



**Ana Patrícia
Gomes da Cruz**

**Aplicação de mecanismos baseados em broker para
ligação de terminais a redes móveis**

**Application of broker-based mechanisms for
end-node attachment in mobile networks**



**Ana Patrícia
Gomes da Cruz**

**Aplicação de mecanismos baseados em broker para
ligação de terminais a redes móveis**

**Application of broker-based mechanisms for
end-node attachment in mobile networks**

“Ser-vos-ao dadas pistas e caminhos, contudo, o trabalho, a perseverança e a força de vontade, irão abrir-vos as portas dos segredos que a vossa dedicação e empenho conseguirem encontrar. Bem vindos ao vosso futuro.”

— Mário Jorge Rodrigues



**Ana Patrícia
Gomes da Cruz**

**Aplicação de mecanismos baseados em broker para
ligação de terminais a redes móveis**

**Application of broker-based mechanisms for
end-node attachment in mobile networks**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Daniel Nunes Corujo, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor António Manuel Duarte Nogueira, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Arnaldo Silva Rodrigues de Oliveira

Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Doutor Sérgio Miguel Calafate de Figueiredo

Engenheiro Sénior na Altran Portugal

Professor Doutor Daniel Nunes Corujo

Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Dedico este trabalho aos meus pais, irmã, cunhado e sobrinho, a quem agradeço pelo incansável apoio, por acreditarem em mim, estarem sempre disponíveis e proporcionarem todas as condições para que este caminho fosse possível. Aproveito também para agradecer às minhas primas Mariana e Matilde, por estarem sempre comigo neste percurso.

Em segundo lugar, agradeço ao Prof. Daniel Corujo e ao Prof. António Nogueira por todo o apoio prestado ao longo da realização deste trabalho. Aproveito também para agradecer ao Instituto de Telecomunicações de Aveiro (UID/EEA/50008/2019) por fornecer todos os recursos e condições necessárias para a realização desta Dissertação.

Por último, agradeço a todos aqueles que tive oportunidade de conhecer, de conviver e acima de tudo por me acompanharem nesta longa caminhada; em particular, ao Marco, à Inês Moreira, ao Ricardo, ao Diogo e à Cristiana por toda a amizade, apoio e ajuda disponibilizada.

A todos Vocês, que tornaram este caminho possível, um muito obrigada!

Esta dissertação foi realizada no âmbito do projeto de investigação PTDC/EEL-TEL/30685/2017 “5G-CONTACT - 5G CONTEXT-Aware Communications optimization” financiado pela FCT/MEC, e do projeto POCI-01-0247-FEDER-024539 “5G”, financiado pelo FEDER (através do POR LISBOA 2020 e do COMPETE 2020) e foi desenvolvida com o apoio do Instituto de Telecomunicações.

Palavras Chave

5G, 4G, EPC, 5GC, OAI, broker

Resumo

Nos últimos anos, o tráfego de dados móveis tem vindo a crescer com o aumento de equipamentos ligados à rede. Devido à demanda do utilizador, os operadores de rede necessitam de atualizar continuamente a sua rede e manter os custos baixos. Atualmente, para essa atualização, o operador precisa de adquirir novos equipamentos, tendo um investimento muito elevado. O 5G visa fornecer maior escalabilidade e flexibilidade na rede. Para isso, a arquitetura do sistema 5G é construída com base numa nuvem nativa, o que significa uma arquitetura baseada em serviços na rede “core”. Este tipo de arquitetura visa fornecer conectividade independentemente da tecnologia de acesso, introduzindo maior redundância na resiliência no plano de controlo e eficiência operacional. Adicionalmente, ao invés de interfaces dedicadas entre cada par de funções de rede “core” intervenientes, as mesmas comunicam através de uma interface baseada em serviços, com vista a uma maior flexibilidade e simplicidade.

OpenAirInterface (OAI) é uma plataforma de software de código aberto que visa fornecer uma aproximação aos padrões 3GPP das redes 4G e 5G. Esta dissertação fornece um estudo do impacto de uma arquitetura baseada em serviços no plano de controlo. Para isso, utilizou-se uma arquitetura que evolui o Evolved Packet Core (EPC) para uma rede “core” próxima da 5G Core (5GC), introduzindo um broker. Um broker é integrado entre os módulos do plano de controlo, no qual estes para comunicarem entre si necessitam de realizar pedidos. A abordagem utilizada consiste na integração de um broker na plataforma OAI, avaliando o seu impacto comparando com o EPC original.

