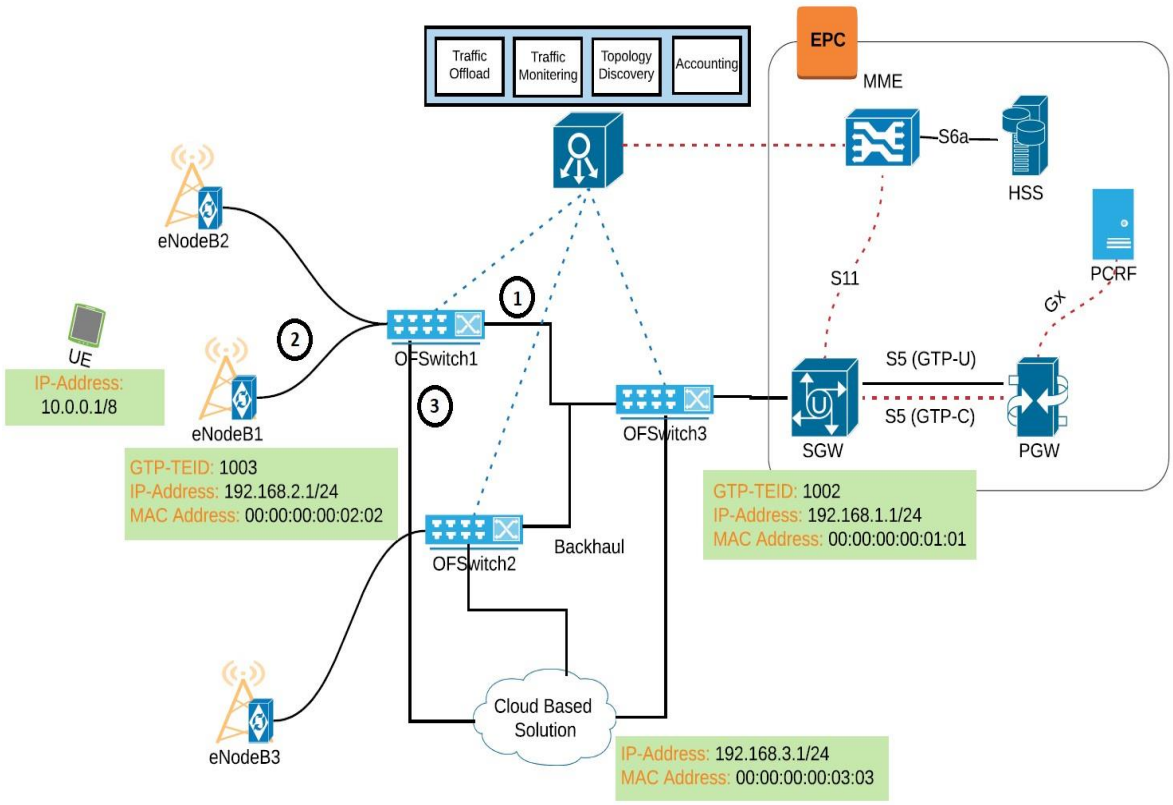


Figure 7.5: Logical Topology of the distributed cloud based offloading solution 1 and 2

7.2.3 Data Plane Traffic Forwarding Procedures

This section briefly describes the network operation performed by network entities to handle both the uplink and downlink. The focus of this section is to describe the operation performed by the Traffic-Offloading-Switch to deliver the received traffic to the correct destination. The procedure performed by the Traffic-Offloading-Switch to process the data plane traffic is shown in Figure 7.8.

7.2.3.1 Uplink Traffic

The main operations performed by the Traffic-Offloading-Switch are shown in Figure 7.6. The received uplink traffic is either sent to the SGW as normal or decapsulated to extract the original packet sent by the UE from the GTP tunnel, which sends it to the cloud. The processing pipeline is responsible for the handling of the uplink traffic received by the switch. In our switch module, the processingUnit module performs the main operations to handle the packet.

The Packet is passed from the mux to the processingUnit module. At this stage, packet header information is extracted and used to find a matching entry in the flow-table. This assumes that the offloading switch is loaded with a flow-entry that matches traffic with UDP port 2152 (GTP-U traffic) to be sent to Inspection-vPort that is adopted in the TOS implementation as an internal part of the processing pipeline module. The main operation performed by Inspection-vPort is the update of the matching fields with new values that include GTP TEID, the source and destination IP addresses of the original packet. Then It sends the packet back to the processingUnit module.

The processingUnit repeats the process again and lookups the flow-table using the new matching fields. If a match found, then the packet is offloaded to the cloud. To do this, the instruction of the matched entry is obtained, and a new control info object is created to include information like the physical port number and the layer 2 header information. This information should be obtained from the instructions of the flow-entry and the packet is send to the deCap-vPort. The latter follows the same procedure that was previously explained in Section 7.2.1.3D. to forward the packet to a cloud-based infrastructure.

In case there is no flow-entry match, the headers information obtained from the Inspection-vPort, then the packet is normally forwarded to the SGW of the UE. This is done by sending the packet to the enCap-vPort with a control info object attached to the packet. In this object the tunnel-Id is set to unspecified, and the outport set to the number of physical port number specified by the output action of the flow-entry instructions. The value of the outport represents the physical port number connected to the SGW (Port number 1 in Figure 7.5). The layer 2 source and destination mac addresses are set to eNodeB mac address as the source address and the SGW mac address as the destination address. When the packet has reached the enCap-vPort module, the procedure previously explained in Section 7.2.1.3C. is performed to send the packet to the correct SGW without any modification.

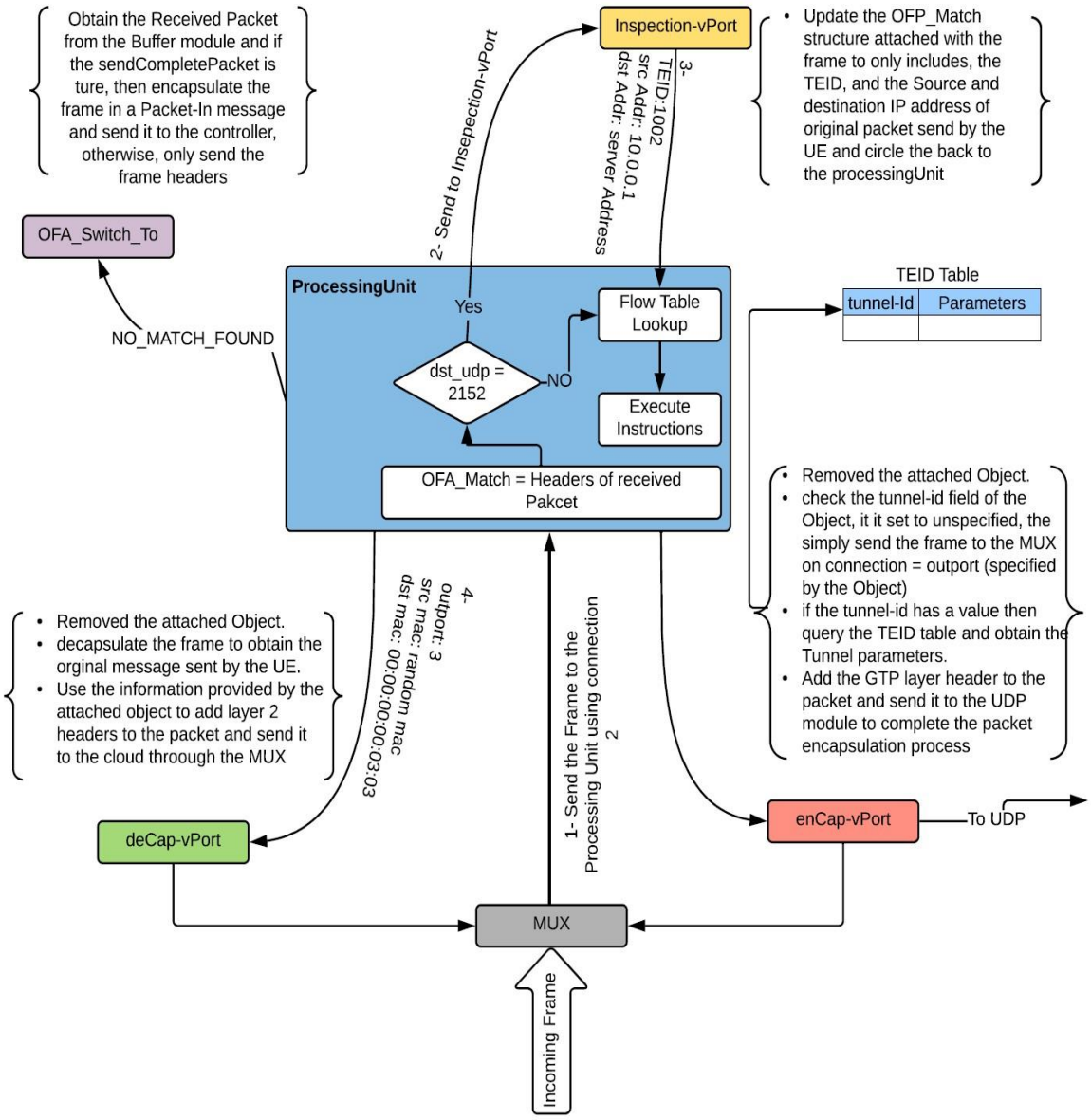


Figure 7.6: Solution 1 - TOS Procedure to extract Uplink traffic from GTP tunnel and send it to the Cloud-Based infrastructure

7.2.3.2 Downlink Traffic

Traffic-Offloading-Switch receives download traffic from either the cloud or the SGW. Downlink traffic from the SGW is GTP-U traffic that should be handled normally and just forwarded to the correct eNodeB without any modification, while the cloud traffic is the traffic that needs to be returned to the GTP tunnel from which it was previously extracted to make the LTE forwarding entities (SGW and eNodeB) unaware of the offloading

operation. Figure 7.7 shows the steps performed by the Traffic-Offloading-Switch to handle the downlink traffic. When a packet is received by the Traffic-Offloading-Switch, it passes from one module to another until it reaches the processingUnit module where the main processing is performed.

In this module, packet header information is extracted from the packet and compared against the flow-table entries, and if the packet is received from the SGW, then it should have UDP port 2152 as the destination port number in the transport layer header. The packet is sent to the Inspection-vPort to obtain the GTP header information and update the matching fields, and then it is circled back to the processingUnit module. The latter checks the flow-table again using the new matching fields and since it is a downlink traffic from the SGW, no match entry will be found in the flow-table. Therefore, the packet is forwarded to the correct eNodeB using the layer 2 and layer 3 information of the GTP headers (not the inner packet). Downlink traffic from the cloud needs to be returned back to the GTP tunnel, and in order to perform this operation, the switch is configured to match the traffic sourced from the cloud and destined to the UE. The flow-entry instructions represent the operation that should be performed on the packet before sending it back to the network. This instruction includes, tunnel-id and Layer 2 source and destination mac addresses.

The processingUnit module, creates a new control Info object and sets the tunnel-id and the layer 2 information based on the flow-entry instructions. The control info object is attached to the packet and then the packet is sent to the enCap-vPort module. When the packet is received by the enCap-vPort module, the same procedure explained in Section 7.2.1.3C. is performed and the packet is encapsulated to be added again to the GTP tunnel. Doing that makes the eNodeB think it received the message from the SGW. This way the offloading procedure is seamless and the involved LTE entities are totally unaware of the traffic offloaded except for the MME which plays a major role in the setup of the forwarding rules.

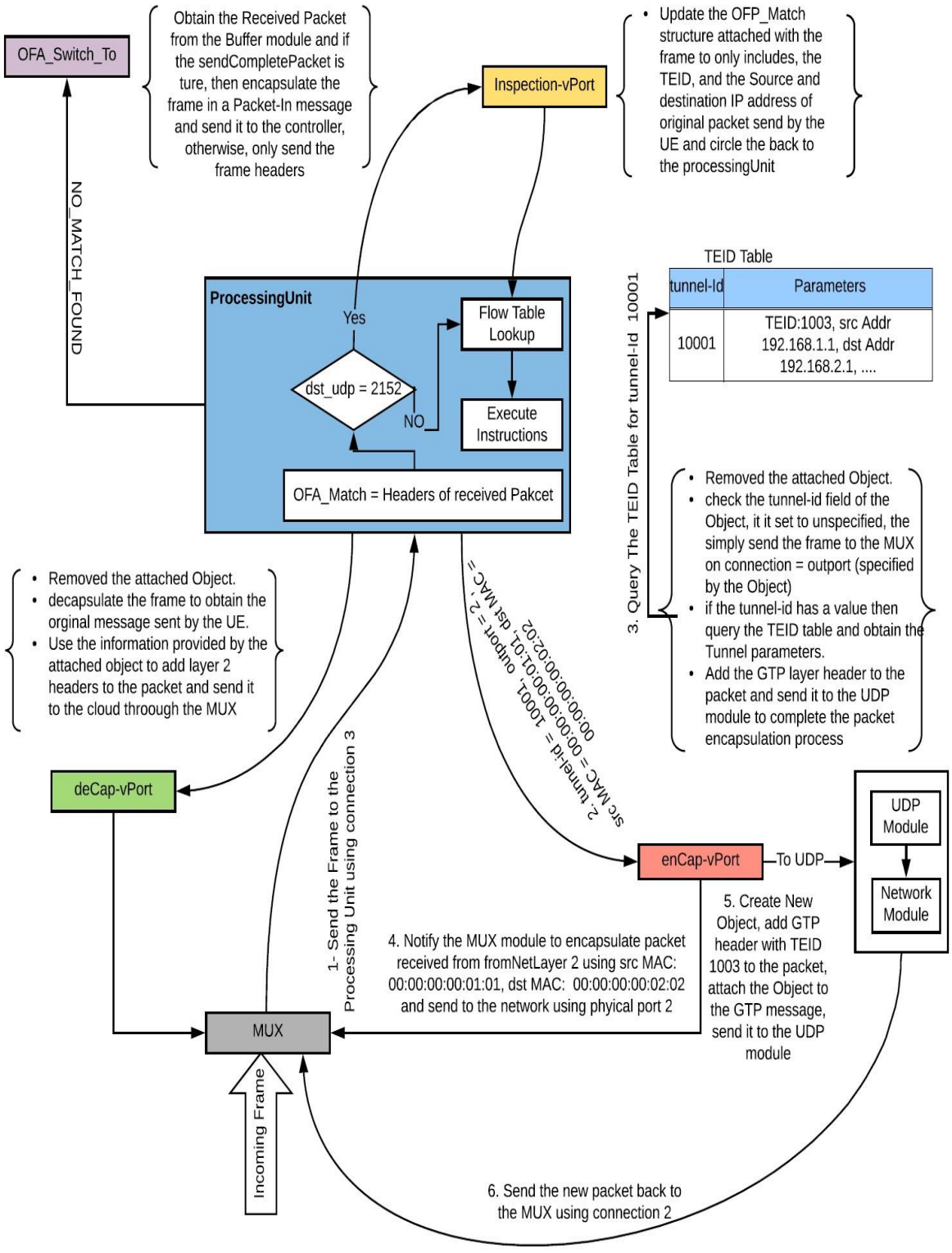


Figure 7.7: Solution 1 - TOS Procedure to return the Cloud traffic back to the GTP tunnel between the SGW and the Serving eNodeB

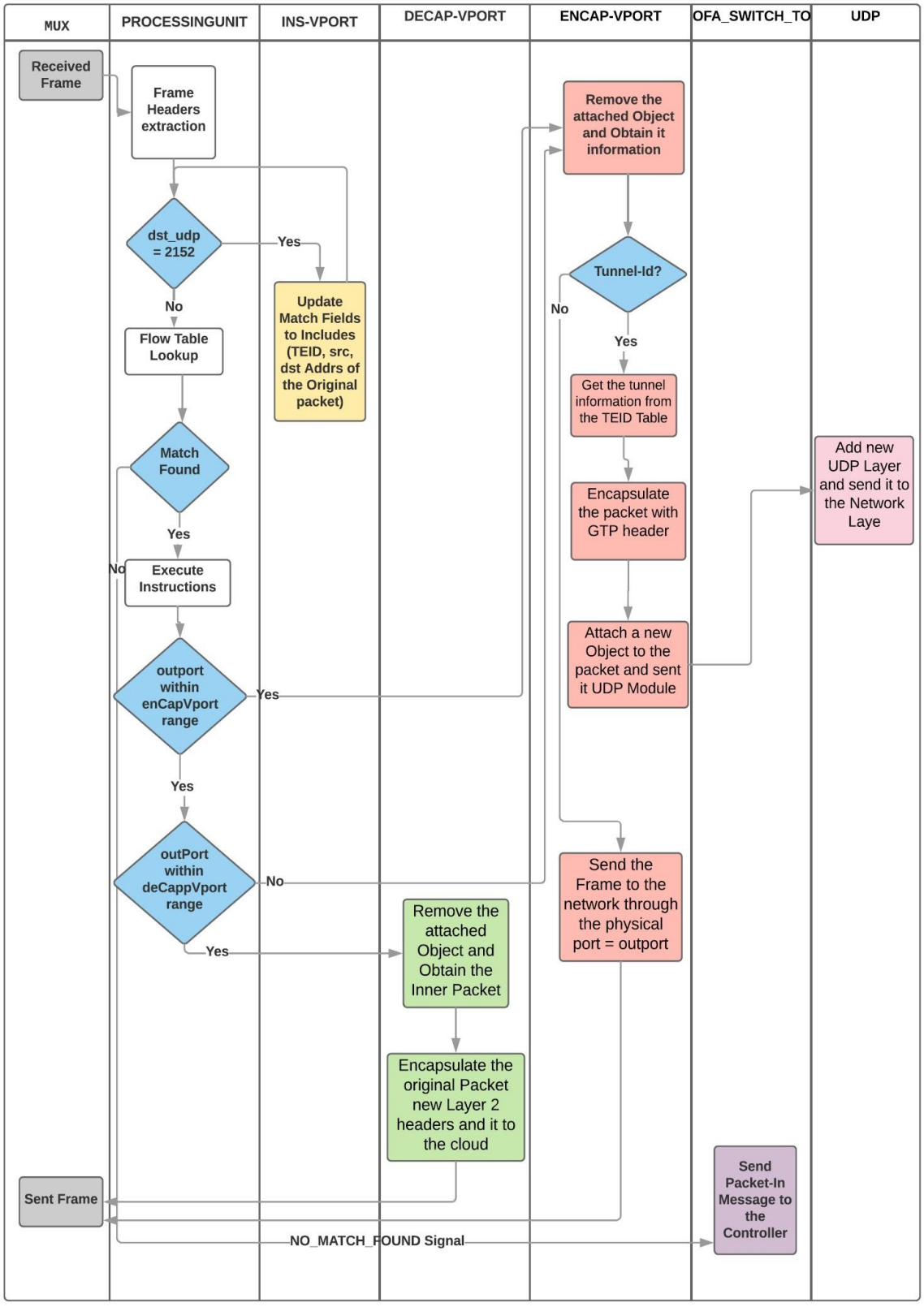


Figure 7.8: Solution 1 - data plane operations

7.2.4 Handover

The procedure that describes the steps required to change the attachment point of a UE form one eNodeB to another is most commonly known as the handover. When the UE changes its location from one place to another and these places under the control of different eNodeBs, a handover procedure is performed. Handover procedure is divided to three phases, namely: preparation, execution, and completion as explained in Chapter 4 Section 4.4.4.1.

The third phase is the focus on this section, in this phase the Target eNodeB (T-eNodeB) notifies the serving MME about the change in the UE location and requests to change the direction of download traffic to the new location. The T-eNodeB sends a S1AP: Path-Switch-Request message to perform that. Upon receiving the message, the MME checks if the UE is registered for cloud traffic offloading, if no, then the normal procedure specified by the 3GPP standard is followed to redirect the traffic to the T-eNodeB where the UE is currently located. If the UE is registered for a cloud traffic offloading, then the SDN controller is notified by the MME about the change of the UE location and based on the information provided by the MME notification message, the SDN controller configures the data plane switch to guarantee that the traffic returned from the cloud-based solution is sent through the correct tunnel to the correct location without causing any service disruption. The handover process involves two possible scenarios. In the first one, both the source and the target eNodeBs are connected to the same Traffic-Offloading-Switch.

Figure 7.9 shows the handover procedure when both eNodeBs are connected to the same Traffic-Offloading-Switch. In this scenario, the SDN controller only needs to re-configure the Traffic-Offloading-switch to add the traffic coming from the cloud to the new tunnel between the SGW and the new eNodeB. This process involves updating the TEID-table to update the GTP tunnel configuration parameters to reflect the new tunnel between the SGW and the new eNodeB, and to do this the SDN controller sends a GTP-Tunnel-Update-Request message to the Traffic-Offloading-Switch. In this message the controller includes, the tunnel-Id and the new GTP tunnel configuration parameters. Upon receiving the message, the offloading switch uses the tunnel-Id against the TEID-table entries to find a matching tunnel configuration, and then the tunnel configuration is updated with the new parameters provided by the GTP-Tunnel-Update-Request received from the controller.

The second scenario is more complicated and involves more than one Traffic-Offloading-Switch. This happens when a UE handover from one eNodeB to another that are connected to two different Traffic-Offloading-Switches, with the assumption that both switches are connected to the same cloud-based solution.