Keywords

5G, 4G, EPC, 5GC, OAI, broker

Abstract

In recent years, mobile data traffic has been growing with the increase of equipments connected to the network. Due to user demand, network operators have to continuously upgrade their networks and keep the costs low. Nowadays, to do this upgrade, the operator needs to acquire new equipment, leading to a very high investment. 5G aims to provide more scalability and flexibility on the network. For this, the 5G system architecture is built based on a cloud-native, which means Service Based Architecture (SBA) in the core network. SBA aims to provide connectivity with all access technologies, introducing more redundancy in the control plane's resiliency and operational efficiency. Additionally, instead of using dedicated interfaces between each pair of interacting core functions, they now communicate through a Service-Based Interface (SBI), aiming for greater flexibility and simplicity. The OpenAirInterface (OAI) is an open-source software platform that aims to provide an approximation to the 3GPP standards of the 4G and 5G networks. This thesis provides a study of the impact of the SBA in the control plane. For that, we used an architecture that evolves the Evolved Packet Core (EPC) into a core network close to 5G Core (5GC) by introducing a broker. The broker is integrated between the modules in the control plane, wherein they have to order requests to communicate with each other. The proposed architecture consists of integrating the broker on the OAI platform, evaluating it, and comparing it with the original EPC.

Contents

Contents	i
List of Figures	vii
List of Tables	ix
Glossary	xi
1 Introduction	1
1.1 Motivation	2
1.2 Goals	2
1.3 Document structure	3
2 State of the Art	5
2.1 Evolved Packet System (EPS)	5
2.1.1 User Equipment	7
2.1.2 Access Network	8
2.1.3 Core Network	9
2.1.3.1 Mobility Management Entity (MME)	9
2.1.3.2 Home Subscriber Server (HSS)	10
2.1.3.3 Serving Gateway (S-GW)	10
2.1.3.4 Packet Data Network Gateway (P-GW)	10
2.1.3.5 Policy and Charging Rules Function (PCRF)	11
2.1.4 Communication Protocols	12
2.1.4.1 Radio Resource Control Protocol (RRC)	12
2.1.4.2 Packet Data Convergence Protocol (PDCP)	13
2.1.4.3 Radio Link Control Protocol (RLC)	13

2.1.4.4	Medium Access Control Protocol (MAC)	13
2.1.4.5	Physical Layer	13
2.1.4.6	S1 Application Protocol (S1AP)/Non-Access Stratum (NAS)	13
2.1.4.7	Diameter	14
2.1.4.8	GPRS Tunneling Protocol (GTP)	14
2.1.5	Connection Procedures	14
2.1.5.1	The EPS Bearer	15
2.1.5.2	LTE authentication	15
2.1.5.3	Attach Procedure	16
2.1.6	LTE implementation platforms	18
2.1.6.1	OpenLTE	18
2.1.6.2	srsLTE, srsUE, srsENB	18
2.1.6.3	OpenEPC	18
2.1.6.4	NextEPC	19
2.1.6.5	GR-LTE	19
2.1.6.6	Open-Source LTE Deployment (OSLD)	19
2.1.6.7	AMARISOFT LTE	19
2.1.6.8	OpenAirInterface (OAI)	19
2.2	5G network	23
2.2.1	Core network	24
2.2.1.1	Access and Mobility Management Function (AMF)	24
2.2.1.2	Session Management Function (SMF)	24
2.2.1.3	User Plane Functions (UPF)	25
2.2.1.4	Policy Control Function (PCF)	25
2.2.1.5	Network Exposure Function (NEF)	25
2.2.1.6	Network Repository Function (NRF)	25
2.2.1.7	Unified Data Management (UDM)	25
2.2.1.8	Authentication Server Function (AUSF)	25
2.2.1.9	Application Function (AF)	25
2.2.1.10	Unified Data Repository (UDR)	25
2.2.1.11	Network Slice Selection Function (NSSF)	26
2.3	Brokers	26
2.3.1	Software Solutions	26

2.3.1.1	RabbitMQ	26
2.3.1.2	Eclipse Mosquitto	28
2.3.1.3	Kafka	29
2.4	Summary	30
3	Architecture and Specifications	31
3.1	Problem statement	31
3.2	Requirements and Specifications	32
3.2.1	Selection of LTE implementation	32
3.2.2	OAI	33
3.2.3	Brokers	34
3.3	Proposed solution	35
3.3.1	Architecture	35
3.4	Introducing brokers	36
3.4.1	Attachment Procedure	37
3.5	Execution Procedure	38
3.5.1	Execution procedure for LTE	38
3.5.2	Execution procedure for message brokers	39
3.6	Summary	40
4	Implementation	41
4.1	Overview	41
4.1.1	Virtual Machine Specifications	42
4.2	Radio Access Network (RAN)	42
4.2.1	Evolved NodeB (eNB)	42
4.2.1.1	Hardware Setup	42
4.2.1.2	Software Setup	42
4.2.2	UE Setup	43
4.2.3	Run eNB and UE	43
4.3	Core Network	44
4.3.1	HSS + MME	44
4.3.1.1	Configuring and building HSS	44
4.3.1.2	Configuring and building MME	45
4.3.2	S/P-GW	45

4.4	Adaptations to HSS+MME and S/P-GW	45
4.4.1	RabbitMQ	47
4.4.1.1	Publisher/Subscriber	47
4.4.2	Mosquito MQTT	48
4.4.2.1	Publisher	50
4.4.2.2	Subscriber	50
4.4.3	Kafka	51
4.4.3.1	Publisher	51
4.4.3.2	Subscriber	51
4.4.4	Message broker VM	52
4.5	Extraction data from captures	52
4.6	Summary	54
5	Results	55
5.1	Scenario	55
5.2	Data Gathering	56
5.3	Signaling Impact	56
5.3.1	Messages defined by 3GPP	56
5.3.2	Size of messages using brokers	57
5.3.3	Throughput	58
5.4	Delay on the control plane	59
5.4.1	Delay between UE+eNB and MME entities	59
5.4.2	Delay between MME and HSS entities	61
5.4.3	Delay between MME and S/P-GW entities	62
5.4.4	Delay in broker entity	65
5.5	Attachment Time	65
5.5.1	Original EPC	65
5.5.2	Message Brokers	66
5.5.3	Comparison between original EPC and message brokers	67
5.6	Total Time	68
5.6.1	Original EPC	69
5.6.2	Message Brokers	70
5.6.3	Comparison total time between original EPC and message brokers	70
5.7	Latency	71

5.8 Summary	72
6 Conclusions	75
6.1 Future work	76
References	77
Appendix-A: oai-mme/oai-spgw source code modifications	81
A.1 Broker init connection configurations	81
A.2 Broker information structure	81
A.3 Changes on nwGtpv2cCreateAndSendMsg function	81
A.4 Init broker connection in s11_mme_task.c	82
A.5 Init publisher	82
Appendix-B: RabbitMQ implementation	85
B.1 publisher and subscriber methods	85
B.1.1 Establishment the connection	85
B.1.2 Private reply queue	85
B.1.3 Publish the message	85
B.1.4 Subscriber method	86
Appendix-C: Mosquitto implementation	87
C.1 Similar methods for publisher and subscriber	87
C.1.1 Create the client	87
C.1.2 Establishment the connection	87
C.2 Publish the message	87
C.3 Subscribe the message	88
Appendix-D: Kafka implementation	89
D.1 Create the client	89
D.2 Publish the message	89
D.3 Subscribe the message	89
Appendix-E: Extraction data from captures	91
E.1 Extraction S1AP captures	91
E.2 Extraction GTPV2 captures	92

E.3 Extraction Diameter captures	93
--	----

List of Figures

2.1	The Evolved Packet System network elements.	6
2.2	Functional split between E-UTRAN and EPC (adapted from [11]).	7
2.3	Internal architecture of the UE (adapted from [14]).	8
2.4	Architecture of E-UTRAN (adapted from [11]).	9
2.5	User plane protocol stack[15].	12
2.6	Control plane protocol stack[15].	12
2.7	EPS Bearer Architecture [11]	15
2.8	EPS attach procedure	16
2.9	OpenAirInterface protocol stack (source [36]).	20
2.10	Brief representation of the modules implemented by OpenAirInterface	21
2.11	Main architecture options proposed by 3GPP for 5G evolution.	23
2.12	5G Architecture [38]	24
2.13	Simplified RabbitMQ architecture.	27
2.14	RabbitMQ RPC system.	28
2.15	Simplified MQTT architecture using Mosquitto broker.	29
2.16	Simplified Kafka architecture.	30
3.1	LTE network elements implemented by OpenAirInterface [53].	33
3.2	Illustration of the integration of the broker between the core elements and eNB.	35
3.3	Proposed solution architecture.	36
3.4	Attachment procedure using a broker.	37
3.5	EPS execution procedure	38
3.6	Execution procedure using a broker.	39
4.1	Architecture implementation in Openstack cloud environment.	41
4.2	Generic initialization of the publisher and subscriber.	46

4.3	RabbitMQ flow using RPC method.	49
4.4	Mosquitto MQTT flow.	50
4.5	Kafka flow.	52
4.6	Implementation of the Flask server.	53
5.1	OAI with and without message broker scenario.	56
5.2	Delay time for messages exchanged with UE+eNB and MME for original EPC. .	60
5.3	Delay time for each pair of messages exchanged between HSS and MME entities.	62
5.4	Delay time comparison for the message pair msg6 for different implementations.	62
5.5	Delay time comparison for the message pair msg10 for different implementations.	63
5.6	Delay time comparison for the message pair msg11 for different implementations.	64
5.7	Delay time for each request and response messages in broker entity.	65
5.8	Attachment time for original EPC	66
5.9	Attachment time comparison between message brokers.	67
5.10	Attachment time comparison between original EPC and message brokers.	68
5.11	Total time for original EPC.	69
5.12	Total time using message brokers.	70
5.13	Total time comparison between original EPC and message brokers.	71
5.14	Comparison between original EPC and different brokers in terms of latency. . . .	72