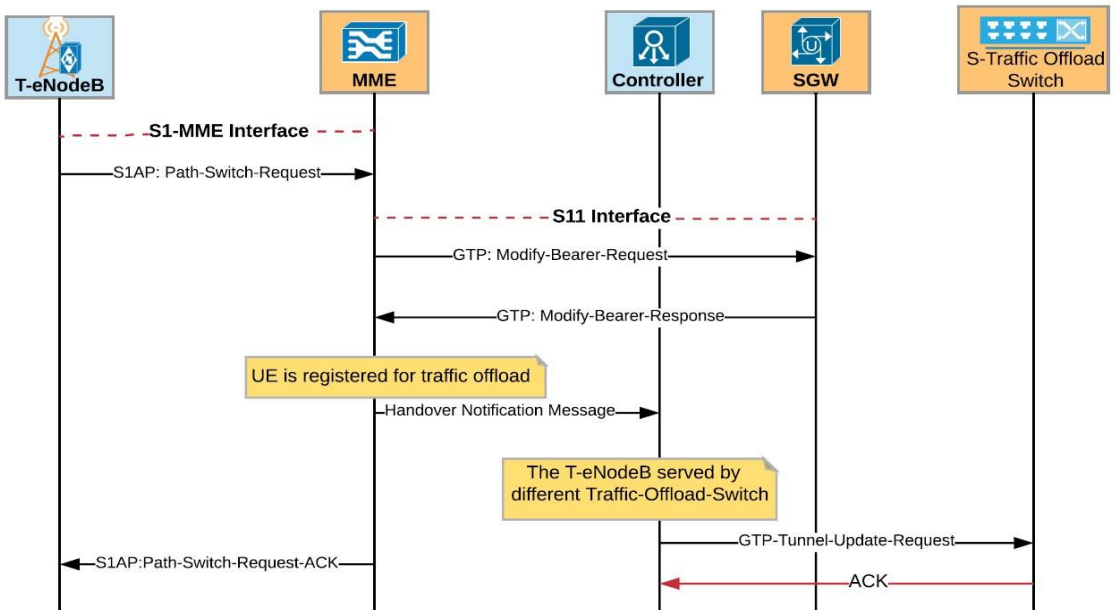


Figure 7.9: Handover Procedure when the source and target eNodeB connected to the same Traffic-Offloading-Switch

Messages are exchanged between the network entities to perform the handover procedure, which is shown in Figure 7.10. In this scenario, the handover procedure is started by the T-eNodeB sending a Path-Switch-Request message to the serving MME to redirect the downlink GTP tunnel of a UE that is registered for the cloud traffic offloading. Upon receiving the message, the MME sends a Modify-Bearer-Request message to the SGW to inform it about the TEID and IP address of the new eNodeB. The SGW responds with a Modify-Bearer-Response message that contains the TEID and IP address that needs to be used by the eNodeB to send UE uplink traffic. Then the MME sends Handover-Notification-Message to the controller to notify it about the UE handover request. Upon receiving the notification message, the SDN controller realizes that the T-eNodeB is connected to a different Traffic-Offloading-Switch, which means that it is necessary to configure both switches.

The process starts by configuring the T-Traffic-Offloading-Switch to perform the cloud offloading procedure. This is done by sending a sequence of GTP-Tunnel-Setup-Request and Flow-Mod messages. Through these messages the controller instructs the T-Traffic-Offloading-Switch to perform the offloading procedure between the UE and the cloud. At the same time, the controller instructs the S-Traffic-Offloading-Switch to report back the UE usage and release its contexts by deleting the flow entries and TEID entries.

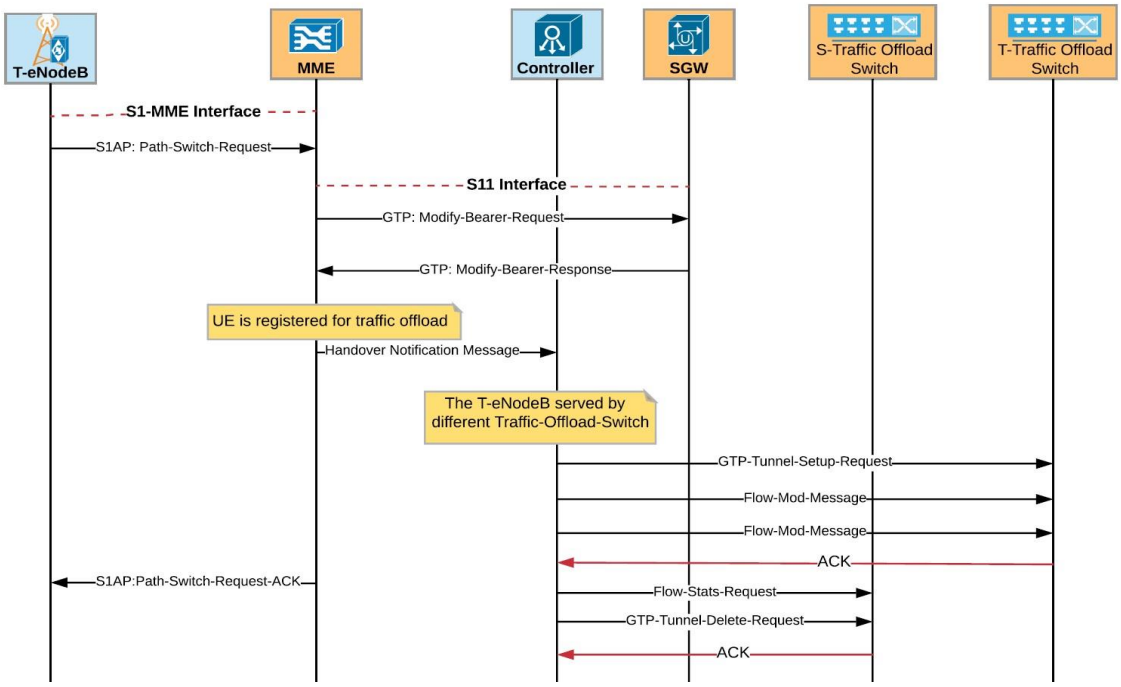


Figure 7.10: Handover Procedure when the source and target eNodeB connected to the different Traffic-Offloading-Switches

7.2.5 Accounting and Charging

In the standard LTE implementation, the PGW is responsible for enforcing policy, applying packet filtering for each user, performing lawful interception and packet screening, and charging support. Considering that the EPS bearers of all UEs terminate at the PGW, this makes it the perfect point to perform the aforementioned operations. At the same time, it is considered the main factor to cause unoptimized routing, and single point of failure. In SDSTO, some of the UE traffic is allowed to be extracted from the S1-U GTP tunnel and sent to the cloud that located in areas near to the mobile access network.

Therefore, it is necessary to have in hand a solution to keep track of the UE traffic sent to the cloud.

In SDSTO design, an SDN controller application is utilized to query the Traffic-Offloading-Switches and obtain the usage of the UEs registered for the traffic offloading to the cloud. The application uses OpenFlow primitives to query the Traffic-Offloading-Switches and uses the statistics of the flow-entries of the UEs to support the accounting and charging operations. When a UE changes its location from one eNodeB to another, and these eNodeBs are connected to different Traffic-Offloading-Switches, the SDN controller obtains the usage of UE from S-Traffic-Offloading-Switches before instructing it to release the UE context. This way the network operator knows the UE's usage that are sent to the cloud and the traffic sent through the EPC, which allows them to apply different treatment if necessary.

7.2.6 Advantage and Drawbacks

This solution performs traffic offloading with minimal change to the standard 3GPP specification of the LTE network. This solution only requires extending the MME operations to communicate with an SDN controller. Although this network design performs the operation perfectly, but it is required to perform many operations that cause more delay. Considering that all the traffic sent over the S1-U interface are GTP-U traffic with UDP port 2152, this means that the switch will inspect all the traffic and considering the ratio of the traffic that should be sent to the cloud is less than the total received traffic.

7.3 Solution 2 Network Design

The second solution is proposed to reduce the unnecessary processing of the traffic that should be just forwarded based on the eNodeB and SGW layer 2 or layer 3 addresses. To implement this solution, a slight modification to the MME and eNodeB modules functionality. This way traffic that should be offloaded to the cloud is tagged and the SDN controller configures the switch to offload only the traffic with a layer 2 tag. In this solution, all the other traffic is simply forwarded to the SGW without any modification, which should reduce the delay of the unnecessary processing. The flow-table of Traffic-

Offloading-Switch is configured to offload UE traffic to a cloud-based infrastructure based on the VLAN tag as shown in Figure 7.11.

7.3.1 Control Plane Setup Procedure

Like the first solution, the MME is connected to the SDN controller and slightly modified to track and maintain the UE bearer information and if the UE is eligible for traffic offloading, then control messages are exchanged between the MME, SDN controller and the eNodeB to setup the control plane. In this solution, few of the standard messages have been modified to allow the MME to inform the eNodeB to add a layer 2 tag after the GTP tunnel headers. The modified messages include the S1AP initial-Context-Setup-Request and Path-Switch-Response messages. Also, a new message has been added to setup the layer 2 tagging on-demand. If subscription-based scenario is used, then the tagging is setup during the bearer setup procedure leveraging the modified messages, and if the on-demand scenario is used, then the new notification message is used to notify the eNodeB. The next sub-sections explain the control plane procedures of both scenarios.

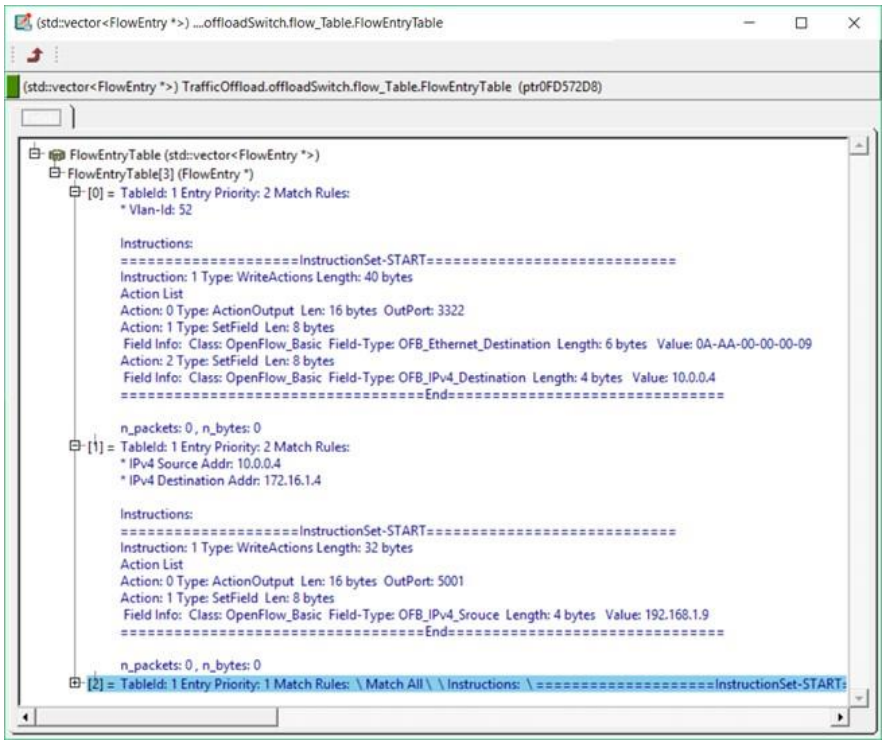


Figure 7.11: flow-table of the Traffic-Offloading-Switch OMNeT++ Module

7.3.1.1 Subscription-Based Scenario

In this scenario, cloud offloading eligibility is specified by the network operator through the subscriber profile, which means that the MME knows if the UE is eligible for the traffic offloading or not. Therefore, in the bearer setup procedure, if the UE is eligible, then the MME waits to receive the Modify-Bearer-Response message from the SGW and sends a Modified-Context-Setup-Request message to inform the eNodeB about the information that it should use to send the UE uplink traffic. This message includes a layer 2 tag. This tag is added to every packet sent using this tunnel. At the same time, the MME sends a Traffic-Offload-Notification message to the SDN Controller, and this message, includes the layer 2 tag, eNodeB TEID, and IP address of the SGW and the serving eNodeB. The layer 2 tag is used by the controller to configure the Traffic-Offloading-Switch to match traffic that includes the tag and extract it from the GTP tunnel and sent to the cloud. Meanwhile the eNodeB TEID and IP address of the SGW and the eNodeB are used to insert the cloud traffic back to the GTP tunnel. This way the Traffic-Offloading-Switch will only extract the traffic that need to be extracted and no unnecessary processing is required.

7.3.1.2 On-Demands Scenario

Most of the operations explained in the previous section are the same and applied in this scenario as well. The main difference is how the MME knows if traffic offloading is required or not. In this scenario, the UE uses its data plane bearer to access the Internet service provider frontend server. Upon receiving the UE request, the frontend server sends an offloading request to the MME, and upon receiving the request, the MME specifies the location of the UE in question and sends a Tag-Request-Notification message to its serving eNodeB. By this message the MME is asking the eNodeB to tag the UE traffic sent over this particular bearer. At the same time, the MME notifies the SDN controller through TrafficOffload-Notification message about the bearer information and the offloading layer 2 tag. Then the controller configures the Traffic-Offloading-Switch.

7.3.2 Data Plane Traffic Forwarding Procedures

This section describes how the Traffic-Offloading-Switch handles the UE uplink and downlink traffic. Traffic-Offloading-Switch can forward the UE uplink traffic to the EPC or the cloud-based infrastructure based on the SDN Controller instructions. In the same way downlink traffic coming from the EPC or the cloud is handled by the switch to make the offloading operation invisible to the LTE entities. The next sub section describes the uplink and downlink forwarding operations.

7.3.2.1 Uplink Traffic

The best way to describe the data plane traffic forwarding is through an example, as shown Figure 7.12, where the uplink data is processed by the Traffic-Offloading-Switch. Figure 7.12 shows how the switch handles the traffic that should be offloaded to the cloud, considering that all the other traffic is handled normally using the standard OpenFlow switch processing. The process starts by receiving a packet from the serving eNodeB (eNodeB 1 in Figure 7.5) and because the cloud traffic is tagged with layer 2 tag (100 in this example), assuming the flow-table is previously configured to match the traffic with tag 100 to be sent to the deCap-vPort.

The same as in the first solution the control Info object is attached to the packet and sent to the deCap-vPort, where the control object includes the number of the physical port and the layer 2 header information to be added to the packet before sending the packet to the cloud.

The deCap-vPort operations are broken down into three simple steps: in the first step, the control info object is extracted from the received packet, then in step 2, the external GTP headers are removed, and the inner packet is obtained, then the third and last step is performed, which is mainly concerned about sending the packet to the right physical port. To perform this step, information from the first step is used to add the layer 2 header to the packet and send the packet to physical port. The mac address of the cloud access entity in Figure 7.5 is used as the destination mac address and a random mac address is used as the source mac address for the traffic sent by a particular UE. The packet is sent to the cloud through physical port 3. Untagged uplink traffic is handled by using flow-entry 3 with

priority 700, and by leveraging this flow-entry, the traffic is simply forwarded to the SGW through physical port 1.

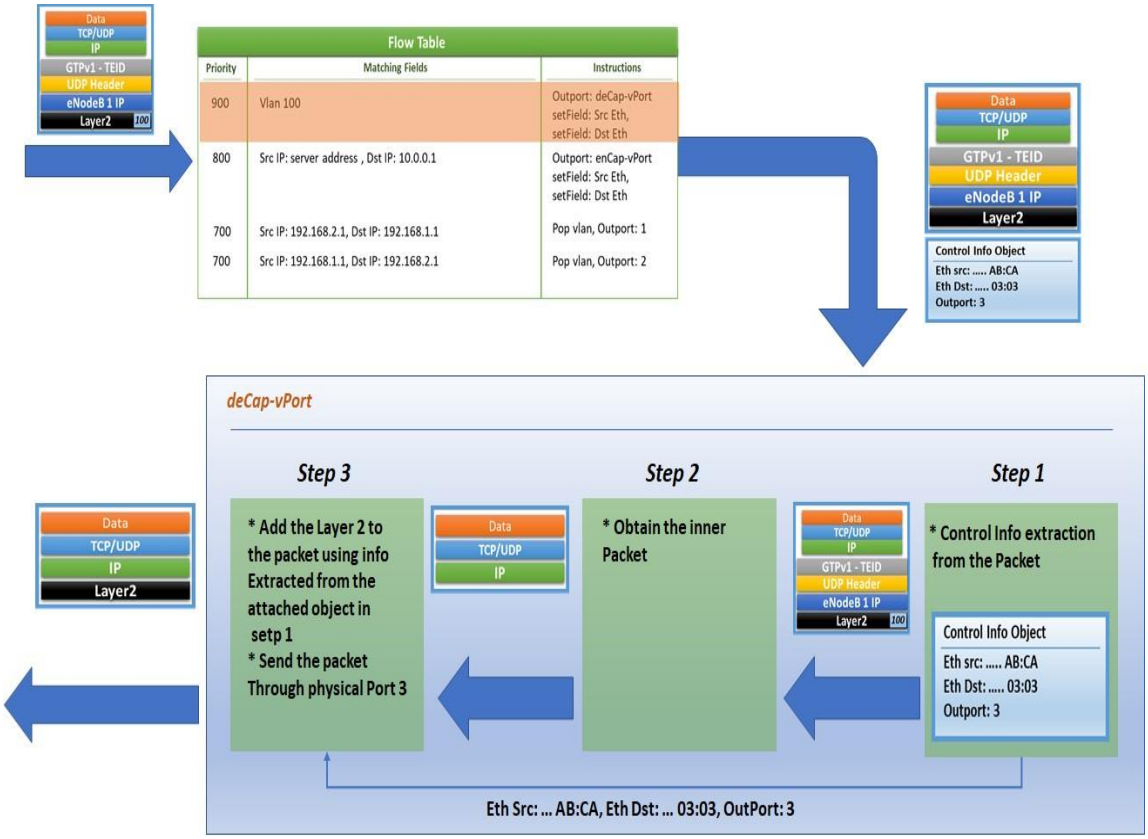


Figure 7.12: Solution 2 TOS - Data Plane Uplink Traffic Processing Procedure

7.3.2.2 Downlink Traffic

Downlink traffic, as previously mentioned can be received from either the SGW or the cloud. In the first case, the traffic is simply forwarded to the correct serving eNodeB without any modification, while the second case represents the cloud traffic sent back to the UE, where this traffic needs to be added back to the GTP tunnel before it is sent to the serving eNodeB.

The procedure used by the Traffic-Offloading-Switch to handle the downlink traffic, is shown in Figure 7.13. Downlink traffic received from the SGW is handled by flow-entry 4 by forwarding the traffic to the eNodeB through physical port 2 without modifying anything. The other traffic received from the cloud are handled by flow-entry 2 with priority 800. Through this flow-entry the cloud traffic is sent to the enCap-vPort to add the traffic back to the GTP tunnel.

The enCap-vPort includes three steps to perform its operations. The first step is the same in both enCap-vPort and deCap-vPort which involves the removal of the control info object from the received packet. Although the information contained in this object maybe different in each case, after removing the control object, the second step is started. In the second step the layer 2 header is removed from the packet, then the most important operation is performed in third step. In this step the tunnel-Id obtained in step 1 is used to query the TEID-table to find the GTP tunnel configuration parameters and based on the result, new GTP, UDP, and IP layers are added to the original packet, then the layer 2 information obtained in step 1 is used to add layer 2 to the packet before sending it to the physical port. As shown in Figure 7.5, TEID 1003 is used in the GTP layer, and UDP used as the transport layer protocol with 2244 and 2152 as the source and destination port numbers respectively. In the network layer header, 192.168.1.1 and 192.168.2.1 represent the SGW and eNodeB1 IP addresses respectively are used as the source and destination IP addresses. The same with the layer 2 header since SGW mac address is used as the source mac address and eNodeB1 mac address is used as the destination mac address.