List of Tables

2.1	Comparison of the network elements between 4G and 5G system.	26
3.1	Comparison between LTE implementations platforms.	33
4.1	Resources used by each VM.	42
4.2	OASIM file configuration.	43
4.3	Main configuration of the HSS files.	44
4.4	Example of user subscribers.	44
4.5	MME file configuration.	45
4.6	Version of server brokers.	52
5.1	Size of messages defined by 3GPP.	57
5.2	Size of messages sent through the different brokers.	57
5.3	Mean throughput per interface for different implementation.	58
5.4	Assign a delta message to the different pairs of messages.	59

Glossary

3G	Third Generation	DHCP	Dynamic Host Configuration Protocol
3GPP	Third Generation Partnership Project	DL	Downlink
4G	Fourth Generation	DNS	Domain Name System
5G	Fifth Generation	eMBB	Enhanced Mobile Broadband
5GC	Fifth Generation Core	EMM	EPS Mobility Management
AAA	Authentication, Authorization and Accounting	eNB	Evolved NodeB
AES	Advanced Encryption Standard	EPC	Evolved Packet Core
AF	Application Function	EPS	Evolved Packet System
AKA	Authentication and Key Agreement	E-RAB	E-UTRAN Radio Access Bearer
ALOE	Abstraction Layer and Open Operating Environment	ESM	EPS Session Management
AMF	Access and Mobility Management Function	E-UTRA	Evolved Universal Terrestrial Radio Access
AMQP	Advanced Message Queuing Protocol	E-UTRAN	Evolved Universal Terrestrial Radio Access Network
AN	Access Network	GBR	Guaranteed Bit Rate
APN	Access Point Name	GR	GNU Radio
AS	Access Stratum	GSM	Global System for Mobile Communications
AuC	Authentication Center	GTP	GPRS Tunnelling Protocol
AUSF	Authentication Server Function	GUMMEI	Globally Unique MME Identifier
AUTN	Authentication Token	GUTI	Globally Unique Temporary Identity
AV	Authentication Vector	HARQ	Hybrid Automatic Repeat Request
AVP	Attribute Value Pair	HLR	Home Location Register
BIOS	Basic Input/Output System	HPLMN	Home Public Land Mobile Network
CAPEX	Capital Expenditure	HSS	Home Subscriber Server
CK	Cipher Key	ICMP	Internet Control Message Protocol
CN	Core Network	ICT	Information and Communication Technologies
CPU	Central Processing Unit	IK	Integrity Key
		IMEI	International Mobile Equipment Identity
		IMS	IP Multimedia Subsystem

IMSI	International Mobile Subscriber Identity	PCRF	Policy and Charging Rules Function
IoT	Internet of Things	PDCP	Packet Data Convergence Protocol
IP	Internet Protocol	PDN	Packet Data Network
ITTI	Inter-task Interface	PDU	Packet Data Unit
LTE	Long Term Evolution	P-GW	Packet Data Network Gateway
MAC	Medium Access Control	PHY	Physical Layer
MBMS	Multimedia Broadcast Multicast Services	PLMN	Public Land Mobile Network
MBR	Maximum Bit Rate	PS	Packet Switched
MCC	Mobile Country Code	QCI	Quality of Service Class Identifier
ME	Mobile Equipment	QoS	Quality of Service
MME	Mobility Management Entity	RAM	Random Access Memory
mMTC	Massive Machine Type Communication	RAN	Radio Access Network
MNC	Mobile Network Code	RAND	Random Number
MQTT	Message Queue Telemetry Transport	RAT	Radio Access Technology
MSIN	Mobile Subscription Identification Number	RLC	Radio Link Control
MSISDN	Mobile Station International Subscriber Directory Number	RPC	Remote Procedure Call
MT	Mobile Termination	RRC	Radio Resource Control
N3IWF	Non-3GPP InterWorking Function	S/P-GW	Serving/PDN Gateway
NAS	Non-Access Stratum	S1AP	S1 Application Protocol
NEF	Network Exposure Function	SA	Standalone
NF	Network Function	SAE	System Architecture Evolution
NR	New Radio	SBA	Service Based Architecture
NRF	Network Repository Function	SBI	Service Based Interface
NSA	Non Standalone	SCM	Security Context Management
NSSAI	Network Slice Selection Assistance Information	SCTP	Stream Control Transmission Protocol
NSSF	Network Slice Selection Function	SDF	Service Data Flow
OAI	OpenAirInterface	SDR	Software Defined Radio
OASIM	OpenAirInterface System Emulation	SEAF	Security Anchor Function
OPEX	Operational Expenditure	S-GW	Serving Gateway
OS	Operating System	SIM	Subscriber Identity Module
OSA	OpenAirInterface Software Alliance	SM	Session Management
OSLD	Open-Source LTE Deployment	SMF	Session Management Function
PCEF	Policy and Charging Enforcement Function	SPR	Subscription Profile Repository
PCR	Policy Control Function	SRB	Special Radio Bearer
		TAC	Tracking Area Code
		TAI	Tracking Area Identity
		TCP	Transmission Control Protocol
		TE	Terminal Equipment
		TEID	Tunnel Endpoint Identifier