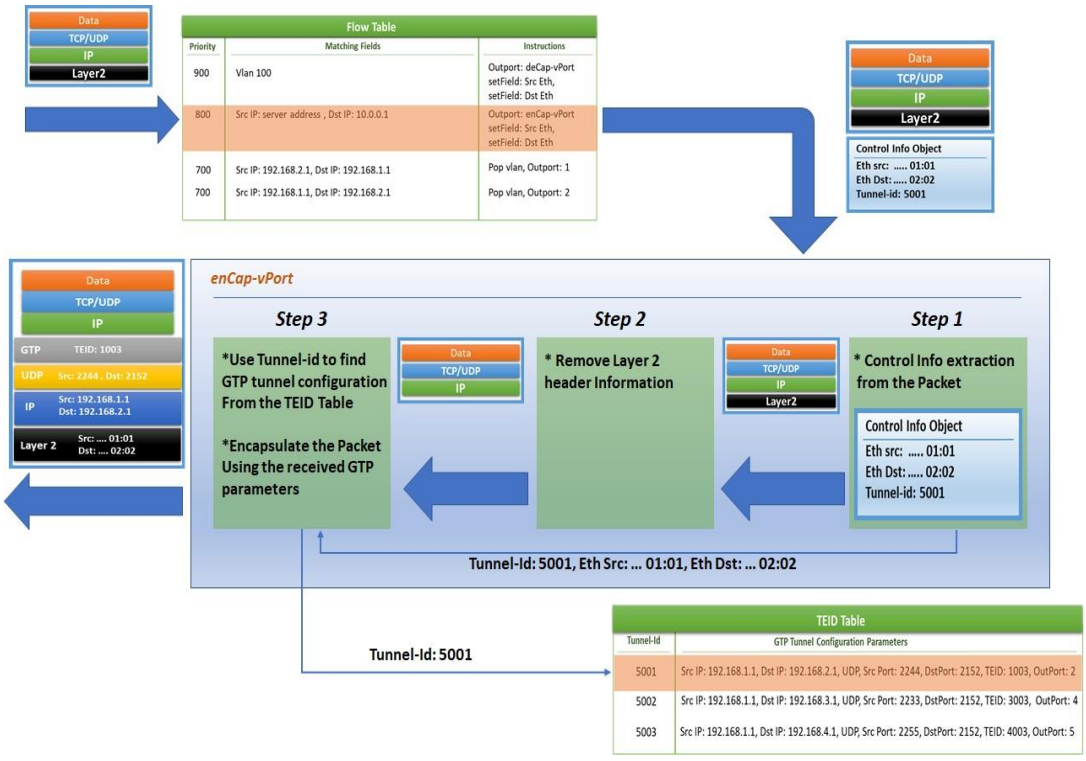


Figure 7.13: Solution 2 TOS - Data Plane Downlink Traffic Processing Procedure

7.3.3 Handover

The handover procedure is explained in Chapter 4 Section 4.4.4.1. This section describes how the proposed solution handles the change of the UE location without causing any service disruption and redirects the downlink traffic coming from the EPC and the cloud-based infrastructure to the new location of the UE. The redirection of the downlink traffic coming from the cloud-based infrastructure requires the SDN controller to know if the source and target eNodeBs involved with the UE handover procedure are connected to the same Traffic-Offloading-Switch or not.

If both eNodeBs are connected to the same Traffic-Offloading-Switch, then the SDN controller will only update the downlink flow-entry considering that the MME will notify the target eNodeB to tag the uplink traffic of UE's bearer eligible for the cloud offloading. In this case, the SDN controller sends a GTP-Tunnel-Modify-Request to Traffic-Offloading-Switch to update the TEID-table. The message includes the information of the new GTP tunnel that has been agreed on between the SGW and the target eNodeB. This way UE traffic coming from the cloud is sent to the target eNodeB where the UE is currently attached.

If the source and the target eNodeBs are connected to two distinct Traffic-Offloading-Switches, then it is necessary to configure the new Traffic-Offloading-Switch and ask the old Traffic-Offloading-Switch to release the UE context. It is also possible to ignore the old Traffic-Offloading-Switch and let the ideal time out of the flow entries to take care of the flow deletion process. This approach can cause a problem and may require a special application to process the UE statistics to build a historical view of the UE activity to come up with the best ideal time-out for each entry. The SDN controller configures the new Traffic-Offloading-Switch following the same procedure explained in Section 7.3.1, which includes sending GTP-Tunnel-Setup-Request followed by two Flow-Mod messages to configure the switch TEID and flow-tables to properly handle the UE traffic sent over a specific bearer in both uplink and downlink directions.

7.3.4 Accounting and Charging

There is no change between this solution and solution 1 in terms of accounting and charging procedure. The SDN controller is equipped with a specialized application to

monitor the UE activities and gather usage statistics of all the traffic sent to the cloud instead of the mobile network EPC. This information is then used by a centralized application to help enforce policy and charging. The Monitoring application keeps tracking the UE statistics as long as it is registered to the SDN controller for the cloud offloading. The application uses OpenFlow multipart messages to query to the Traffic-Offloading-Switch to obtain the statistics. The application is also notified when the UE changes it point of attachment. Normally no action is required if the UE is still served by the same Traffic-Offloading-Switch, but if the movement of the UE requires changing the Traffic-Offloading-Switch, then few steps are performed by the application to make sure that the information it has about the UE are correct and up to date.

7.3.5 Advantage and Drawbacks

This solution helps to reduce the OpenFlow switch processing time required to correctly forward the received traffic to the next hop. In this solution, it is not necessary to inspect each and every received packet like the first solution. Instead traffic eligible for the cloud offloading is tagged by the eNodeB and the OpenFlow switch just decapsulates the packet and sends it to the cloud and inserts the return traffic back to the GTP tunnel before sending it to the serving eNodeB. At the same time, this solution requires more modification to the EPS entities compared to the first solution because modification to the eNodeB structure and some of the S1AP messages are required to model this solution.

7.4 Solution 3 Network Design

This solution is based on the concept of eliminating the needs for GTP tunnelling and utilizes L2-based mobile backhaul [44]. Before starting with the Traffic offloading solution, it is important to understand the backhaul network design. Figure 7.14 shows the logical network topology of this solution, where the backhaul network is composed of two parts, namely access and the aggregation sub-networks. The access sub-network is responsible for providing forwarding paths for a group of eNodeBs and aggregation sub-network connects several access sub-networks to the mobile core network. SDN based network is used in the access sub-network and Carrier Ethernet is used in the aggregation sub-network.

In this solution the eNodeB MAC address is used as the UE locator address and the UE IP address, which is assigned during the initial attachment procedures, is mainly used as the UE identifier. In order to increase the scalability of the proposed solution VLAN stacking with 802.1ad frames is used [21]. At the same time, the SDN controller is equipped with a Local Mobility Management application that is responsible for handling the change of the UE location as long as it is still connected to the same backhaul access sub-network. This approach keeps the EPC unaware of the UE movement and reduces the signalling messages sent to the core network. To implement this solution, the controller of the backhaul access sub-network keeps an IP-MAC address mapping between the UE address and the MAC address of its serving eNodeB. Also, the controller configures the switch to change the source MAC address of the traffic coming from all the eNodeBs connected to it with its own MAC address. The destination IP address of all the traffic received from the core network is used to find the MAC address of the serving eNodeB and the output port number.

The core network is only aware of the MAC address of the access switch, which makes the UE movement between the eNodeB groups connected to the same backhaul access switch not relevant to the core network and does not need to do anything about it. Instead, the Local Mobility Management application updates the IP-MAC mapping to forward the UE traffic to the new location. To support selective traffic offloading, a new cloud offloading application is modelled and added to the SDN controller.

This application is responsible for handling the offloading request messages from the MME. At the same time, it cooperates with the Local Mobility Management application to handle UE handover. The next sub-sections, first describe the process of triggering cloud offloading and the way to configure the access switch to enforce the offloading traffic , then describe how the access switch handles the uplink and downlink traffic, followed by the explanation of the handover procedure with more focus on how the controller applications work together to redirect both EPC and cloud downlink traffic to the new location of the UE, and finish with the advantages and disadvantages of this solution.

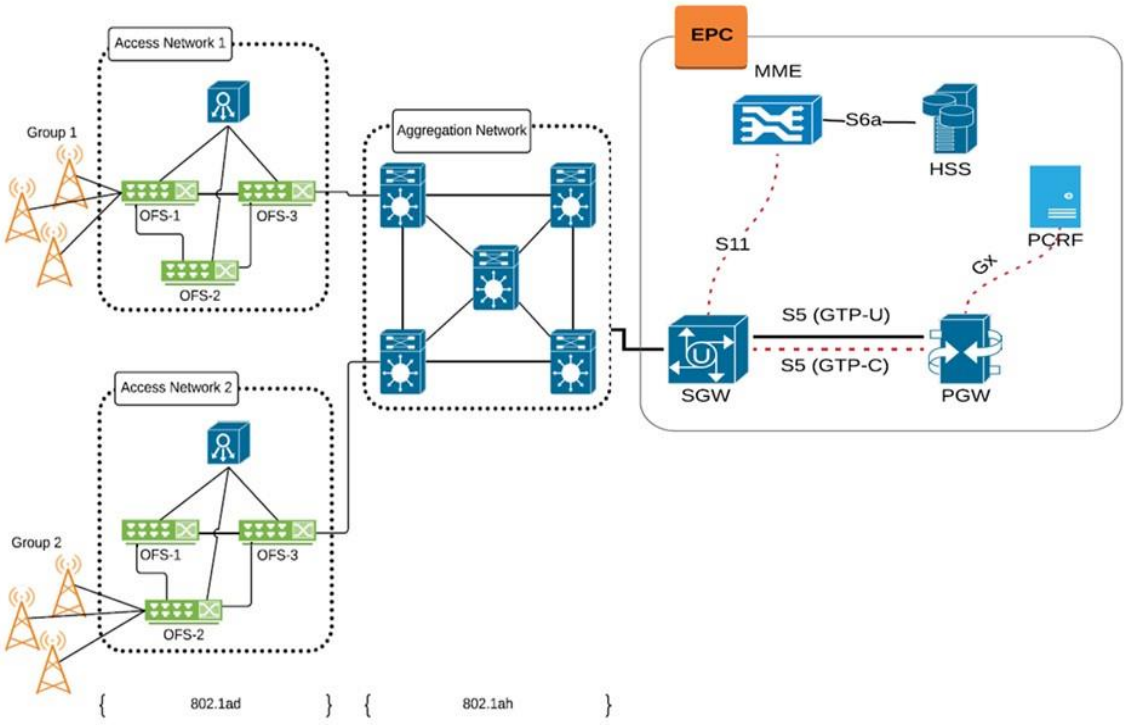


Figure 7.14: Solution 3 - Logical Network Topology

7.4.1 Control Plane Operations

On-demand and subscription-based offloading represent two possible ways to trigger the selective Traffic Offloading operations. The main difference between these ways is when the SDN controller of the backhaul network is notified to configure the access switch to perform the cloud offloading operations. On-demand is more flexible because the offloading request can be sent at any time from the service provider to the mobile core network (the MME). Subscription-based access is easy and can be configured by the network operator, but it is less flexible than the On-demand way.

In this section, the process of network configuration to send some of the UEs' traffic to the cloud after the reception of the offloading request from the MME (whether it is configured or sent by the service provider) is explained in detail. The subscription-based process starts immediately after finishing the Service-Request-Procedure, while the On-demand process starts after receiving the offloading request from the service provider, as shown in Figure 7.15. The MME needs to notify two entities to implement the UE traffic offloading, precisely, the eNodeB and the SDN Controller. The MME starts by asking the eNodeB to

tag UE traffic either by using Modified version of the Context-Setup-Request or Tag-Request-Notification based on whether the network is configured to support On-demand or Subscription-based offloading. Then, the MME sends a Traffic-Offload-Notification message to the backhaul access sub-network controller. In this message the MME includes the protocol type, transport layer designation port number, source and destination IP address, cloud-Solution IP address, and VLAN Id. The MME sends these messages through the backhaul network by utilizing the control plane VLAN id.

The backhaul access sub-network switch is configured to send all the traffic tags with the control plane VLAN to the SDN controller. Therefore, the Context-Setup-Request or Tag-Request-Notification and Traffic-Offload-Notification messages are sent to the SDN controller first. The procedure used by the controller to handle the received message is shown in Figure 7.16. The process starts by checking the received message type and based on the result an application is triggered to handle the received message. In this case the message sent by the MME to the eNodeB is handled by the switching application to simply forward the message to the correct eNodeB.

On the other hand, Traffic-Offload-Notification is handled by the cloud offloading application. In the application the packet handler is the one responsible for the organization of the job performed by application modules. First the packet handler module asks the UE store module to add a new entry for this particular UE, then it requests the FlowWriteService to send two flow entries to configure the access switch to offload UE traffic sent over a specific layer 2 tunnel to the cloud-based infrastructure and delivers the response to the correct location of the UE. To help understand the offloading procedure let us assume that UE 1 is connected to eNodeB1, where the latter is connected to access switch 1 to be connected to the mobile core network.

During the Initial-Attachment-Procedure, the IP address 10.0.0.1/24 is assigned to UE1. At the same time, the network operator has configured UE1 to be eligible to offload all of its TCP traffic with destination port 4545 to a cloud-based infrastructure with the IP address 20.0.0.1/24. Moreover, VLAN 100 is used as the cloud offloading VLAN. The backhaul access sub-network controller configures the access switch to offload UE1 traffic after receiving the Traffic-Offload-Notification message from the MME. The controller configures access switch 1 by sending Flow-Mod message. This message includes the

following matching fields (VLAN id = 100, source IP address = 10.0.0.1/24, Protocol = TCP, Destination port number = 4545) with instruction to:

- Remove the VLAN Id
- Change the source and destination MAC address to Switch 1 and Cloud router MAC addresses respectively.
- Change the destination IP address to 20.0.0.1/24.

This message is sent by the controller to configure the switch to handle the uplink traffic, and another Flow-Mod message is sent to it with the following matching fields (destination IP address = 10.0.0.1/24, Protocol = TCP, source port number = 4545) with the instruction to:

- Add new VLAN Id = 100 to the received frame
- Change the source and destination MAC address to the Switch 1 and eNodeB1 MAC address respectively.
- Change the source IP address back to 20.0.0.1/24.

This way TCP traffic with destination port = 4545 is sent to the cloud instead of the core network and the response is sent back to the UE without any knowledge of the offloading process.

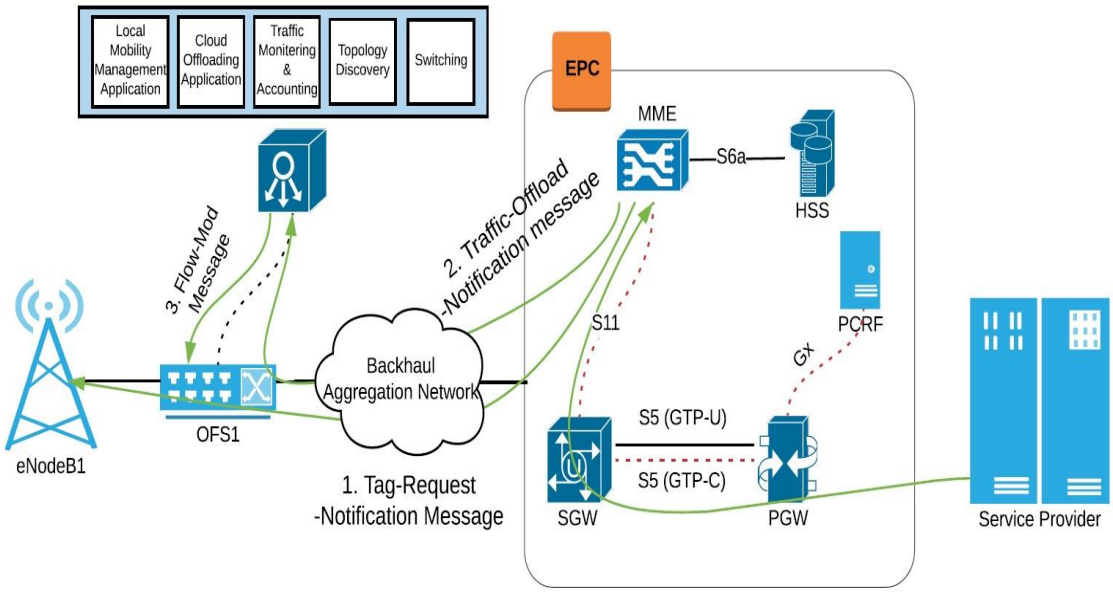


Figure 7.15: Solution 3 - Control Plane Operations

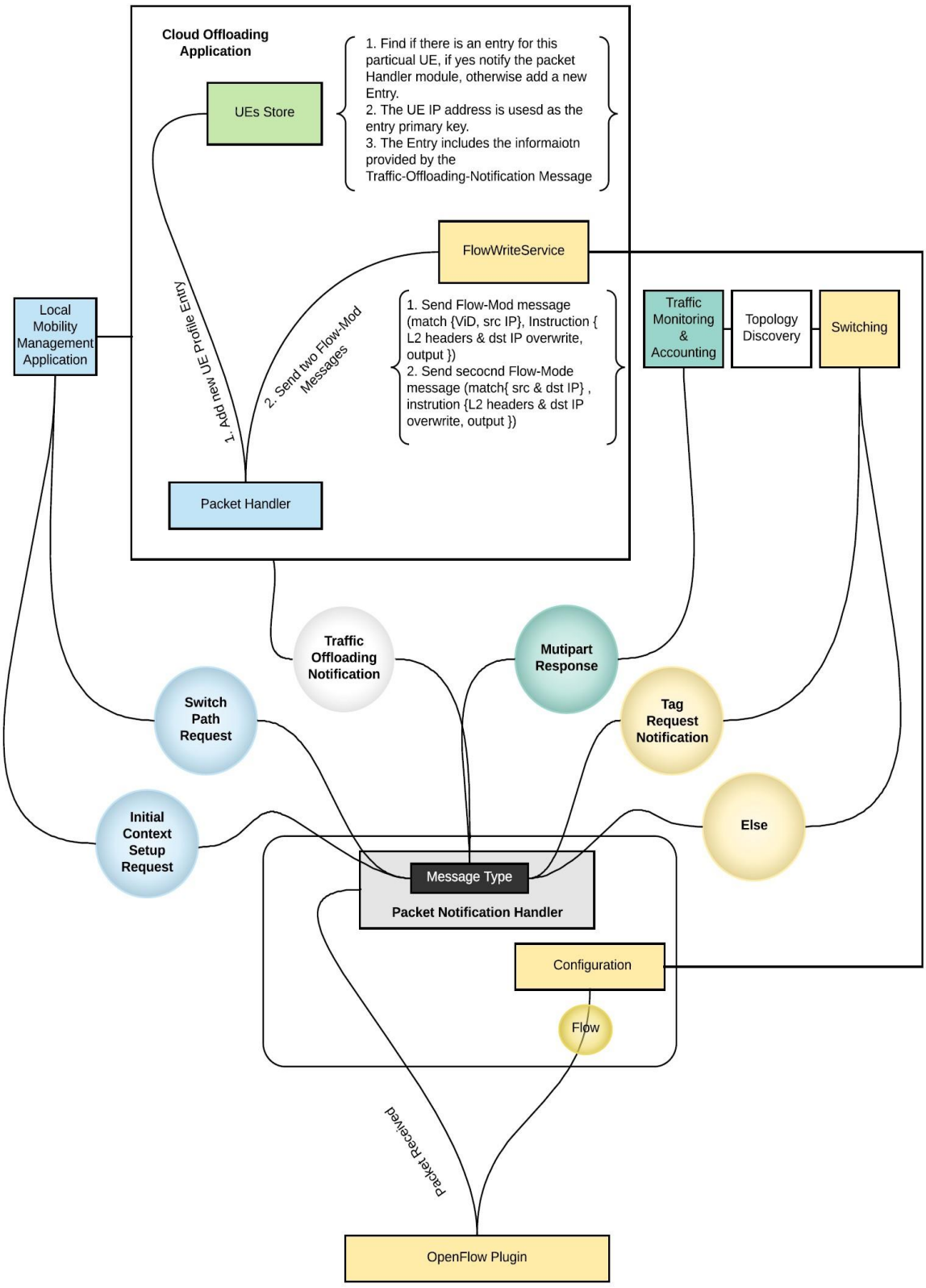


Figure 7.16: Solution 3 - Cloud Offloading Application Operations

7.4.2 Data Plane Traffic Forwarding Procedures

This section explains how uplink and downlink traffic is normally forwarded between the EPC and the EUTRAN and how the traffic of a specific UE is sent to cloud-based infrastructure that assumed to be available in near proximity to the EUTRAN.

7.4.2.1 Uplink Traffic

Uplink Traffic refers to the operation of delivering UE traffic to the correct destination. This normally explains the operation required to send the UE traffic through the core network to the desired destination specified by the packet sent by the UE. This section also describes the operation to send a selective traffic to a cloud-based architecture that assumed to be located near to the access network.

As shown in Figure 7.17, UE 1 is trying to send two different TCP traffic, one of them is specified by the network to be offloaded to a cloud-based infrastructure. Flow path number 1 with black arrow represents UE1 traffic (HTTP traffic that has TCP destination port number equal 80) that should be handled normally by sending the traffic through the UE default bearer to its SGW. While the flow path number 2 with a blue arrow represents UE1 traffic (traffic with TCP destination port number equal to 4545, whose random port number is used to identify the service eligible for cloud offloading) that needs to be offloaded to the cloud-based infrastructure. UE uplink traffic is sent through the air interface to the serving eNodeB. In the latter, layer 2 header information is added to the UE packet and sent to the backhaul access sub-network, and the eNodeB uses its own MAC address as the source and the SGW MAC address as the destination. The access switch enforces the SDN controller rule of whether to send the received traffic to the SGW or offload it to the cloud infrastructure.

Section 7.4.1 explains how different parts of the network works together to implement the traffic offloading solution. Assuming that the access switch (OpenFlow Enabled switch) is configured to offload UE 1 TCP traffic that has destination port number equal to 4545 to the cloud, and all the other traffic should be forwarded normally to the SGW. When the access switch has received an uplink traffic from the eNodeB connected to it, it first, checks the flow-table looking to see if it has a flow-entry to offload the received traffic to the cloud. If no match is found, then the packet is sent to next hop after changing the source

MAC address with its own MAC address. Otherwise, if it has a flow-entry that matches the received traffic characteristics, then the instruction attached to the matched entry is executed on the packet before sending it to the cloud.