TFT	Traffic Flow Template	UPF	User Plane Functions
UDM	Unified Data Management	URLLC	Ultra-Reliable and Low Latency Communications
UDP	User Datagram Protocol	USIM	Universal Subscriber Identity Module
UDR	Unified Data Repository	USRP	Universal Radio Peripheral
UE	User Equipment	VM	Virtual Machine
UICC	Universal Integrated Circuit Card	VoIP	Voice over Internet Protocol
UL	Uplink	X2AP	X2 Application Protocol
UMTS	Universal Mobile Telecommunications System	XRES	Expected Response
UPF	User Plane Functions		

Introduction

Over the years, mobile telecommunications and the number of equipments needing a connection to the network have been growing. The Fourth Generation (4G) network provides high data rates and low complexity when compared with the previous generation networks.

In 2017, mobile data traffic increased by 71 percent. It is predictable that, by 2022, mobile data traffic keeps growing, reaching 77 exabytes per month. That points to approximately one zettabyte per year. Besides the expectations about the increasing mobile traffic of 4G, it is expected that Fifth Generation (5G) will reach 12 percent by 2022. Furthermore, until 2022, global mobile traffic is expected to rise seven-fold [1]. With the 4G evolution, new services appeared, and not only human customers take advantage of using it but also Internet of Things(IoT) services, such as cars, sensors, among others. The network operators are continuously upgrading and extending their network infrastructure to attend the user demand of data and services, increasing not only in Capital Expenditure (CAPEX) but also in Operational Expenditure (OPEX). On the other hand, the revenue per user is not enough to cover the investment, resulting in losses. To reduce costs and increase revenue, network operators need to progress their mobile networks to 5G [2].

5G network is being developed due to the rise of data traffic and demand for a high data rate as well as to reduce the costs. 5G considers three main scenarios: enhanced mobile broadband (eMBB), massive machine type communication (mMTC), and ultra-reliable low latency communications (URLLC). The 5G Core Network (5GC) architecture is built upon in Service Based Architecture (SBA), which takes advantage of the service concept, i.e., multiple services are provided by Network Functions (NF) and services communicate with each other through general interfaces [3]. Considering the HTTP-based (e.g., REST) nature of such interactions, this approach to web-based

technologies allows new ideas to be pursued in the domain of cloud-native mobile networks, such as the exploration of broker-based interactions between the elements of the mobile core network. Furthermore, 5GC provides architectural agility enabling 1:N redundancy for control plane resiliency, processing efficiency, and operational efficiency (CAPEX and OPEX)[4].

Third Generation Partnership Project (3GPP)¹ is an organization that develops telecommunications standards. This community works on cellular telecommunications technologies, such as radio access, core network and service capabilities, providing a complete system for mobile telecommunication [5]. Projects like OpenAirInterface(OAI)² are developed to emulate and simulate the 4G/5G networks following 3GPP standards.

1.1 MOTIVATION

The future network of 5G mobile telecommunications will depend on a reformulation of the architecture of its core, standardized by 3GPP in standard TS 23.501.

This new architecture is different from the previous ones by including mechanisms that allow the network to increase the capacity to integrate different types of usage scenarios, evidenced by the increasing integration of Information and Communication Technologies (ICT) in various sectors of our society.

The new architecture also shows an evolution of mobile network core architecture closer to cloud-native systems and technologies, allowing operators to benefit from added flexibility and cost savings. As a result, this opens up the way for further research concerning the impact of the adoption of other computer system-related mechanisms, such as the utilization of message brokers for control signaling routing.

Although there are some open-source software platform that implement the core network of the operator, like OAI, they are still associated with the 4G standard and there are no implementations with 5G.

1.2 GOALS

The focus of this thesis is to evolve the 4G network by introducing message brokers between core entities. That means we will use message brokers to make the signaling routing between the core entities, removing the dedicated interfaces of 4G. For that, we will use a software platform, OAI, developed with the 4G standard.

The OAI platform will be analyzed in terms of code structure, functionality, and what type of emulation it is possible to achieve. The message brokers will be integrated into the proper modules of the platform. Subsequently, the traffic will be analyzed for

¹3GPP: <https://www.3gpp.org/>

²OpenAirInterface: <https://www.openairinterface.org/>

both implementations with and without a broker. Furthermore, their impact will be evaluated with multiple users in the network.

1.3 DOCUMENT STRUCTURE

The remainder of this document is organized in the following way. Chapter 2 describes the relevant 3GPP standards of the 4G network as well as the different open-source 4G software platforms, the 5G network standards, and the brokers used for the implementation. Chapter 3 provides the requirements needed to build the solution and a detailed description of the proposed architecture. Chapter 4 presents the architecture's implementation details such as hardware and software that were used and all the configurations needed. Chapter 5 presents the analysis of the results gathered. Finally, chapter 6 presents the work conclusion and future work.

State of the Art

This chapter gives an overview of all the concepts that are necessary to understand this document. The goal of this thesis is to evolve the EPC into a core network close to 5GC, while integrating a broker. For that, it is first necessary to understand the scope of the 4G and 5G networks.

The first section of this chapter addresses the EPS by describing its architecture and exposing some LTE software platforms. The second section presents an overview of the 5G network. The last section describes different brokers.

2.1 EVOLVED PACKET SYSTEM (EPS)

In 2004, the 3GPP started to study the evolution of the 3G Universal Mobile Telecommunications System (UMTS) system. The reasons were to keep 3G system competitive. The first release documented of EPS specifications was Release 8, completed in 2008. The main motivations behind this were an optimized Packet Switched (PS) system, which answers the user's demand for Quality of Service (QoS) and high data rates, low complexity, and to keep the need for cost reduction, such as CAPEX and OPEX [6].

Simultaneously, the 3GPP community worked on the evolution of the core network called System Architecture Evolution (SAE), wherein it defines an all-Internet Protocol (IP) network architecture. There were two main reasons to design this new architecture. The first one was to have a “flat architecture” which means fewer network nodes were implicated in the user's data traffic, allowing performance and cost-efficiency. The second reason was to separate the signaling (known as control plane) from the user's data (known as user plane) to allow them to scale independently [7].

3GPP communications have continued to increase data rates, increasing the number of active subscribers at the same time, and improving performance, which was

documented on release 10, completed in 2011. Release 10, known as *LTE-Advanced*, corresponds to the 4G of the mobile telecommunications network [8].

There are three main components that integrate the EPS, namely the user equipment (UE), the access network (Long Term Evolution (LTE) or Evolved Universal Terrestrial Radio Access Network (E-UTRAN)), and the core network, the Evolved Packet Core (EPC). Figure 2.1 shows the functional components and the standardized interfaces of EPS.

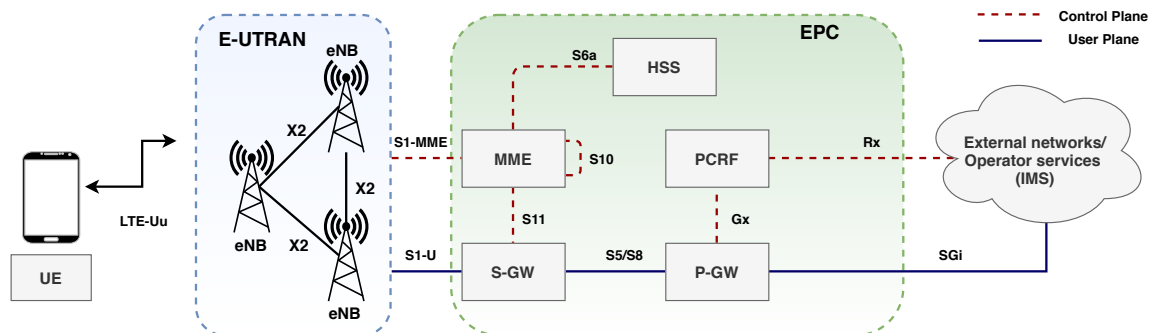


Figure 2.1: The Evolved Packet System network elements.

EPS provides to the user an IP address to a Packet Data Network (PDN) for accessing some services such as Voice over Internet Protocol (VoIP) and Internet. All data traffic is transported using bearers (section 2.1.5) through QoS policies from the gateway in PDN to the UE. Multiple bearers may be associated to each user to provide them with different QoS streams or to associate them to different PDNs. The E-UTRAN deals with tasks associated with radio functionality of the EPS like encryption and scheduling. The EPC deals with non-radio tasks [9].