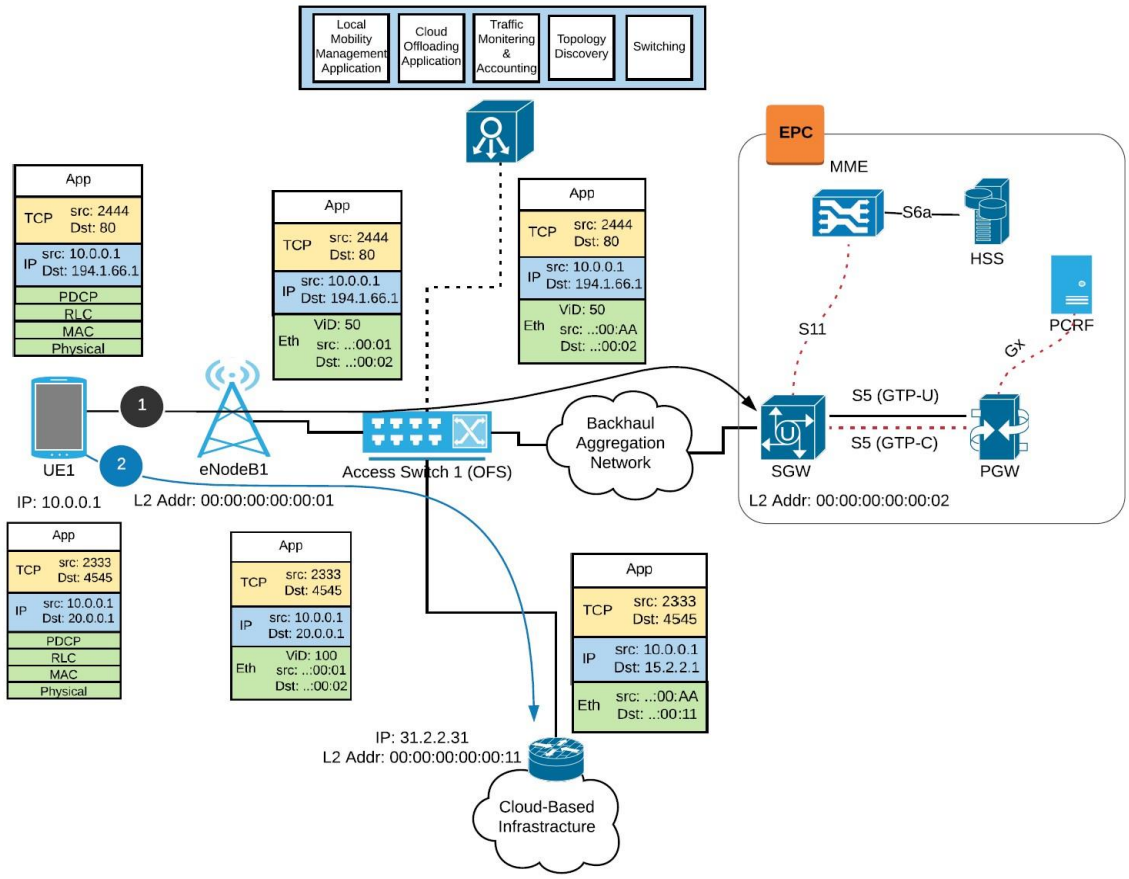


Figure 7.17: Solution 3 - Uplink Traffic Operations

7.4.2.2 Downlink Traffic

In this solution the SGW is modified to map S5/S8 GTP tunnel to layer 2 tunnel by utilizing the IEEE 802.1ad technology. Normally, Downlink traffic represents the traffic sent from EPC to the UE. In this solution the EPC sends the traffic destined to a UE to the MAC address of the access switch to which the UE serving eNodeB is connected. With the cloud-based offloading, downlink traffic can either be sent by the EPC or the cloud-based infrastructure as shown in Figure 7.18. In the first case, EPC traffic is sent through the backhaul network utilizing the 802.1ad technology and uses the backhaul access sub-network switch MAC address as the destination MAC address. Upon receiving the frame, the access switch uses the packet destination IP address to find the correct output port, then

it adds a new layer 2 header before sending the frame to the serving eNodeB. In this Header, the SGW MAC address is used as the source MAC address and serving eNodeB MAC address is used as the destination MAC address. In the second case, the access switch receives downlink traffic from the cloud-based infrastructure. The process of configuring the backhaul access switch to handle cloud downlink traffic is explained in Section 7.4.1. This section explains how the switch processes the traffic coming from the cloud-based infrastructure assuming that it is already configured and has all the required flow-entries to correctly forward the received cloud traffic to the current location of the UE. Upon receiving a frame from the cloud, the destination IP address along with the transport layer protocol and source number is used to identify the traffic, then the traffic is returned back to the L2 tunnel by adding a VLAN Id to the frame after updating the source IP address and the source and destination MAC address. Then the frame is sent through the correct physical port to which the UE serving eNodeB is connected.

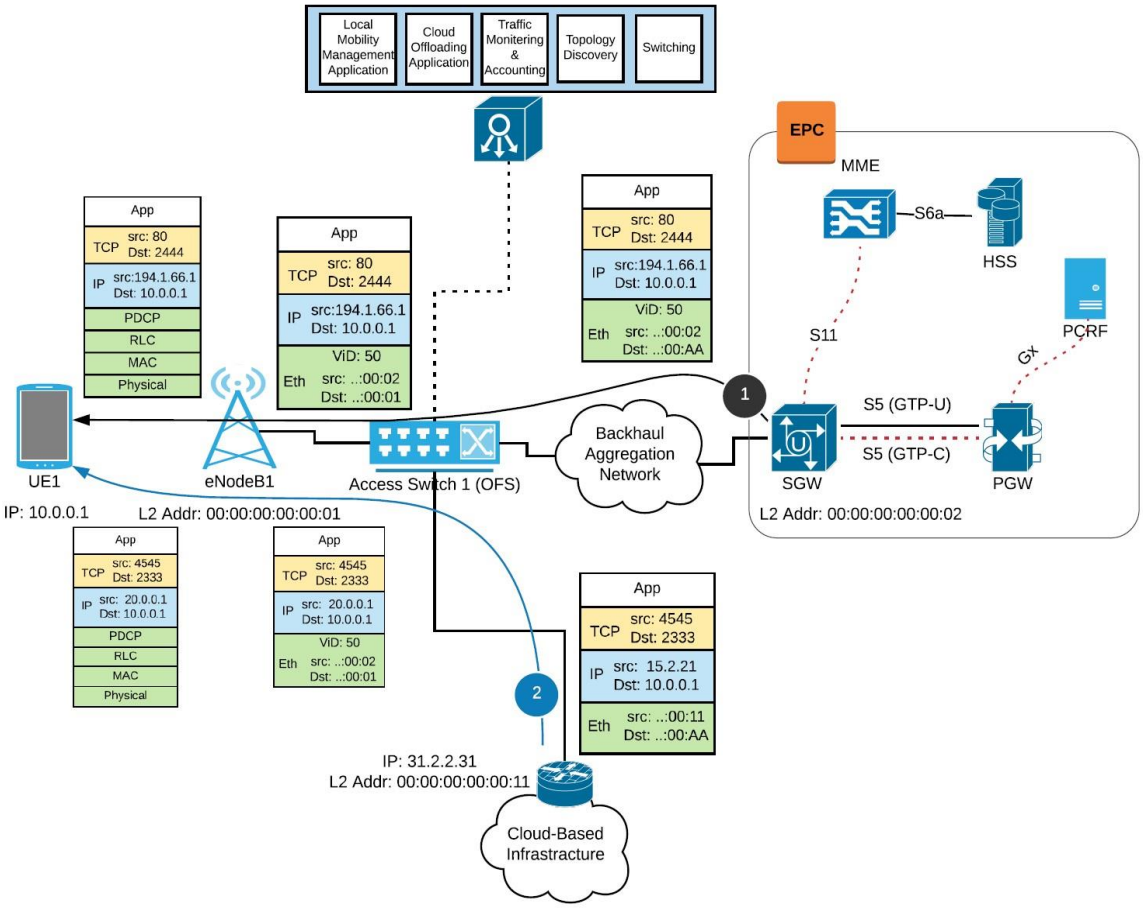


Figure 7.18: Solution 3 - Downlink Traffic Operations

7.4.3 Handover

This section briefly describes two possible scenarios of the X2 handover when the SGW and the MME are kept the same. As mentioned in Section 7.4, the backhaul access sub-network's controller is equipped with two main applications, namely the Local Mobility Management (LMM) and the Cloud OFFloading (COFF) Applications. First, the interaction between these applications is explained to demonstrate the process of downlink traffic redirection to the correct eNodeB without any involvement from the core network (when both the source and target eNodeBs are connected to the same access sub-network switch or switches that under the controller of the same SDN controller). Then the handover procedure between two eNodeBs that are connected to a different access sub-networks is briefly described. 3GPP LTE X2-based handover procedure has been previously explained in Chapter 4 Section 4.4.4.1.

In this solution, control signalling messages are sent over a specific VLAN and all the access switch is configured to send this traffic to the SDN Controller. Let us assume that UE1 with IP address 10.0.0.1/24 has performed handover from eNodeB1 with MAC address 00:00:00:00:00:01 to eNodeB2 with MAC address 00:00:00:00:00:02. Both eNodeBs are connected to access switch 1 through port 1 and 2 respectively. In the handover completion phase, the Path-Switch-Request message is sent by the eNodeB2 to notify the MME about the new location of the UE and request downlink traffic redirection. This message is intercepted by the access switch and sent directly to the SDN controller.

As shown in Figure 7.19, the message is passed by the Packet notification to the Local Mobility Management application. The latter lookups the UE in the UE profile store, and if no match is found, then the application understands that the UE was connected to another access network, which is not under its control. Therefore, the message is forwarded to the MME to handle it as normal. At the same time, it keeps monitoring the control messages to update the UE profile store with the UE information. In our example both eNodeB1 and eNodeBs are connected to the same access network through access switch 1. Therefore, the packet handler is used to update the UE profile in the local UE profile store, then it sends a handover notification message to the Cloud Offloading application and re-configures the access switch flow-tables to redirect the traffic to the target eNodeB (eNodeB2). This configuration is related to the downlink traffic sent by the SGW from the

core network. When the Cloud Offloading application receives the handover notification message, it first checks if the UE is registered for the traffic offloading services, and if it is not registered, then nothing needs to be done and the application ignores the received notification message. Otherwise, the application updates the switch flow-entries to send the cloud traffic to eNodeB2 through port 2 instead of eNodeB1 through port 1.

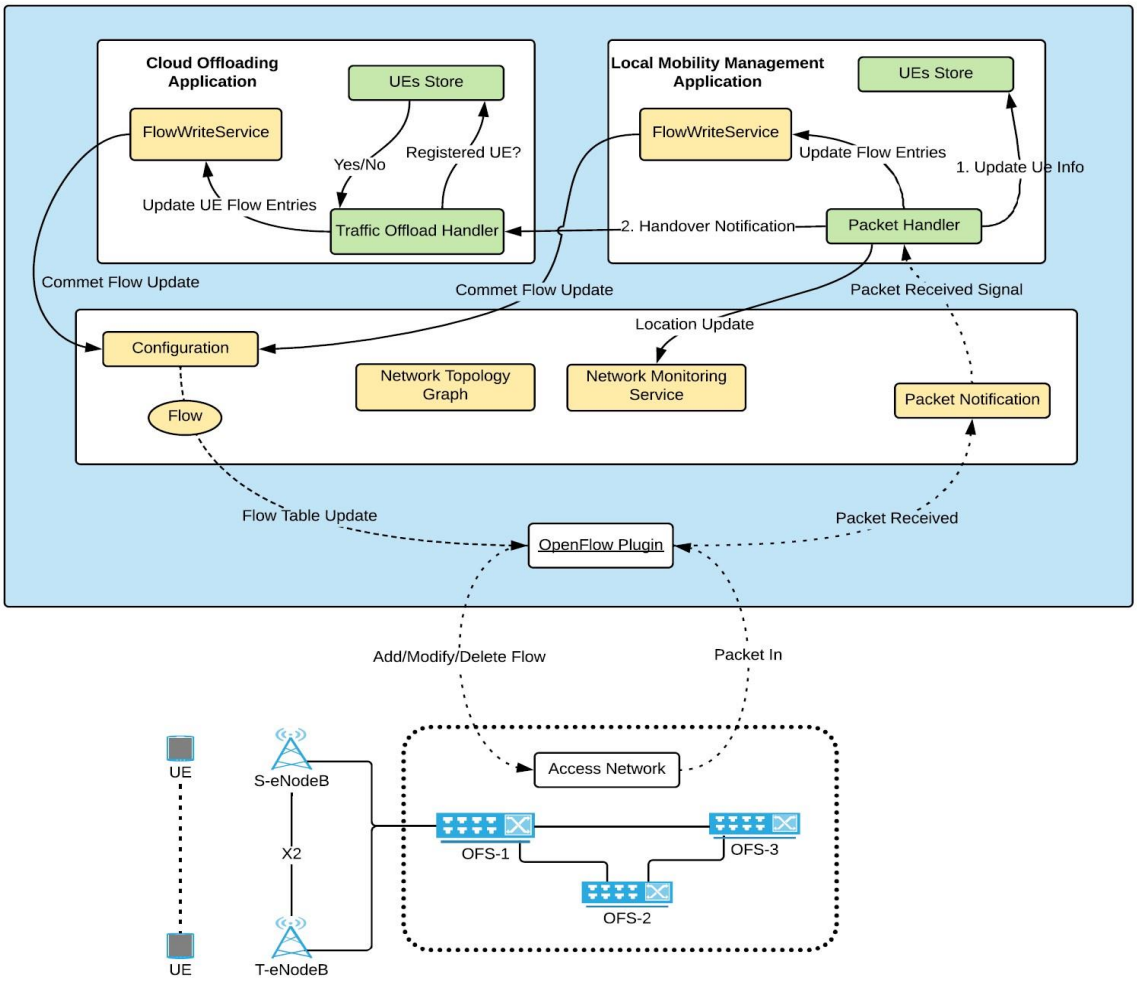


Figure 7.19: Solution 3 - UE Handover Procedure

7.4.4 Accounting and Charging

This solution is no different from the previous solutions, because it is necessary to have a mechanism to track the UE usage regarding the traffic sent to the cloud. The SDN controller has an application specially built to keep track of the UE activity within its region of control. The application utilizes OpenFlow multipart messages to obtain statistics

of the UE flow entries. The Accounting and Charging application periodically updates the UE profile store. These statistics are used by the controller applications to enforce a policy to drop UE traffic after its allowance has been exceeded. Statistics gathered by the Accounting and Charging are then fed to a centralized Monitoring and Charging application to keep track of the UE usage when its handovers from one access network to another.

7.4.5 Advantage and Drawbacks

This solution helps to reduce the control signalling loads by keeping the UE movement away from the mobile core network and handles it locally through the use of the SDN controller in the backhaul access sub-network. The offloading procedure is simple and does not require any change to the OpenFlow 1.3 switch specification because there is no GTP inspection or encapsulation/decapsulation required. This reduces the processing time required to decide whether to send the traffic to the EPC or offloaded it to the cloud. At the same time, it requires almost all the EPS entities to support the Layer 2 based backhaul and the traffic offloading operations, which is a heavy price to pay.

7.5 Simulation Scenario

To evaluate the proposed solutions, a simulation environment has been developed for each solution. The simulation project is built on OMNeT++ 4.3 and uses INET 2.3, simuLTE and OpenFlow 1.3 as preference projects (which mean that all the modules built by the aforementioned libraries can be used in the simulation project known as SoftwareDefindEPC shown in Figure 7.21.

The SoftwareDefindEPC project is used to create a simulation platform to determine the impact of the traffic offloading solutions. The network topology used in our simulation is shown in Figure 7.20. It consists of: Access, Backhaul, and Core networks. The access network consists of two eNodeBs to which random number of UE are connected (the number of UEs specified in each simulation run). The backhaul network is simplified and is simulated by an SDN-based network that is mainly composed of a single OpenFlow controller and modified OpenFlow switches (named Traffic-Offloading-Switch, which is responsible for forwarding the traffic coming from a group of eNodeBs to the EPC and

vice versa). Cloud-based infrastructure is distributed in the backhaul and connected to the Traffic-Offloading-Switch. The core network consists of multiple important entities that include the MME, SGW, and PGW. The latter is connected to the Internet through the SGi interface which is simulated as a Point-to-Point interface. The HSS or PCRF are not implemented in our simulator. Instead, a simplified version of the functionalities offered by these entities are included as part of the MME and PGW respectively.

The Internet is simulated by a single OMNeT++ standard Host (InternetHost). In the simulation environment the Traffic-Offloading-Switch, which is a modified version of the standard OpenFlow 1.3 switch, is used in solutions 1 and 2 and normal OpenFlow 1.3 switch is used in solution 3.

The MME entity is modified to include new features that help implement the traffic offloading solution. The design and the structure of the standard MME entity is explained in Chapter 4 Section 4.5.1.3. In order to make the simulator work like a normal EPS network and at the same time offer the possibility to simulate cloud-based offloading solution, the mmeApp module is modified to include a new module named sdnAgent module. Also, a few configuration parameters are added to specify the operation of the MME entity. This section explains in more detail the changes and the configurations required to change the MME behaviour.

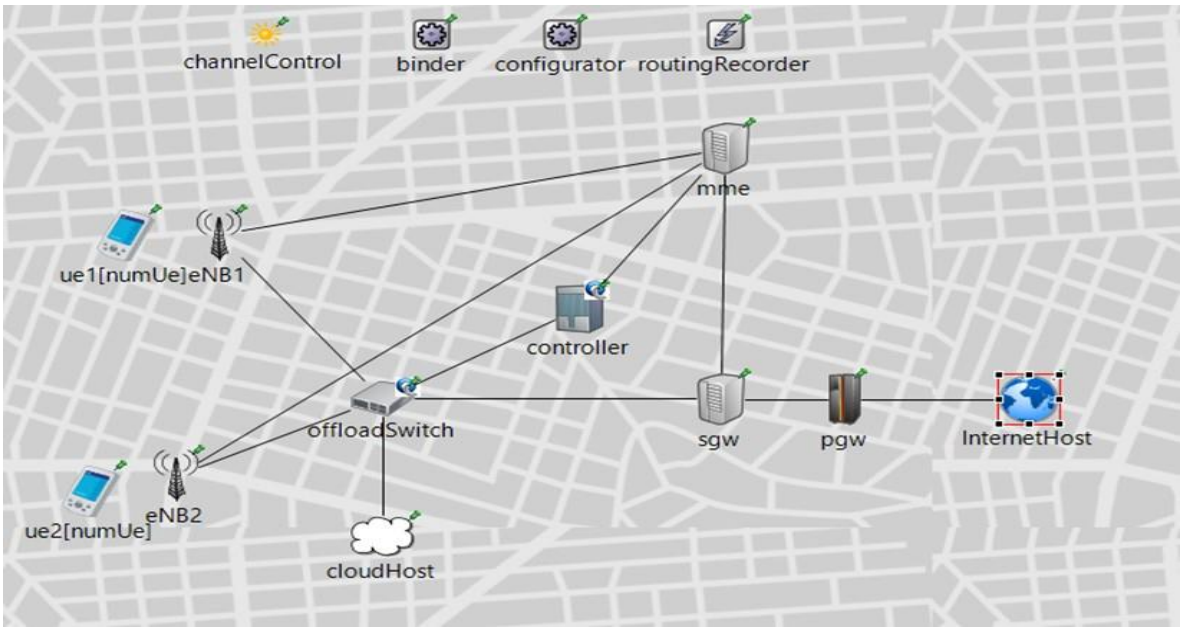


Figure 7.20: Network Topology Simulated in OMNeT++

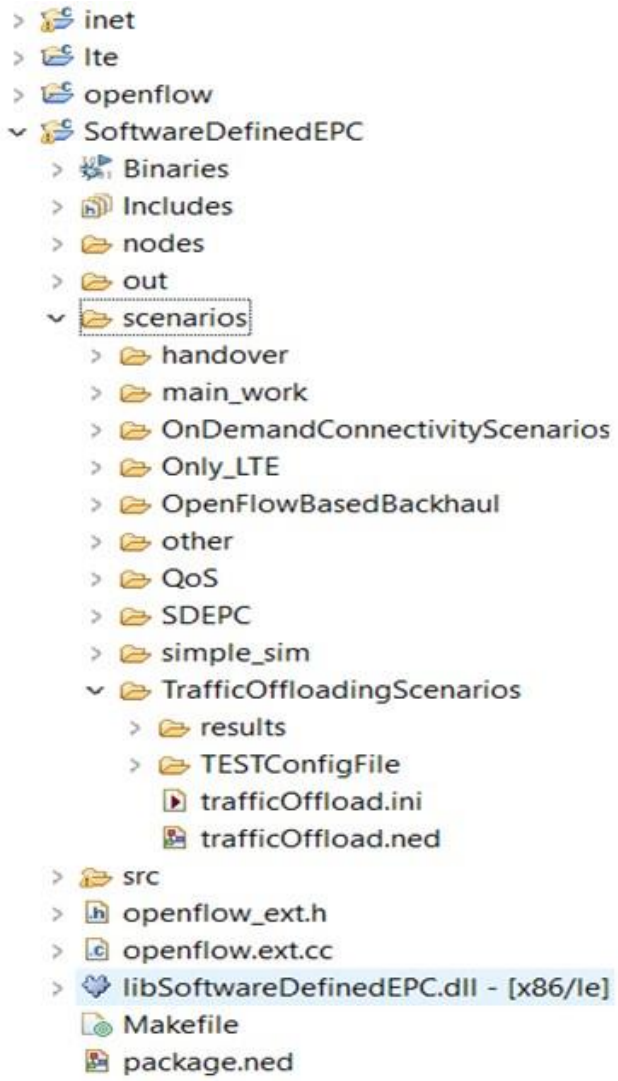


Figure 7.21: Simulation platform File hierarchy

7.5.1 Simulated Module of the MME Entity

The design and the structure of the MME entity are kept the same, while the internal structure of the mmeApp module is extended to include a sdnAgent module. Figure 7.22 shows the design of the MME application Module used in our simulation. The sdnAgent module provides the MME entity with the functionality and tools required to implement the selective traffic offloading. The sdnAgent module is responsible for gathering a copy of bearer information of the UEs eligible for the traffic offloading and communicates with the SDN controller to implement the offloading solution in the backhaul access sub-network.

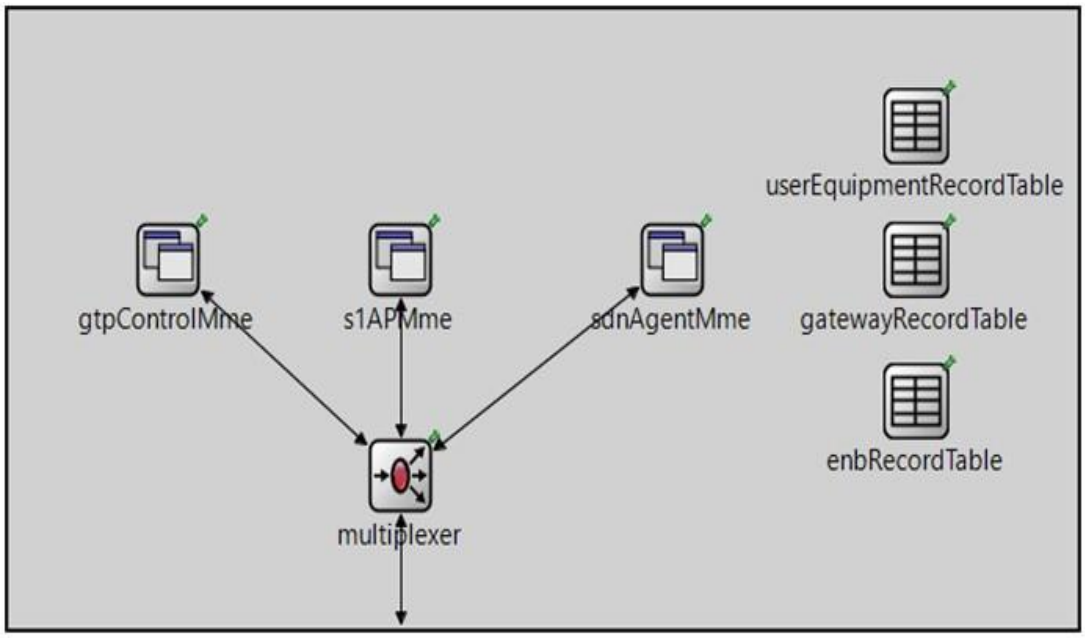


Figure 7.22: MME Application Internal Structure

The sdnAgent module is programmed to work like a plugin, which can be turned On or Off by utilizing a set of new parameters added to MME entity. These parameters include sdnEnabled, and sdnAgentPeerAddr. The sdnEnabled parameter is used to specify the operation mode of the MME module (Standard, Traffic Offloading Enabled). If this parameter is set to false, then the MME works based on the 3GPP specification and the sdnAgent module is not functional during the simulation. Otherwise, if the sdnEnabled parameter is set to true, then the sdnAgentPeerAddr parameter must be set as well. This parameter specifies the IP address or the name of the SDN controller. Also, the sdnAgent module is programmed with multiple blocks of code and uses another set of parameters to specify operation mode.

The parameters used by the sdnAgent module to specify the block of code it should run includes sdnOperatingMode, offloadOperatingMode, and trafficOffloadVlans. The sdnOperatingMode is used to specify the operating mode of the sdnAgent Module. The sdnOperatingMode must be set to one of the following (solution 1, solution 2 and solution 3). The offloadOperatingMode parameter is used to specify whether to use a subscription-based or on-demand offloading. In case of the subscription-based offloading, the offloadEligibleUeAddresses parameter is used to specify the IP addresses or the names of the UEs that are eligible for the traffic offloading. If the sdnOperatingMode is set to either

solution 2 or solution 3, then the trafficOffloadVlans parameter is used to specify a list of VLANs that should be used to identify the traffic that needs to be offloaded to the cloud-based infrastructure. The trafficOffloadVlans parameter only accepts two values that represent the start and the end of the VLANs list. A snippet of the MME configuration that utilizes the aforementioned parameters is presented below.

```
======== MME Configuration Setup for Solution 1 =========
**.mme.sdnEnabled = true
**.mme.sdnAgentPeerAddr = "controller"
**.mme.sdnOperatingMode = "solution1"
**.mme.offloadingOperatingMode = "subscription-based"
**.mme.offloadEligibleUeAddresses = "ue1[0] ue1[1] ue1[2] ue1[3] ue1[4]"
======== MME Configuration Setup for Solution 2 =========
**.mme.sdnEnabled = true
**.mme.sdnAgentPeerAddr = "controller"
**.mme.sdnOperatingMode = "solution2"
**.mme.offloadingOperatingMode = "subscription-based"
**.mme.offloadEligibleUeAddresses = "ue1[0] ue1[1] ue1[2] ue1[3] ue1[4]"
**.mme.trafficOffloadVlans = "100 300"
======== MME Configuration Setup for Solution 3 =========
**.mme.sdnEnabled = true
**.mme.sdnAgentPeerAddr = "controller"
**.mme.sdnOperatingMode = "solution3"
**.mme.offloadingOperatingMode = "subscription-based"
**.mme.offloadEligibleUeAddresses = "ue1[0] ue1[1] ue1[2] ue1[3] ue1[4]"
**.mme.trafficOffloadVlans = "100 300"
```

The configuration snippet shown above describes how to configure the MME entity to perform traffic offloading based on the different solutions described in Sections 7.2, 7.3 and 7.4. This configuration tells the MME to work in a subscription-based offloading mode and sends the traffic of UE1(0) to UE1(4) to the cloud. In solution 2 and 3 the VLAN range of 100 to 300 is used to tag the UEs traffic to help the Traffic-Offloading-Switch to send the traffic to the correct offloading infrastructure. It is also required to specify the controller behaviour. This parameter identifies the control plane application that the controller needs to use to implement the network forwarding behaviour.

```
**.controller.behavior="OffloadingApp"
**.controller.sdnAgentPeerAddr="mme"
```

OffloadingApp is a C++ class that built to support the traffic offloading operations. It works like a normal layer 2 switch application but at the same time it has the capability to intercept the MME offloading notification messages and react to it by configuring the data plane FDs (Traffic-Offloading-Switch). The controller also needs to know the address or the name of the MME entity. The second line of the above configuration does just that.

7.5.2 Simulation Parameters

This section describes and defines the parameters used in our simulation. These parameters are divided into general and network specific parameters. The general parameters describe the simulation time, repetition number, entity positions and simulation environment, while the network specific parameters describe the EUTRAN settings and EPC setting. Finally, a general overview of the used traffic generator and receivers is presented in the next sub-sections.

7.5.2.1 General Parameters

The simulation of each solution is repeated 5 times with different seed numbers to reduce simulation artifacts and the average result of the 5 runs is used as optimal result. Each run lasted for 100s with the UE statically positioned at a fixed point of the simulation playground, and all the UEs are guaranteed to have QCI 15 to ensure that the measured results are not affected by the distance between the UE and its serving eNodeB. This section presents the parameters of the simulation environment.

7.5.2.2 Network specific parameters

This section presents the common settings of the access and the core networks used in the simulation of all the solutions.

A. EUTRAN-Settings

As previously mentioned, the EUTRAN network consists of two eNodeBs to serve multiple UEs. In the simulation scenarios, the eNodeB employs a MaxC/I scheduler and operates on a 20MHz frequency band (100 Physical Resources Blocks available). The Main EUTRAN simulation parameters are summarized in Table 7.1.

Table 7.1: EUTRAN Configuration Parameters [88]

Parameter	Value
Carrier Frequency	2 GHz
Bandwidth	20 MHz (100 RBs)
Mobility Model	Linear/Stationary
UE Speed	10/0 m/s
Path Loss	Urban Macro
eNodeB Tx Power	40 dBm
UE Tx Power	26 dBm
eNodeB Antenna Gain	18 dB
Noise Figure	5 dB
Cable Loss	2 dB

B. EPC-Settings

In our simulation all the EPC entities are statically positioned in the playground. EPC entities are connected to the access network through Ethernet links, at the same time they are connected to each other and the Internet through Point-to-Point links. IP addresses are assigned to each entity at the beginning of the simulation by the IPv4NetworkConfigurator module. The latter reads an XML file to properly assign an IP address to each Interface of each node.

The PGW is connected to SGW from one side and the Internet from the other side, the delay of the link between the InternetHost and the PGW is set to 8ms, while the delay between the PGW and SGW is set to 4ms. The MME is connected to the SGW and the serving eNodeBs through a Point-to-Point link and the Ethernet backhaul respectively, while the SGW is connected to the MME and PGW through point-to-point link and the serving eNodeBs through the Ethernet backhaul. The backhaul delay is divided into two parts. The first is the delay from the eNodeB to the OpenFlow switch which is set to 1ms,

and the second is from the OpenFlow switch to the EPC, more specifically the SGW and MME respectively. The delay of this link set to 4ms.

C. Traffic Generator Applications

INET library represents a place rich with multiple traffic generators and sinks that work as UDP or TCP applications. It also has applications that work on top of the network layer like ICMP application and other application that works with SCTP. In our simulation's different types of traffic generators and receivers are used. This includes the UDPBasicApp, UDPVideoStreamApp, and PingApp.

The UDPBasicApp is used as traffic generator and the sink is used in the destination node to gather statistics and drop the received packet. The same process is applied to the UDPVideoStreamApp. The UEs use UDPVideoStreamClient application to request a UDP video stream from the InternetHost or the CloudHost. These two entities are running UDPVideoStreamServer application that respond to the video stream request of the UEs. The PingApp sends ICMP request messages and measure the statistics of the reply messages. The design and the work structure of UDPVideoStream and PingApp are explained in detail in Chapter 5 Section 5.5.1, while the UDPBaiscApp is explained in Chapter 6 Section 6.4.1. The send interval, packet size and the start and the end of the application is mentioned in the experiments.

To simulate one-direction traffic, a traffic generator is run on one side and a sink in the other side. For example, to simulate uplink traffic, a traffic generator is used on the UE and a sink application is used on the InternetHost and the same applies with the downlink traffic where the generator is used on the InternetHost and the sink on the UEs.

7.5.3 Performance Metrics and Evaluation Results

In this section, OMNeT++ is used to simulate the EPS network with three different offloading solutions to compare the system performance with these solutions and weight the side effect of the standard EPS implementation. End-to-end delay, RTT, processing Time, and virtual ports statistics are used as metrics to evaluate the system performance. Three experiments were conducted to measure and evaluate the system performance. The end-to-end delay and RTT are measured in the first experiment, while the second

experiment deals with the TOS performance in the different offloading solutions. The last experiment measures the performance of the three offloading solutions during handover.

7.5.3.1 Traffic Offloading Experiment

The aims of this experiment are to measure and compare the performance of the EPS network with and without the traffic offloading solution. The end-to-end delay and the RTT are used as metrics for the performance evaluation. In this experiment, 4 simulation runs are used to measure the required statistics. The first two simulations use UDPBasicApp to measure the end-to-end delay, and the other two simulations use pingApp to measure the RTT. In all simulation runs UE1 is connected to eNodeB1.

A. UDPBasicApp Case

In this case two simulation runs are used. The first one without traffic offloading solution and the second run with traffic offloading solution. The same network topology and configuration is used in both simulation-runs with only one difference. The MME sdnEnbled parameter is set to false in the first simulation run and true in the second.

```
#==========First run==============
**.mme.sdnEnabled=false
#==========Second run==============
**.mme.sdnEnabled=true
**.controller.behavior="OffloadingApp"
**.mme.sdnAgentPeerAddr=controller

**.mme.sdnOperationMode=solution1
**.mme.offloadOpeatingMode=subscription-based
**.mme.offloadEligibleUeAddresses=ue1[0]
```

In the first run, the MME does not run the sdnAgent module and operates normally as specified by the 3GPP specifications. In the second run, the MME knows it should use the sdnAgent module and therefore, other parameters are required to correctly configure the MME to know the list of eligible UEs for the offloading operation and at the same time exchange the required information to implement the offloading solution with the SDN controller of the backhaul access sub-network.

In these simulations, ue1(0) uses UDPBasicApp to send a 200B packet every 1s. The application is configured to wait a random time between 0.1s and 0.2s before it starts sending the UDP packets and stops after 90s. Both the InternetHost and the cloudHost modules are using a sink application. The sink application records few statistics about the received packet before dropping it. In this simulation ue1(0) sends 90 UDP packets. In the first run, UDP packet pass through the EPC network to the InternetHost, which measures and records the end-to-end delay. In the second run, the MME is configured to notify the SDN controller to send ue1(0) UDP packets to the cloudHost. The latter does exactly the same as the InternetHost, it measures and records the end-to-end delay before dropping the received packet. Figure 7.23 shows the measured system performance in terms of end-to-end delay with and without the traffic offloading solution. It can be observed that using a distributed cloud-based infrastructure to perform traffic offloading enhances the system performance and provides better performance especially for applications that are very delay sensitive.

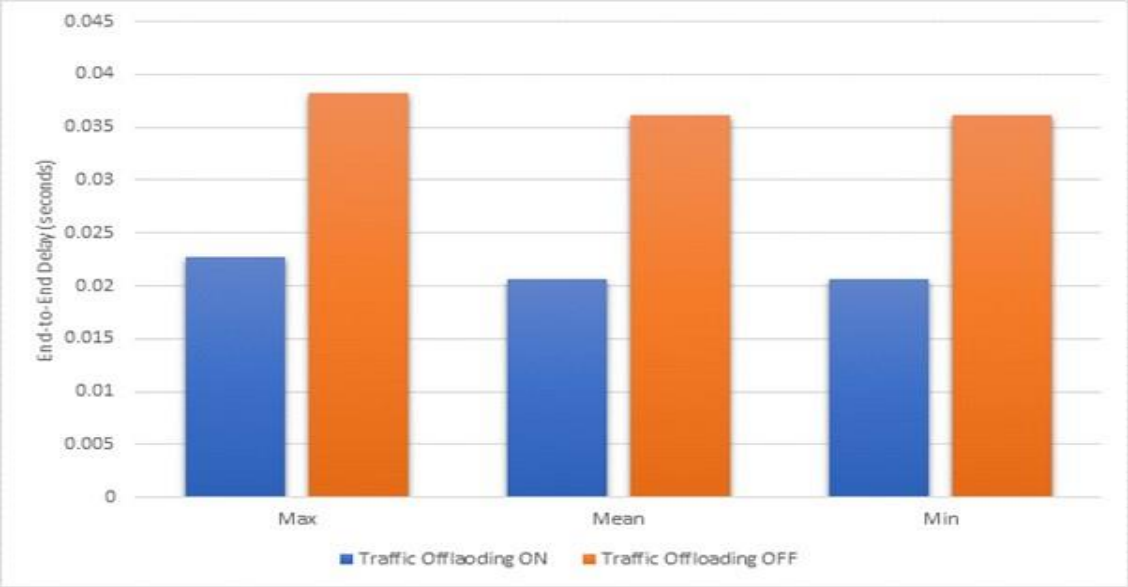


Figure 7.23: Expirement1-UDPBasicApp end-to-end delay of the EPS with and without the traffic offloading Solution

Figure 7.23 shows the minimum, maximum and the average end-to-end delay with and without the traffic offloading solution. The blue bars represent the measured statistics of the system when solution 1 of the cloud offloading is used, while the orange bars represents the measured statistics of the system without traffic offloading solution. It can be observed

that the traffic offloading solution helps to enhance the system performance by providing a better end-to-end delay compared to the standard EPS implementation. This is due to the fact that the packet has travelled a shorter distance which reduced the link and processing delays. With the offloading solution, a virtualized server is deployed in a cloud infrastructure deployed near to the access network and it is responsible for handling the UE1(0) request, which contributes to reduce the distance between the services and the UE.

B. PingApp Case

In this simulation 10 UEs are connected to eNodeB1, each UE is equipped with PingApp application module to send an ICMP request message to the IP address of the server. The server responds by sending an ICMP reply message back to the UE. Upon receiving the message, the PingApp calculates the RTT that was used to provide an estimate of the service latency with and without the traffic offloading solution. In this simulation the PingApp starts at 0.2s and sends ICMP request message every 1s to the destination node. The applications stop after 90s and the simulation stops at 100s. Two packet sizes are used in these simulations, specifically 32B, and 200B. Figure 7.24 illustrates the average RTT of the 10 UEs with and without traffic offloading solution when the packet size is 32B or 200B.

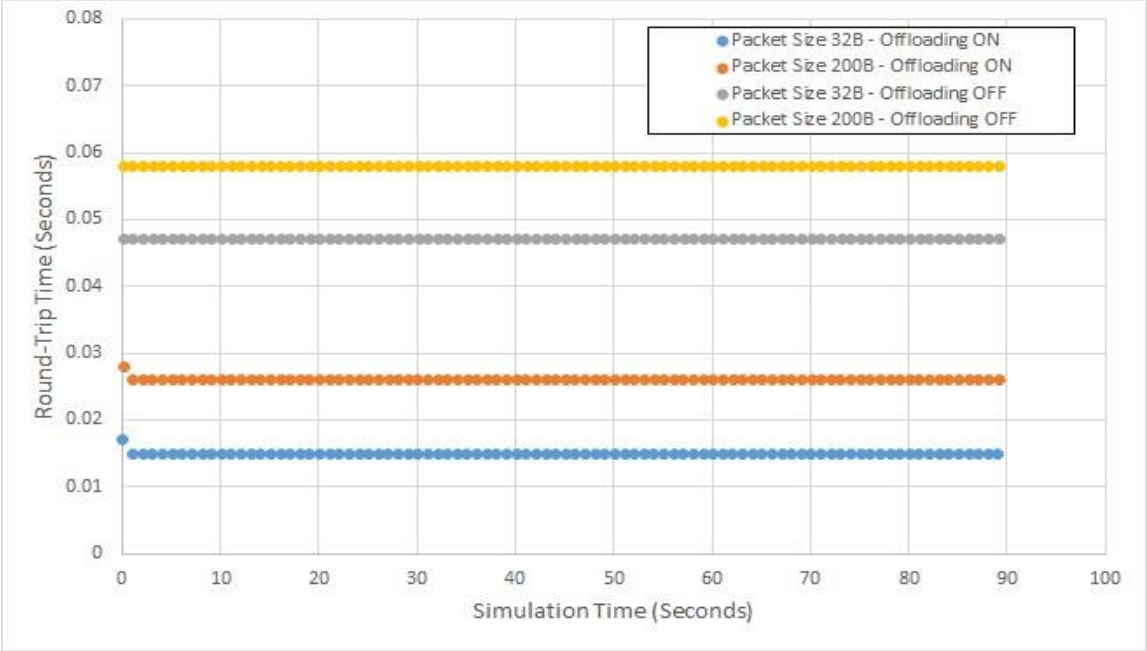


Figure 7.24: Expirement1-PingApp RTT with and without Cloud Offloading Solution

The results plotted in Figure 7.24 shows:

- When cloud offloading is on, the RTT of the first packet is higher than the following packets because the first packet is sent to the controller for more processing.
- The larger the packet the higher the RTT, but this has nothing to do with the offloading solution and is related to mobile network access network work flow.
- Using the cloud-based offloading helps to improve the services RTT to almost the half because the services are always near to the UEs.
- No packet loss or out-of-order packets are encountered in this simulation run because only a small number of UEs are used.

7.5.3.2 Traffic Offloading Switch Performance Experiment

This experiment aims to compare the Traffic-Offloading-Switch (TOS) processing time of each solution. OMNeT++ simulator is used to simulate the network and captures the number of packets sent to the inspection virtual port and the number of packets extracted/returned from/to the S1-U GTP tunnel. To achieve this goal, two different scenarios are used. The first scenario measures the number of packets handled by the TOS virtual ports. In this scenario 10 UEs are connected to eNodeB1. Two of them (UE1(0), UE1(1)) are configured to be registered for cloud traffic offloading. UEs are using PingApp to send a 32B packet every 1s and expect a response from the destination node. The TOS virtual port modules can capture and store statistics about the received messages, which can be used for post processing. Figure 7.25 shows the number of packets handled by each virtual port of the TOS device during the simulation of three different offloading solutions that previously explained in Sections 7.2, 7.3 and 7.4.