As represented in figure 2.1, the core network has many logic nodes, whereas the access network only has one, the Evolved NodeB (eNB). Each one of these elements interconnects through standardized interfaces. It allows operators to choose their networks by splitting or merging these network elements [10]. Figure 2.2 depicts the principal functions between the E-UTRAN and EPC. The following subsections describe the network elements in more detail.

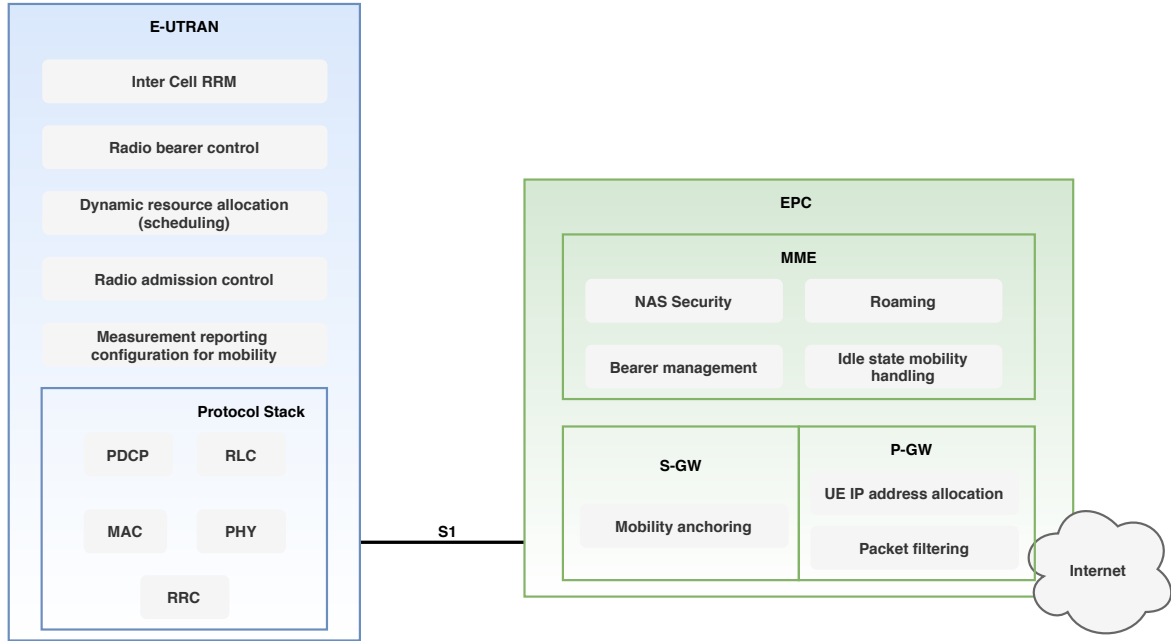


Figure 2.2: Functional split between E-UTRAN and EPC (adapted from [11]).

2.1.1 User Equipment

The architecture of the UE is the same as previous generations UMTS and Global System for Mobile Communications (GSM), shown in figure 2.3.

The mobile equipment (ME) is known as the present communication device. The ME is split into two components, the mobile termination (MT), which deals with all the communication functions, and the terminal equipment (TE), which terminates the data streams.

The universal integrated circuit card (UICC) is a smart card, known as the subscriber identity module (SIM) card. When the UICC is activated, the ME selects a universal subscriber identity module (USIM) application. After a successful USIM application selection, the USIM selected is stored on the UICC, which saves specific user data like the user's phone number and home network identity. The USIM executes different security-related calculations, having the secure keys stored in the smart card. The LTE network supports mobile devices that are using a USIM from Release 99 or later. However, it does not support the SIM used by previous releases of GSM [12], [13].

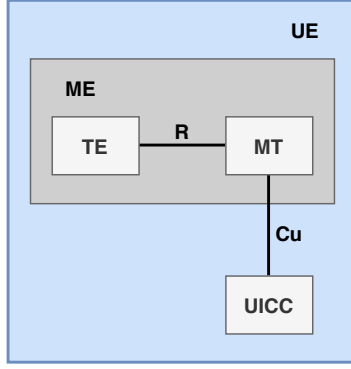


Figure 2.3: Internal architecture of the UE (adapted from [14]).

2.1.2 Access Network

Figure 2.4 represents the E-UTRAN architecture. The access network, E-UTRAN, deals with the radio communications between the mobile and the EPC and it is only composed by one component, the eNB. The E-UTRAN consists of a network of eNBs, providing the Evolved Universal Terrestrial Radio Access (E-UTRA) user plane and control plane terminations towards the UE (more information concerning protocols on section 2.1.4).

Each eNB may be connected to each other through the X2 interface. It also connects to EPC using the S1 interface that subdivides into S1-MME, connected to Mobility Management Entity (MME), and S1-U, connected to Serving Gateway (S-GW). The S1 interface has the functionality to support multiple connections between MME/S-GW.