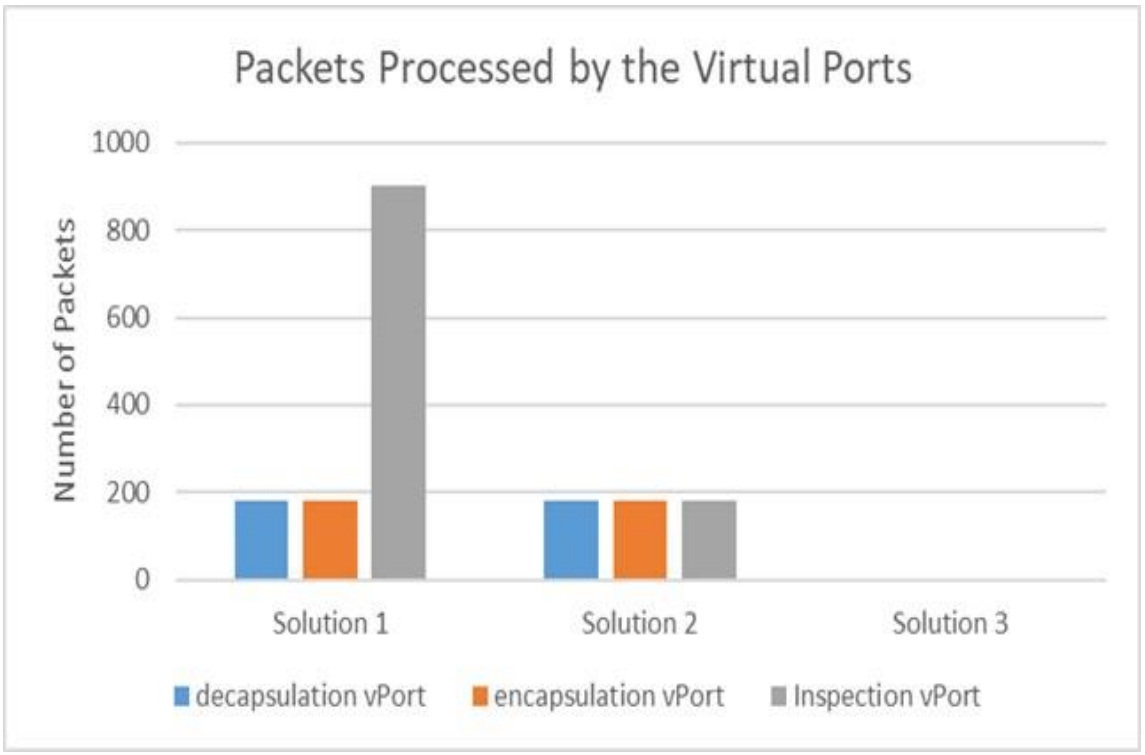


Figure 7.25: TOS Virtual Port Statistics

The results illustrated in Figure 7.25 show that:

- In Solution 1, the TOS device passes all the traffic it received through the inspection virtual port and only decapsulates/encapsulates the traffic needs to be sent to the cloud infrastructure. This behaviour introduces extra processing delay for the traffic that need to be sent the EPC.
- In Solution 2 only the traffic needs to be sent to the cloud is inspected and decapsulated before sending it to the cloud and the return traffic is encapsulated and returned back to the correct GTP tunnel. That way only 180 packets are inspected, decapsulated and encapsulated which is equal to the traffic sent and received by UE1(0) and UE1(1).

In solution 3 the TOS does not use the virtual ports to handle the traffic offloading operations.

The second scenario measures the TOS processing time to perform the following: i) differentiate between UE1(0),UE1(1) traffic and the other UEs traffic; ii) extract the uplink traffic sent by UE1(0) and UE1(1) from the GTP tunnel and send the UEs original packets to the cloud infrastructure, at the same time it returns the cloud response back to the GTP

tunnel; iii) forwards the traffic of the other UEs to the EPC without any modification. In this scenario 10 UEs are connected to eNodeB 1 and only UE1(0) and UE1(1) are eligible for the cloud offloading. The UEs are equipped with UDPVideoStreamCli application. Both the InternetHost and the cloudHost are using UDPVideoStreamSrv application. The UDPVideoStreamCli application send a video stream request packet to the server application, and upon receiving the request message the UDPVideoStreamSrv sends a 1000 Byte packet every 400ms until the simulation ends. Figure 7.26 shows the processing time of the TOS to perform the aforementioned operations with three different solutions.

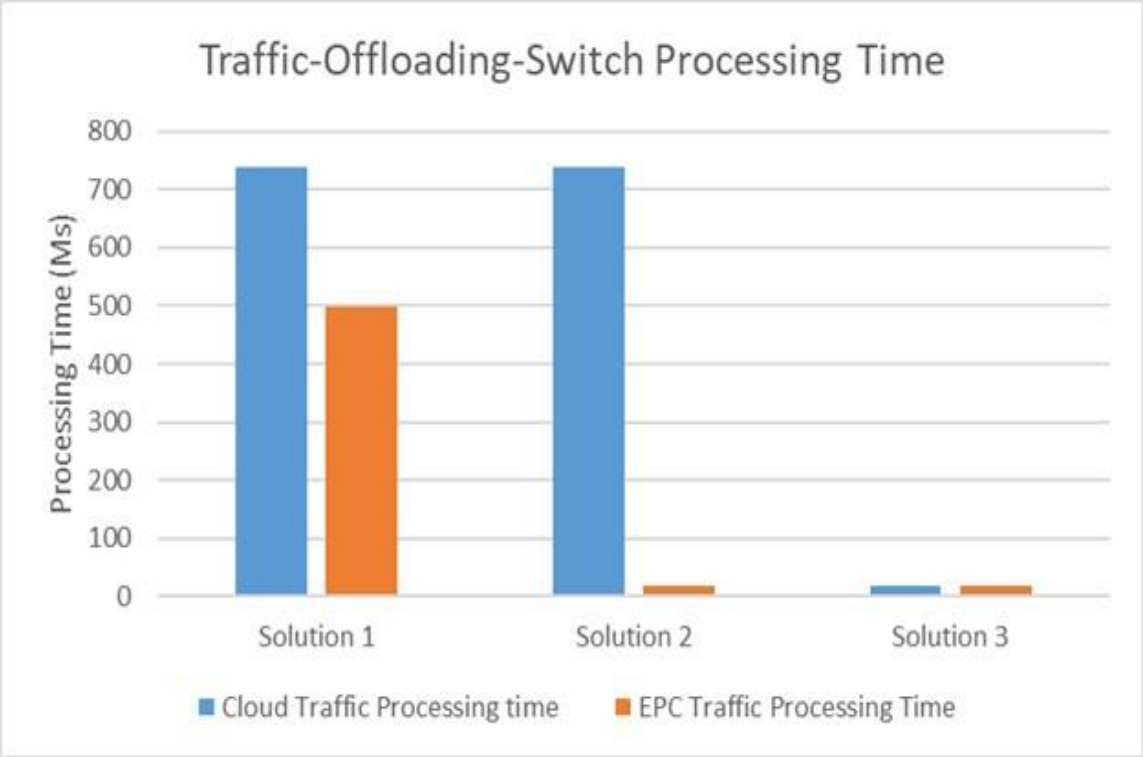


Figure 7.26: TOS Processing Time

It can be observed from the results shown in Figure 7.26 that:

- Solution 3 provides the best performance in terms of the processing time required by the TOS device to: i) identify the traffic needs to be sent to the cloud infrastructure; ii) altering the direction of the identified traffic from the EPC to the cloud.
- In both solution 1 and 2 the time required to handle the traffic that needs to be sent to cloud is more than time required for the traffic that needs to be sent to the EPC.

This is because the TOS devices need to extract the traffic from the GTP tunnel (uplink traffic) and return them back to the tunnel (downlink traffic).

- Solution 2 enhanced the system performance in terms of the processing time required to forward the traffic that needs to be sent to the EPC because in this solution the TOS only need to inspect and decapsulate tagged traffic, while all other traffic is forward normally without any extra processing.

7.5.3.3 UE Handover Experiment

The experiment focusses on the user mobility and the performance of the three offloading solutions during the UE handover. Network topology used in this experiment is shown in Figure 7.20. This network is used to simulate UE handover when both eNodeBs are connected to the same backhaul TOS. Then the backhaul network is modified to include two TOS devices under the control of two separate SDN Controllers.

In this topology eNodeB1 it connected to TOS1 and eNodeB2 is connected to TOS2. This new topology is used to simulate the UE handover between two eNodeBs connected to two TOS that are under the control of different SDN controllers. In this network the link delay between the eNodeBs and the MME assumed to be 5.5 ms, link delay of the S11 interface between the MME and SGW is assumed to be 0.5 ms, link delay between the SDN controller and the MME is assumed to be 4.5ms and the delay between the controller and the TOS switch is assumed to be 0.5 ms. Finally link delay between the eNodeBs and the TOS is assumed to be 1 ms. In the beginning of the simulation, UE1(0) and UE2(1) start connected to eNodeB1 and move in a straight line toward eNodeB2 till they reach the simulation playground boundary. At this point the UEs change their direction toward eNodeB1. The UEs keep following this pattern throughout the simulation time (100s).

Figure 7.27 shows the change of the serving eNodeB during the simulation time. The UEs are periodically sending a measurement report to their serving eNodeBs and based on these reports the serving eNodeB initiates the handover procedure. The Handover procedure is explained in Chapter 4 Section 4.4.4.1. Here the involvement of the core network in downlink traffic redirection to the new eNodeB where the UE is currently connected is the main concerned. In our simulation UE1(0) is registered for the cloud traffic offloading, while UE1(1) is not. The time required to redirect the downlink traffic to the current

location of the UE in question is used as a metric to evaluate the system performance of the proposed solutions. In order to achieve this goal, OMNeT++ signal and statistic mechanisms is used to fire a trigger for the start and the end of the handover completion process. The target eNodeB triggers the ”start” signal after it sends a Path-Switch-Request message to the MME and the ”end” signal after receiving the Path-Switch-Request-ACK message.

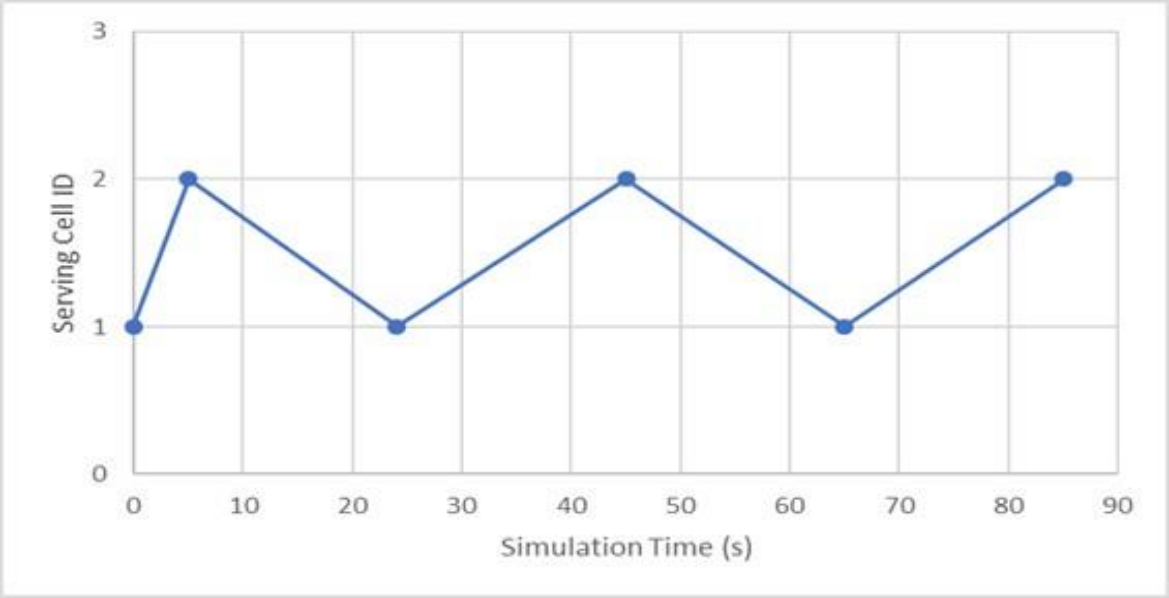


Figure 7.27: UE1(0) Serving Cell Id during the simulation

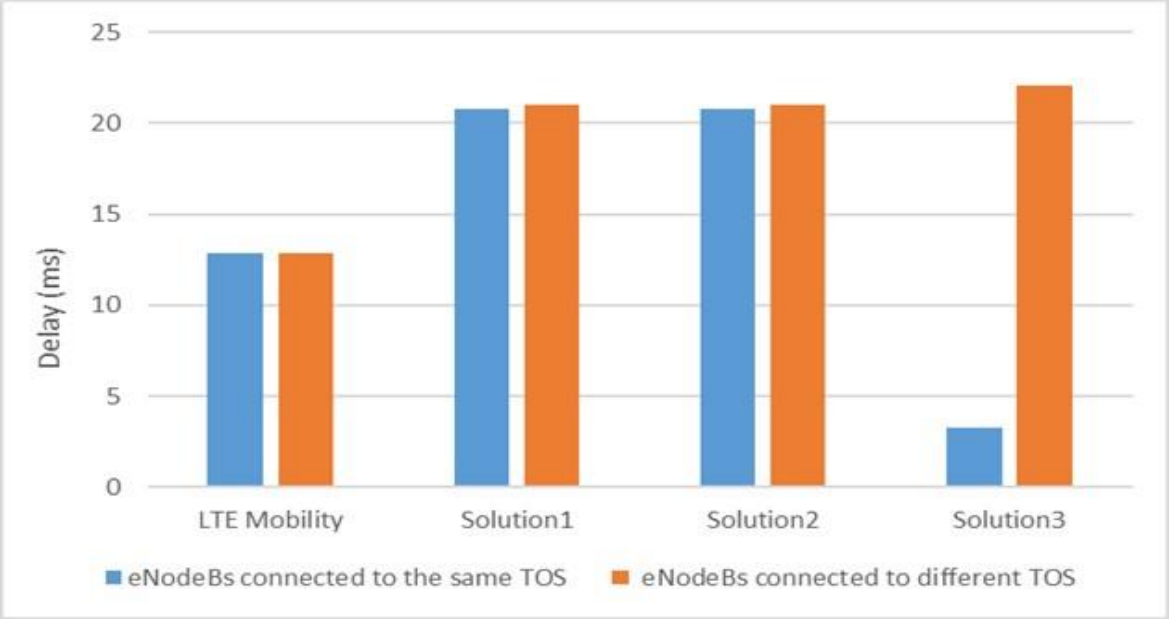


Figure 7.28: Time required to Handle Downlink Traffic Redirection

Results illustrated in Figure 7.28 shows:

- The time required to handle the downlink redirection in the cloud traffic offloading solutions (Solution 1 and 2 and one case of solution 3) is more than the time required to handle the same process in the standard LTE implementation. Normally the time required to redirect the downlink traffic is calculated by using the below equation.

$$TRT = (EMLDx2) + (MSLDx2) + MPT + SPT + EPT \quad (7.1)$$

where TRT represent the Traffic Redirection Time, EMLD and MSLD represent the link delay of the S1-MME and the S11 interfaces respectively. The MPT, SPT and the EPT represent the processing time of the MME, SGW, and the eNodeB respectively. The EMLD and MSLD are multiplied by 2 because two messages are sent on each interface. For example, the MME sends a Modify-Bearer-Request message to the SGW, and the latter send a Modify-Bearer-Response message back to the MME over the same interface. With the cloud traffic offloading the process is slightly different because the UE in question may have downlink traffic coming from the cloud and the direction of this traffic needs to be altered as well. Both Solutions 1 and 2 require the same time to handle the downlink traffic redirection. In these solutions, the time required to handle the redirection is less if the UE is moving between two eNodeBs, which are connected to the same TOS device. Otherwise, an extra 1.5ms is required to handle the traffic redirection. In the cloud offloading solutions, the MME needs to notify the SDN controller about the UE handover and the controller needs to update the configuration of the TOS device to send the cloud traffic to the new eNodeB (in case both the source and the target eNodeBs are connected to the same TOS). If more than one TOS are involved, then more time is required to finish the redirection process.
- Solution 3 can provide a very efficient way to handle the UE handover and redirect the cloud traffic to the current location of the UE if both the source and the target eNodeBs involved in the UE handover process are connected to the same TOS or between two TOS that under the control of the same SDN controller. Otherwise, solution 3 behaves like the first two solutions and requires more time to handle the traffic redirection. This solution is different and requires less processing time

because the handover process is handled locally by the backhaul access network SDN controller, unless the target eNodeB is connected to a TOS that is under the control of a different SDN controller. In this case, the system falls back to the normal procedure and sends the Path-Switch-Request message to the core network. Processing the Path-Switch-Request locally by the SDN controller is used to realize that handling this request is out of it control area. It then decides to send this request to the MME to handle it, which introduces more time delay to complete the downlink redirection process. This explains the result of solution 3.

CHAPTE 8

8 Conclusion and Future Works

8.1 Conclusion

This thesis presented multiple ways to build SDN-based mobile core network. Several experiments are used to test and evaluate the system performance in terms of end-to-end delay, Packet loss, queueing time, signalling loads, and processing time. In these experiments different types of applications are used in multiple simulation scenarios. The system is simulated in OMNeT++ and some of the results are displayed as a time vector while others are displayed in scatter or bar charts.

- Incorporating SDN in the mobile core network boost the network performance in term of signaling loads. It also helps providing better load distribution as showed in the SBMCNA.
- SDN enables the removal of GTP which contribute to:
    - Provides better resource utilization, routing and reduces the single point of failure.
    - Enhances the system performance in terms of end-to-end delay, Mean Opinion Score, queuing times and packet loss
- Hybrid system that supports heterogeneous network of multiple technologies used to overcome the backward compatibility issue introduced from removing GTP.

- SDN can be used as add-on solution to work with the 4G network not to replace it as showed in SDSTO. Which can be used to move the content near to the user either through caching or cloud-based offloading
- Openflow in its current version is not fit to be used as a southbound protocol for the mobile network. Extension is required whether it used in GTP enabled environment or not.
- Extending the Openflow plugin of the SDN controller and the forwarding device to support new messages that are specifically related to the mobile network tunnels management enhance the system performance by reducing the signalling loads.
- Adding SDN agent to the network forwarding device helps reducing the signaling loads and the processing time of the first packet for every new flow. More advanced and sophisticated agent can be used to address more tasks.
- OMNeT++ is one of the best available network simulations that can be used to simulate software defined mobile network. However, beside it requires a considerable learning curve; it is also not suitable for testing system processing time performance or use it to benchmark the SDN controller performance.

8.2 Future Works

The SDN-EPC integration is discussed from two different approaches. First, SBMCNA which considers keeping GTP as the main data plane forwarding protocol. Second, SDEPC which shows the advantage and disadvantage of removing GTP. This thesis opens doors of possibility for the future works such as the following:

- The implementation and testing of the proposed architectures in a real testbed environment. This includes:
    - The implementation of the proposed GTP extension methods in a real hardware or software OpenFlow switches like Open vSwitch [89].
    - Use a testbed to investigate and interrogate the ability of real SDN controller such as Opendaylight [90] and Floodlight [36] to handle the high

volume of mobile network signalling, then use the findings to extend and enhance our OMNeT++ model.

- Further investigation is required to study the capability of FDs to scale in handling the increasing number of users, their mobility, and providing fine grain access control. The side effect of sending the first packet of each new flow to the controller requires further investigation in terms of latency and link loads in a very large network.
- The proposed architectures use single SDN controller, more investigation is required to check if one controller is sufficient, if not, a simulation of large SDN based mobile network that includes multiple controllers in a cooperative environment is important as part of the future work.
- Building new routing algorithm that works with multiple controllers in a mobile network environment to handle traffic redirection when the UE changes its location to an eNodeB under the control of different SDN controller.
- Extend the system to implement in-network caching.

References

[1] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Taking Control of Cellular Core Networks," *CoRR*, vol. abs/1305.3568, 2013.

[2] M. Mendonca, K. Obraczka, and T. Turletti, "The Case for Software-defined Networking in Heterogeneous Networked Environments," in *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, 2012, pp. 59–60.

[3] H. Wang, S. Chen, H. Xu, M. Ai, and Y. Shi, "SoftNet: A software defined decentralized mobile network architecture toward 5G," *IEEE Netw.*, vol. 29, no. 2, pp. 16–22, 2015.

[4] S. Ben *et al.*, "New Control Plane in 3GPP LTE / EPC Architecture for On-Demand Connectivity Service," in *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*, 2013, pp. 205–209.

[5] L. J. Chaves, V. M. Eichemberger, I. C. Garcia, and E. R. M. Madeira, "Integrating OpenFlow to LTE: Some issues toward software-defined mobile networks," in *2015 7th International Conference on New Technologies, Mobility and Security - Proceedings of NTMS 2015 Conference and Workshops*, 2015, pp. 1–5.

[6] J. Costa-Requena, "SDN integration in LTE mobile backhaul networks," in *The International Conference on Information Networking 2014 (ICOIN2014)*, 2014, pp. 264–269.

[7] J. Costa-Requena, V. F. Guasch, and J. L. Santos, "Software defined networks based 5G backhaul architecture," in *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication - IMCOM '15*, 2015, pp. 1–5.

[8] J. Costa-Requena, R. Kantola, A. Y. Ding, J. Manner, Y. Liu, and S. Tarkoma, "Software Defined 5G Mobile Backhaul," in *Proceedings of the 1st International Conference on 5G for Ubiquitous Connectivity*, 2014.

[9] S. Shanmugalingam and P. Bertin, "Programmable Mobile Core Network," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, 2014, vol. Workshops, pp. 1–7.

[10] J. Costa-Requena, J. L. Santos, and V. F. Guasch, "Mobile backhaul transport streamlined through SDN," in *2015 17th International Conference on Transparent Optical Networks (ICTON)*, 2015, pp. 1–4.

[11] A. Banerjee, X. Chen, J. Erman, V. Gopalakrishnan, S. Lee, and J. Van Der Merwe,

"MOCA: A Lightweight Mobile Cloud Offloading Architecture," in *Proceedings of the eighth ACM international workshop on Mobility in the evolving internet architecture - MobiArch '13*, 2013, p. 11.

[12] J. Cho, B. Nguyen, A. Banerjee, R. Ricci, J. Van der Merwe, and K. Webb, "SMORE: software-defined networking mobile offloading architecture," in *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges - AllThingsCellular '14*, 2014, pp. 21–26.

[13] K. Wang, M. Shen, J. Cho, A. Banerjee, J. Van der Merwe, and K. Webb, "MobiScud: A Fast Moving Personal Cloud in the Mobile Network," in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges - AllThingsCellular '15*, 2015, pp. 19–24.

[14] M. Rodrigues, G. Dan, and M. Gallo, "Enabling transparent caching in LTE mobile backhaul networks with SDN," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, pp. 724–729.

[15] M. R. Sama, S. B. H. Said, K. Guillouard, and L. Suciu, "Enabling Network Programmability in LTE/EPC Architecture Using OpenFlow," in *2014 12th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2014, pp. 389–396.

[16] V. Nguyen and Y. Kim, "Proposal and evaluation of SDN-based mobile packet core networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2015, no. 1, p. 172, Jun. 2015.

[17] M. R. Sama, L. M. Contreras, J. Kaippallimalil, I. Akiyoshi, H. Qian, and H. Ni, "Software-defined control of the virtualized mobile packet core," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 107–115, Feb. 2015.

[18] K. Pentikousis, Y. Wang, and W. Hu, "Mobileflow: Toward software-defined mobile networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 44–53, 2013.

[19] A. Bradai, K. Singh, T. Ahmed, and T. Rasheed, "Cellular software defined networking: a framework," *IEEE Commun. Mag.*, vol. 53, no. 6, pp. 36–43, Jun. 2015.

[20] T. Taleb *et al.*, "EASE: EPC as a service to ease mobile core network deployment over cloud," *IEEE Netw.*, vol. 29, no. 2, pp. 78–88, 2015.

[21] P. Gurusanthosh, A. Rostami, and R. Manivasakan, "SDMA: A semi-distributed mobility anchoring in LTE networks," in *2013 International Conference on Selected Topics in Mobile and Wireless Networking, MoWNeT 2013*, 2013, pp. 133–139.

[22] V. G. Nguyen and Y. H. Kim, "Slicing the next mobile packet core network," in *2014 11th International Symposium on Wireless Communications Systems (ISWCS)*, 2014, pp. 901–904.

[23] S. Khairi, M. Bellafkih, and B. Raouyane, "QoS management SDN-based for LTE/EPC with QoE evaluation: IMS use case," in *2017 Fourth International Conference on Software Defined Systems (SDS)*, 2017, pp. 125–130.

[24] M. Karimzadeh *et al.*, "Double-NAT Based Mobility Management for Future LTE Networks," in *2017 IEEE Wireless Communications and Networking Conference*

(WCNC), 2017, pp. 1–6.

[25] M. Karimzadeh, L. Valtulina, H. van den Berg, A. Pras, M. Liebsch, and T. Taleb, "Software Defined Networking to support IP address mobility in future LTE network," in *2017 Wireless Days*, 2017, pp. 46–53.

[26] M. Karimzadeh Motallebi Azar, L. Valtulina, and G. Karagiannis, "Applying SDN/OpenFlow in Virtualized LTE to support Distributed Mobility Management (DMM)," in *Proceedings of the 4th International Conference on Cloud Computing and Services Science, CLOSER 2014*, 2014, pp. 639–644.

[27] L. Valtulina, M. Karimzadeh, G. Karagiannis, G. Heijenk, and A. Pras, "Performance evaluation of a SDN/OpenFlow-based Distributed Mobility Management (DMM) approach in virtualized LTE systems," in *2014 IEEE Globecom Workshops (GC Wkshps)*, 2014, pp. 18–23.

[28] Y. h. Kim *et al.*, "Distributed PDN gateway support for scalable LTE/EPC networks," in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, 2014, pp. 139–144.

[29] Y. Kim, H. Lim, K. Kim, and Y.-H. Han, "A SDN-based distributed mobility management in LTE/EPC network," *J. Supercomput.*, vol. 73, no. 7, pp. 2919–2933, Jul. 2017.

[30] T. Chen, M. Matinmikko, X. Chen, X. Zhou, and P. Ahokangas, "Software defined mobile networks: concept, survey, and research directions," *IEEE Commun. Mag.*, vol. 53, no. 11, pp. 126–133, Nov. 2015.

[31] V.-G. Nguyen, T.-X. Do, and Y. Kim, "SDN and Virtualization-Based LTE Mobile Network Architectures: A Comprehensive Survey," *Wirel. Pers. Commun.*, vol. 86, no. 3, pp. 1401–1438, Feb. 2016.

[32] L. E. Li, Z. M. Mao, and J. Rexford, "Toward Software-Defined Cellular Networks," in *2012 European Workshop on Software Defined Networking*, 2012, pp. 7–12.

[33] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "Cellsdn: Software-defined cellular core networks," *Open Netw. Summit SDN Event*, 2013.

[34] T. Mahmoodi and S. Seetharaman, "On using a SDN-based control plane in 5G mobile networks," *Wireless World Research Forum (WWRF)*. 2014.

[35] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and Flexible Cellular Core Network Architecture," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, 2013, pp. 163–174.

[36] F. Project, "Open Source Software for Building Software-Defined Networks." [Online]. Available: http://www.projectfloodlight.org/floodlight/. [Accessed: 23-May-2018].

[37] J. Kempf, B. Johansson, S. Pettersson, H. Luning, and T. Nilsson, "Moving the mobile Evolved Packet Core to the cloud," in *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2012, pp. 784–791.

[38] L. Wu, J. Liu, T. Huang, W. Wu, and B. Da, “A universal mechanism to handle ION packets in SDN network,” pp. 20–24, Apr. 2017.

[39] L. J. Chaves, V. M. Eichemberger, I. C. Garcia, and E. R. M. Madeira, “Integrating OpenFlow to LTE: Some issues toward software-defined mobile networks,” in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, 2015, pp. 1–5.

[40] N. Adalian, G. Ajaeiya, Z. Dawy, I. H. Elhajj, A. Kayssi, and A. Chehab, “Load balancing in LTE core networks using SDN,” in *2016 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)*, 2016, pp. 213–217.

[41] J. Costa-Requena *et al.*, “SDN and NFV integration in generalized mobile network architecture,” in *Networks and Communications (EuCNC), 2015 European Conference on*, 2015, pp. 154–158.

[42] S. Chourasia and K. M. Sivalingam, “SDN based Evolved Packet Core architecture for efficient user mobility support,” in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015, pp. 1–5.

[43] E. B. Hamza and S. Kimura, “A Scalable SDN-EPC Architecture Based on OpenFlow-Enabled Switches to Support Inter-domain Handover,” in *2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2016, pp. 272–277.

[44] N. Varis, J. Manner, and J. Heinonen, “A layer-2 approach for mobility and transport in the mobile backhaul,” in *2011 11th International Conference on ITS Telecommunications*, 2011, pp. 268–273.

[45] M. Corici, T. Magedanz, D. Vingarzan, and P. Weik, “Prototyping mobile broadband applications with the open Evolved Packet Core,” in *2010 14th International Conference on Intelligence in Next Generation Networks*, 2010, pp. 1–5.

[46] J. Heinonen, T. Partti, M. Kallio, K. Lappalainen, H. Flinck, and J. Hillo, “Dynamic tunnel switching for SDN-based cellular core networks,” in *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges - AllThingsCellular ’14*, 2014, pp. 27–32.

[47] J. Costa-Requena, M. Kimmerlin, J. Manner, and R. Kantola, “SDN optimized caching in LTE mobile networks,” in *2014 International Conference on Information and Communication Technology Convergence (ICTC)*, 2014, pp. 128–132.

[48] M. Kimmerlin, J. Costa-Requena, J. Manner, D. Saucez, Y. Sanchez, and Y. Sanchez Caching Using, “Caching Using Software-Defined Networking in LTE Networks,” 2015.

[49] E. Ghazisaeedi and R. Tafazolli, “Mobile core traffic balancing by OpenFlow switching system,” in *2013 IEEE Globecom Workshops (GC Wkshps)*, 2013, pp. 824–829.

[50] T. Kitahara, A. Riikonen, and H. Hammainen, “Characterizing traffic generated

with laptop computers and mobile handsets in GPRS/UMTS core networks," in *IEEE Local Computer Network Conference*, 2010, pp. 1046–1053.

[51] N. Project, "NS3 discrete-event network simulator for Internet systems." .

[52] M. Liyanage *et al.*, "Enhancing Security of Software Defined Mobile Networks," *IEEE Access*, vol. 5, pp. 9422–9438, 2017.

[53] M. Liyanage *et al.*, "Security for Future Software Defined Mobile Networks," in *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, 2015, pp. 256–264.

[54] T. Taleb, "Toward carrier cloud: Potential, challenges, and solutions," *IEEE Wirel. Commun.*, vol. 21, no. 3, pp. 80–91, Jun. 2014.

[55] T. Chen, H. Zhang, X. Chen, and O. Tirkkonen, "SoftMobile: control evolution for future heterogeneous mobile networks," *IEEE Wirel. Commun.*, vol. 21, no. 6, pp. 70–78, 2014.

[56] M. Arslan, K. Sundaresan, and S. Rangarajan, "Software-defined networking in cellular radio access networks: potential and challenges," *IEEE Commun. Mag.*, vol. 53, no. 1, pp. 150–156, Jan. 2015.

[57] A. M. Nayak, P. Jha, and A. Karandikar, "A Centralized SDN Architecture for the 5G Cellular Network," 2018.

[58] K. Hudson and K. Cannon, *CCNA Guide to Cisco Networking Fundamentals*. Course Technology, 2000.

[59] R. M. Roberts, *Networking Fundamentals*. GOODHEART-WILLCOX PUBL, 2011.

[60] F. Hu, *Network Innovation through OpenFlow and SDN: Principles and Design*. Taylor & Francis, 2014.

[61] P. Goransson, C. Black, and T. Culver, *Software Defined Networks: A Comprehensive Approach*. Elsevier Science, 2016.

[62] ONF, "OpenFlow Switch Specification Version 1.0.0 (Wire Protocol 0x01)." .

[63] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Futur. Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[64] ONF, "OpenFlow Switch Specification Version 1.2 (Wire Protocol 0x03)." .

[65] ONF, "OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04)." .

[66] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[67] K. S. S, *OpenFlow Cookbook*. Packt Publishing, 2015.

[68] D. Kreutz, F. M. V Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[69] A. Varga and R. Hornig, “AN OVERVIEW OF THE OMNeT++ SIMULATION ENVIRONMENT,” in *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications Networks and Systems*, 2008.

[70] D. Klein and M. Jarschel, “An OpenFlow Extension for the OMNeT++ INET Framework,” in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques (SimuTools ’13). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering)*, 2013, pp. 322–329.

[71] C. R. A. Vidal E. Fernandes and M. Salvador, “The OpenFlow Driver (libfluid).” [Online]. Available: http://opennetworkingfoundation.github.io/libfluid/. [Accessed: 23-May-2018].

[72] A. Ghosh and R. Ratasuk, *Essentials of LTE and LTE-A*. Cambridge University Press, 2011.

[73] S. Ahmadi, *LTE-Advanced: A Practical Systems Approach to Understanding 3GPP LTE Releases 10 and 11 Radio Access Technologies*. Elsevier Science, 2013.

[74] X. Chang, “Network simulations with OPNET,” in *Simulation Conference Proceedings, 1999 Winter*, 1999, vol. 1, pp. 307–314 vol.1.

[75] S. Y. Wang, C. L. Chou, and C. M. Yang, “EstiNet openflow network simulator and emulator,” *IEEE Commun. Mag.*, vol. 51, no. 9, pp. 110–117, 2013.

[76] G. F. Riley and T. R. Henderson, “The ns-3 Network Simulator,” in *Modeling and Tools for Network Simulation*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34.

[77] G. Piro, L. A. Grieco, G. Boggia, F. Capozzi, and P. Camarda, “Simulating LTE Cellular Systems: An Open-Source Framework,” *IEEE Trans. Veh. Technol.*, vol. 60, no. 2, pp. 498–513, 2011.

[78] A. Virdis, G. Stea, and G. Nardini, “Simulating LTE/LTE-Advanced Networks with SimuLTE,” in *Simulation and Modeling Methodologies, Technologies and Applications*, 2015, pp. 83–105.

[79] I. Community, “INET Framework for OMNeT++ Manual,” pp. 1–157, 2016.

[80] G. Nardini, A. Virdis, and G. Stea, “Modeling X2 backhauling for LTE-advanced and assessing its effect on CoMP coordinated scheduling,” in *2016 1st International Workshop on Link- and System Level Simulations (IWSLS)*, 2016, pp. 1–6.

[81] G. Hampel, M. Steiner, and T. Bu, “Applying Software-Defined Networking to the telecom domain,” in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2013, pp. 133–138.

[82] M. A. Salih, J. Cosmas, and Y. Zhang, “OpenFlow 1.3 extension for OMNeT++,” in *Proceedings - 15th IEEE International Conference on Computer and Information Technology, CIT 2015, 14th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2015, 13th IEEE International Conference on Dependable, Autonomic and Se*, 2015.

[83] 3GPP, “Restoration procedures,” no. TS 23.007, pp. 1–157, 2012.

[84] Z. A. Qazi, M. Walls, A. Panda, V. Sekar, S. Ratnasamy, and S. Shenker, “A High Performance Packet Core for Next Generation Cellular Networks,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 348–361.

[85] R. Chayapathi, S. F. Hassan, and P. Shah, *Network Functions Virtualization (NFV) with a Touch of SDN*. Pearson Education, 2016.

[86] J. Y. Yen, “Finding the K Shortest Loopless Paths in a Network,” *Management Science*, vol. 17. INFORMS, pp. 712–716.

[87] K. T. Chen, P. Huang, and C. L. Lei, “Effect of Network Quality on Player Departure Behavior in Online Games,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 5, pp. 593–606, 2009.

[88] G. Nardini, G. Stea, and A. Virdis, “Performance Evaluation of TCP-Based Traffic over Direct Communications in LTE-Advanced,” in *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, 2016, pp. 1–5.

[89] B. Pfaff *et al.*, “The Design and Implementation of Open vSwitch,” in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, 2015, pp. 117–130.

[90] O. controller, “OpenDaylight Foundation.” [Online]. Available: https://www.opendaylight.org/. [Accessed: 23-May-2018].

Appendix

Please see the source code of the presented Software Defined Mobile Networks in the CD included with this thesis. To use the source files, you need to:

- Download OMNeT++ 4.4.1 source code from the following url: https://omnetpp.org/component/jdownloads/download/32-release olderversions/2273-omnet-4-4-1-win32-source-ide-mingw-zip. This URL last checked on 15th Aug 2018. if the URL is not working please visit https://omnetpp.org/ and click on Download on the top of the page, then click on older version and go through the different versions until you find the required version.
- Install OMNeT++ 4.4.1 on your computer please make sure that you download the version compatible with your operating system. The link provided is for windows, but you can find version for Linux as well. To install OMNeT++ on your computer, first unzip the source code you downloaded in step 2 to your C:/ drive, after doing that go to C:/omnetpp-4.4.1 where the source files of the OMNeT++ is located, then double click on the mingwenv file and write the following commands:

  $. setenv

  $ ./configure

  $ make

  Write these commands one by one, which mean that you need to wait until the need of the execution of the command before writing the next command.

- Now open the simulator by double clicking on the mingwenv file and write the command:

  $ omnetpp

  Figure A.1 shows the way to run OMNeT++ simulator from the mingwenv file.

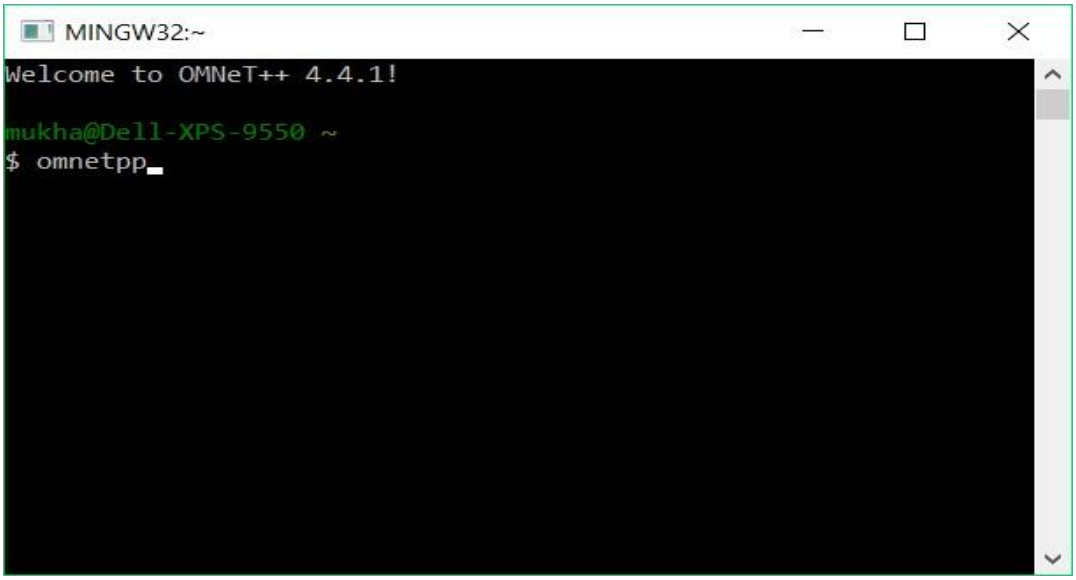


Figure A. 1: Open OMNeT++ 4.4.1 simulator

- The First time you open the simulator, it will ask you to import the latest version of the INET library from the Internet. Please click no because INET 2.3 used in our simulation in not compatible with the newer versions.
- Now you are ready to import the source code of our Software Defined Mobile Networks. First copy the source code from the CD to your workspace folder. This folder you must specify in step number 3.
- Now you can import the source code to the OMNeT++ simulator. To do that click on File→ Import→ general→ existing projects into workspace as shown in Figure A.2 and Figure ??, then click next.

After you click next, you will see Figure A.4, you need to click browse and go to the workspace folder and click on the inet folder, the click finish. Please make sure that the ”Add project to working set is unchecked as shown in Figure A.4.

Now at the left-down corner of the OMNeT++ interface, you will see c/c++ indexer as shown in Figure A.5. Please wait for it until it finished before starting the next step.

Right click on the inet folder shown in the project explorer and click on build project as shown in Figure A.6 and Figure A.7

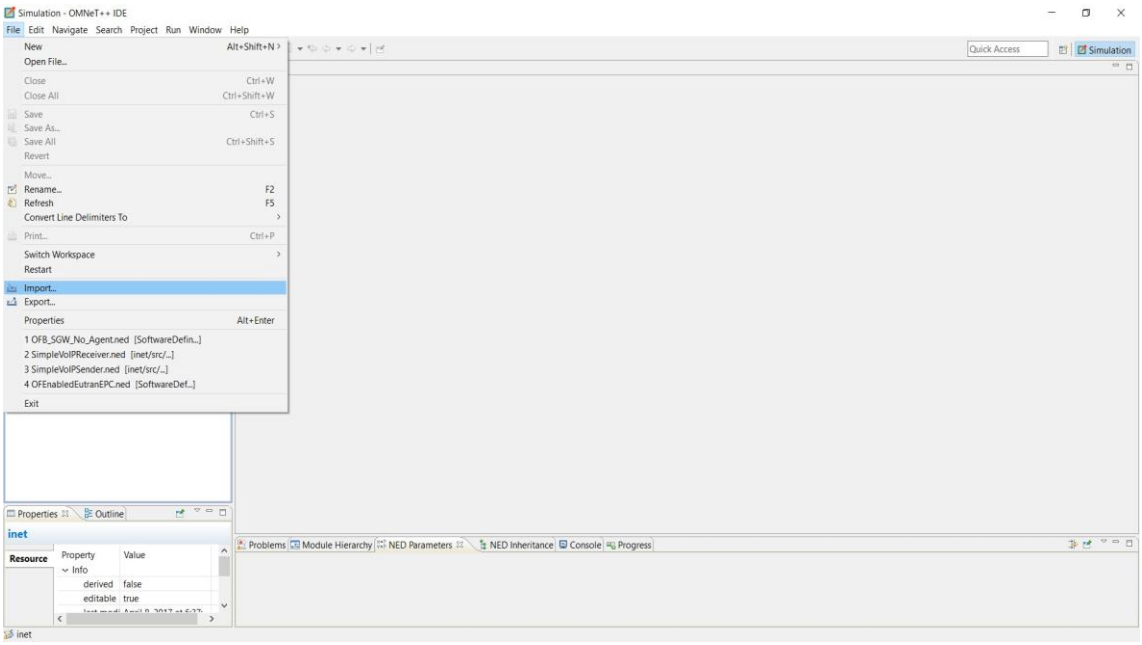


Figure A. 2:Import the source code of the INET library - step 1

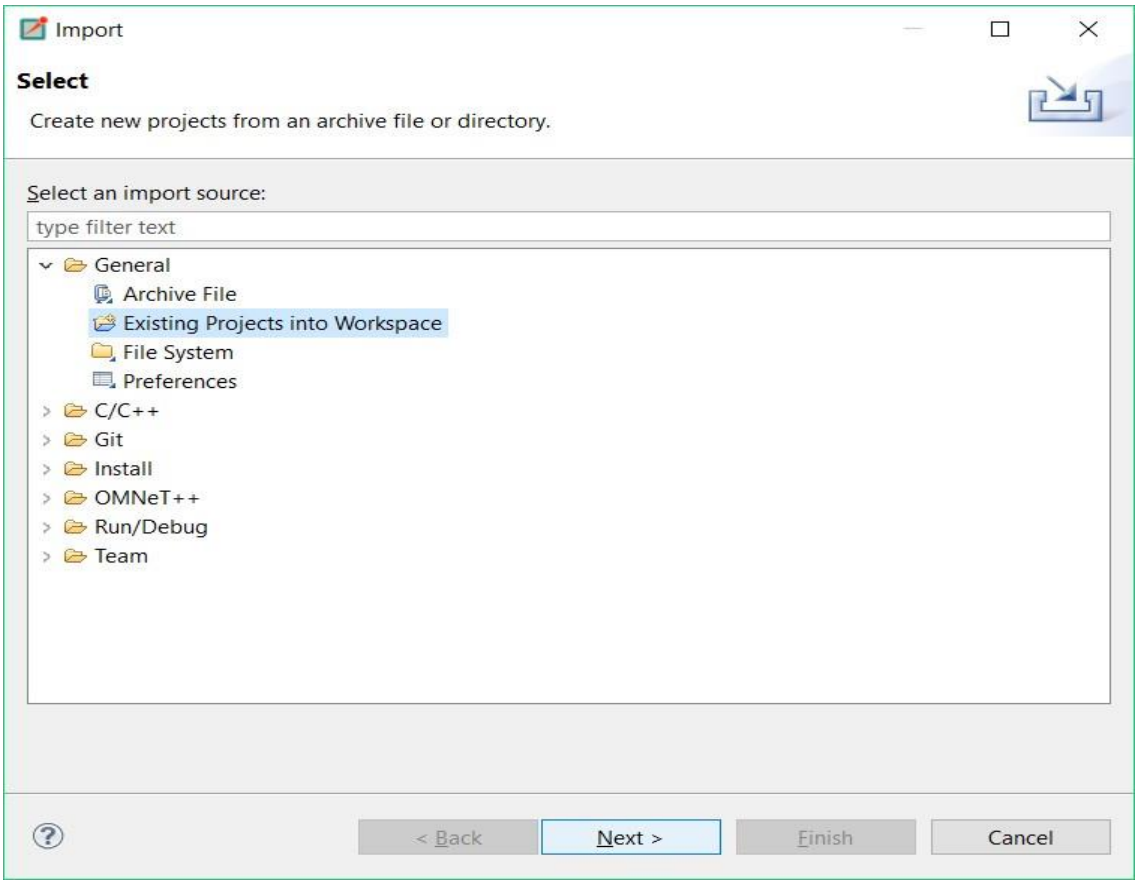


Figure A. 3: Import the source code of the INET library - step 2

- In the project explorer click on the SoftwreDefinedEPC→ scenario, and you will see all the scenarios described in this thesis and more. To run one of them click on the folder and right click on the '.ini' file→ run as → OMNeT++ simulation as shown in Figure A.8.
- In this case, the simulation will run using the default parameters written in the '.ini' file. You can change the simulation parameters by clicking on the '.ini' file and alter the parameters you want as shown in

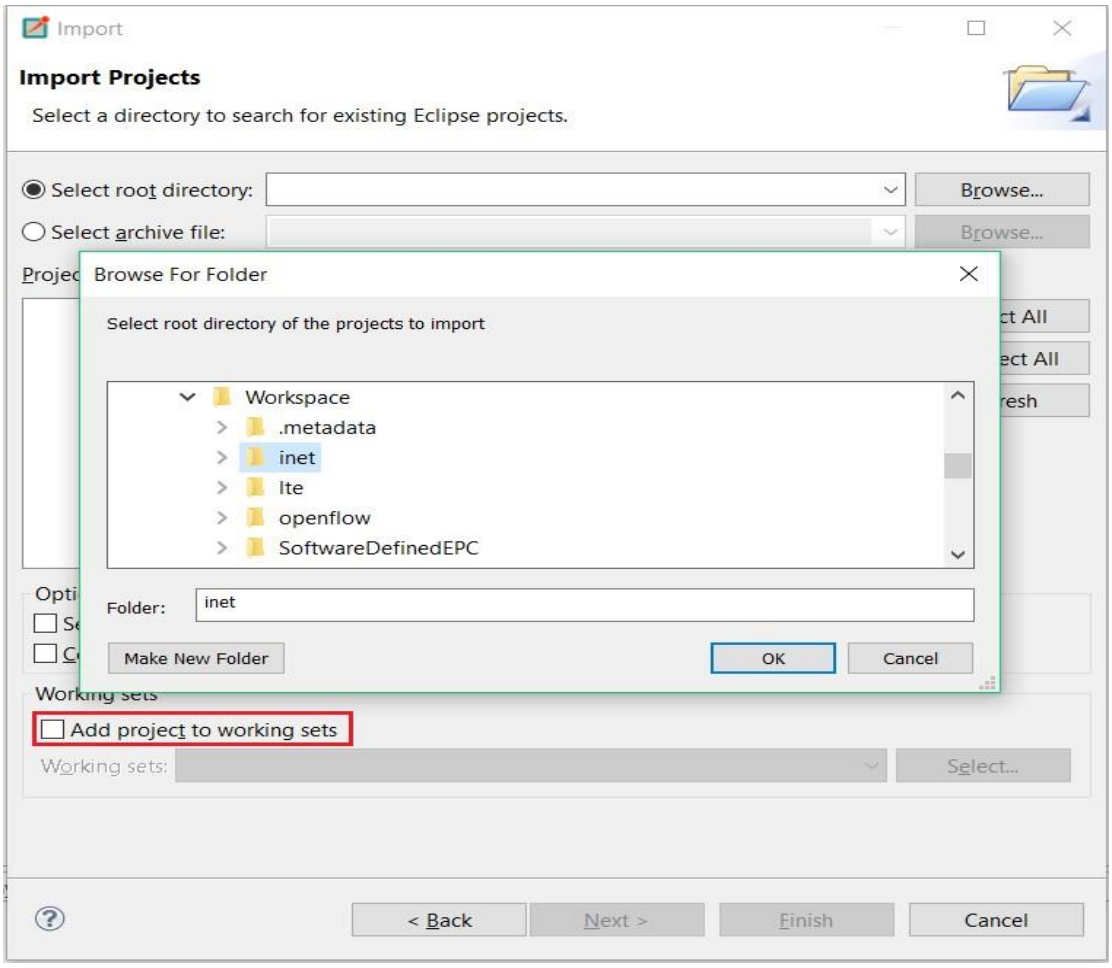


Figure A. 4: Import the source code of the INET library - step 3

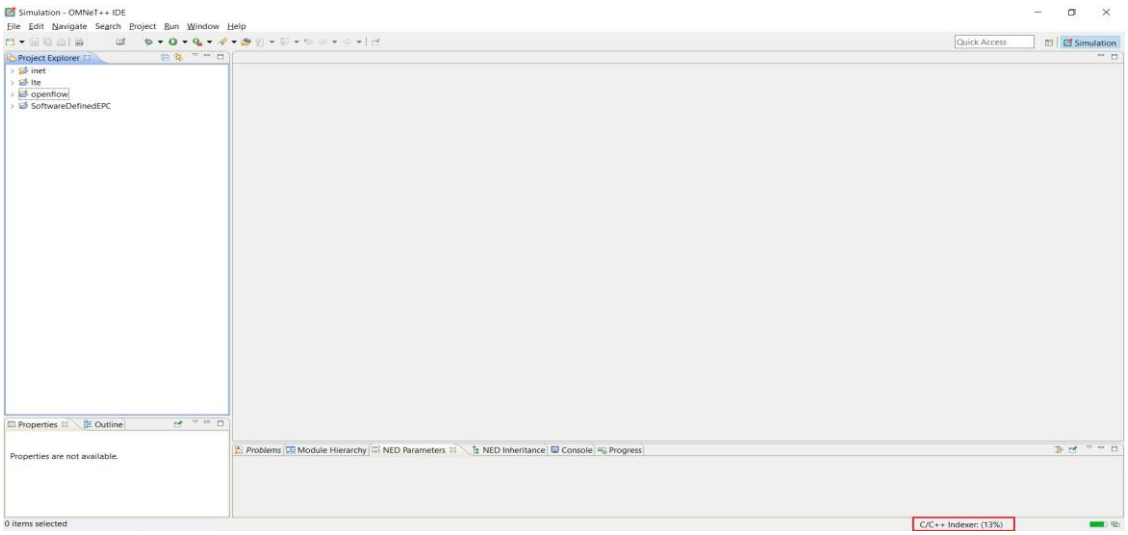


Figure A.5: Import the source code of the INET library - step

Figure A.9.

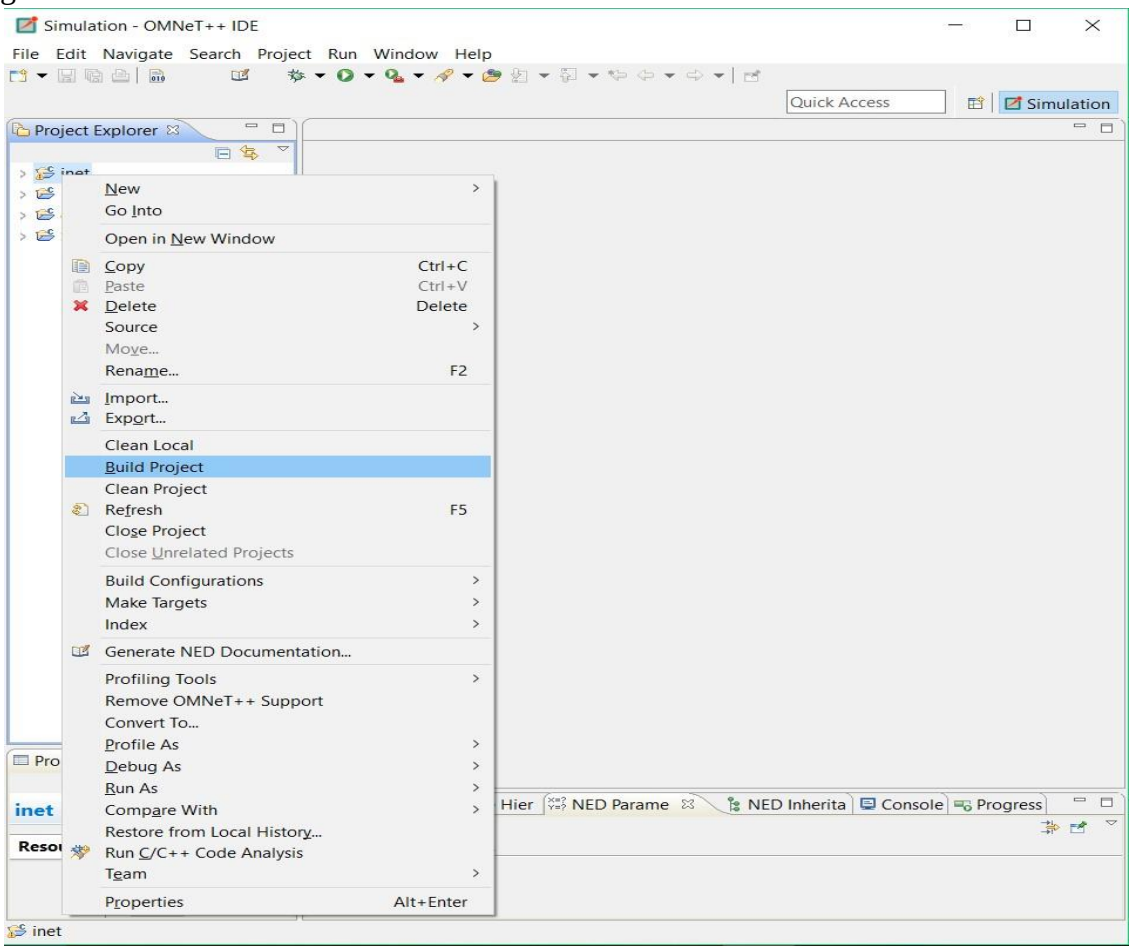


Figure A.6: Build the source code of the INET library - step 1

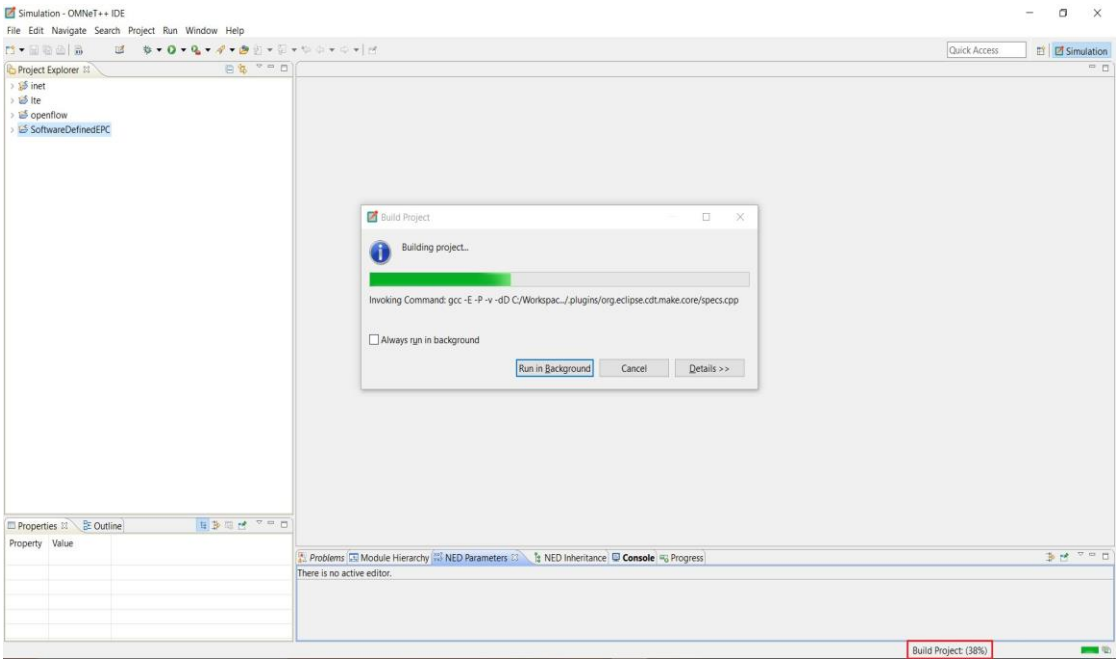


Figure A.7: Build the source code of the INET library - step 2

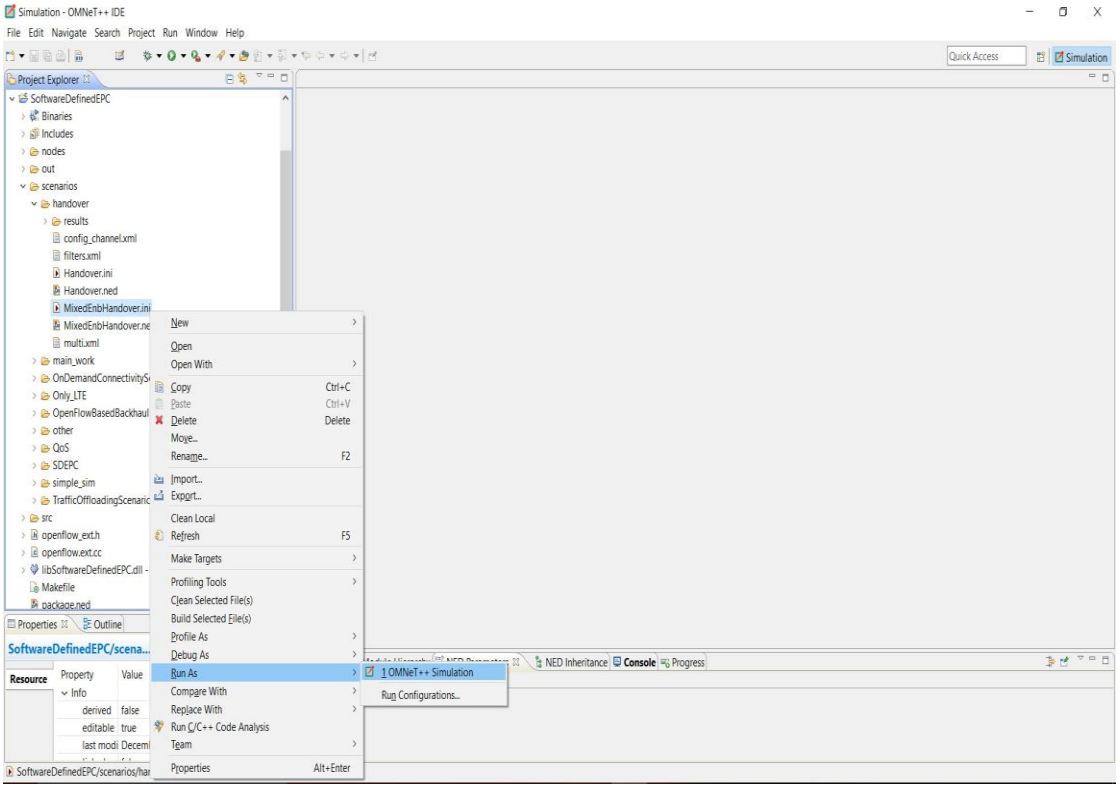


Figure A.8: Run a simulation in OMNeT++

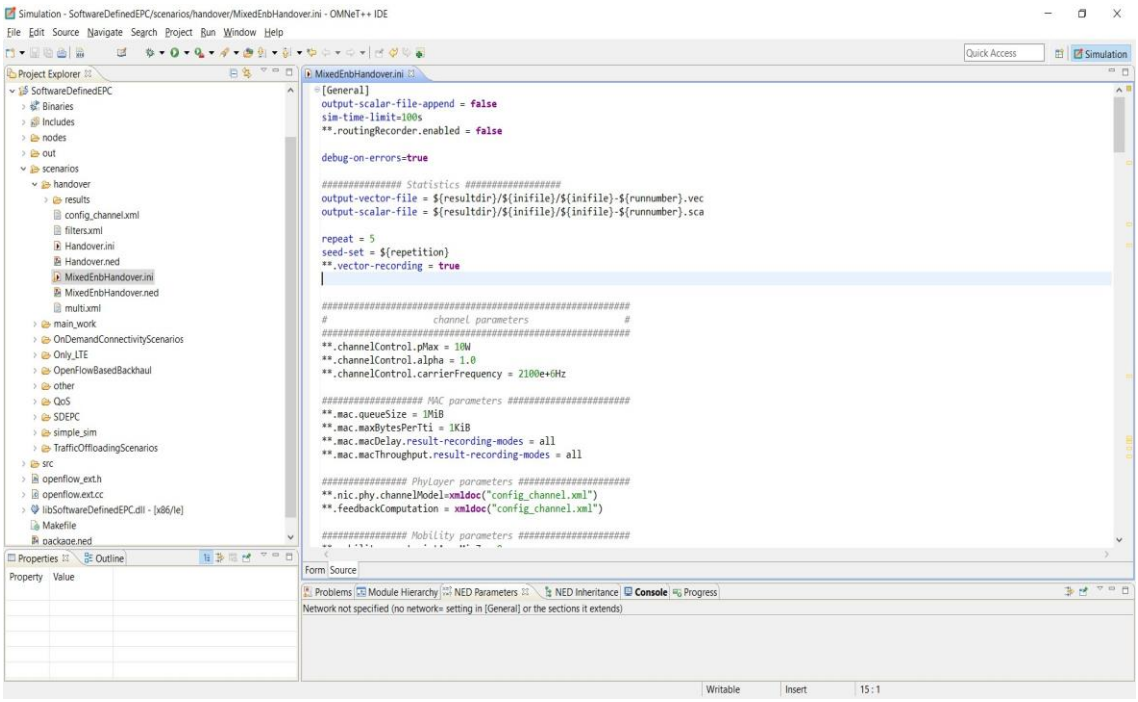


Figure A.9: Change the simulation parameters from the ini file