The eNB is responsible for all radio related roles like functions for radio resource management. It covers features associated with the radio bearers like radio bearer control, radio admission control, connection mobility control, scheduling, and dynamic allocation of resources to UEs in both uplink and downlink. Furthermore, it determines which of the MMEs would receive the signaling sent from UE. This MME will serve the UE while it is in the radio coverage of the pool area wherein the MME is associated. However, when the UE moves to a new pool area and into a new pool area, the new MME will send the Identification Request or the Context Request messages to the old MME using Globally Unique Temporary Identity GUTI [11], [15].

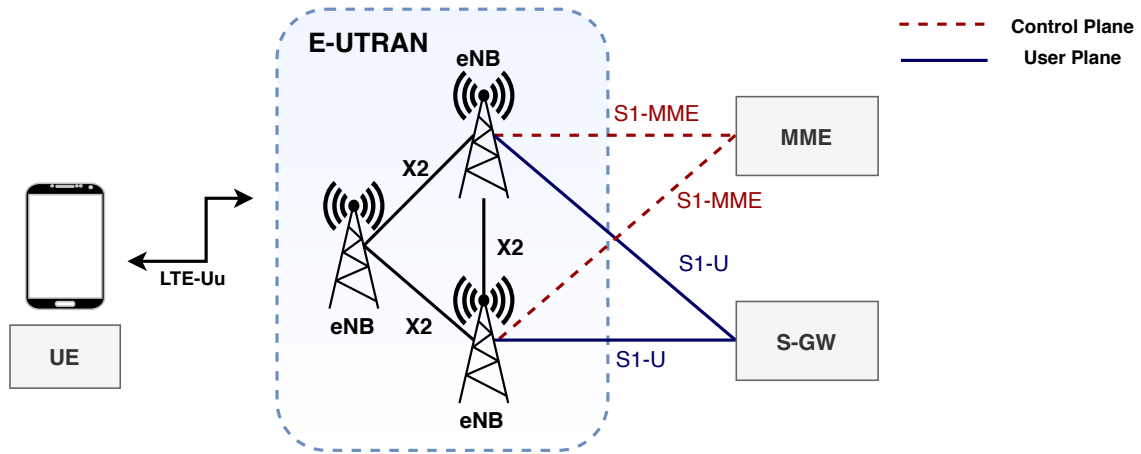


Figure 2.4: Architecture of E-UTRAN (adapted from [11]).

2.1.3 Core Network

The core network, known as EPC, is the cornerstone of the EPS. It is responsible for the overall control of the UE and the establishment of bearers. The EPS provides the bearer path with some QoS policies. However, the control of multimedia applications (VoIP) is supplied by the IP Multimedia Subsystem (IMS), which is designed to be outside the EPS.

The elements that compose the core network displayed in figure 2.1 are discussed below in more detail.

2.1.3.1 Mobility Management Entity (MME)

The MME is the main control plane element of the LTE network. The MME processes Non-Access Stratum (NAS) signaling between the EPC and the UE, which is responsible for idle mode UE tracking and paging procedures. The main functions supported by the MME are classified as:

- bearer related functions which include the establishment, maintenance and release of the bearers, and is handled by the session management layer in the NAS protocol;
- connection management functions which include the establishment of the connection and security between the network and the UE, and is handled by the mobility management layer in the NAS protocol.

A typical network may include multiple MMEs. Each of them are selected for operations based on the geographical location or distances from the UE. After being selected, the MME selects other network elements required for the service, such as S-GW and P-GW. The MME gets and stores the location reports, sent by the UEs. When the UE needs to be paged, it uses this data to do so. The MME also presents a significant role in the authentication process in interfaces with HSS for getting and

providing subscribers data. When there are multiple MME in the LTE network, the S10 interface is used to connect them. When a new MME is selected to serve a specific UE, the new MME can contact the old MME through S10 to retrieve data about the identity(MME), security information, and information related to active bearers (PDN gateways to communicate, QoS) [15].

2.1.3.2 Home Subscriber Server (HSS)

The HSS is the central subscriber database of the LTE networks. It is a combination of the Home Location Register (HLR) and Authentication Center (AuC) from previous 3GPP versions. The HSS stores permanent subscription details for its subscriber in the network, including the authentication key. It also saves the temporary mobility and service data for every subscriber. During a user's authentication, the HSS provides the MME with an authentication vector based on the authentication key, that can verify if a specific user can be allowed to connect to PDN. However, the authentication key itself never leaves the HSS [16].

2.1.3.3 Serving Gateway (S-GW)

The S-GW terminates the user plane towards E-UTRAN. For each UE associated with the EPS, there is a single S-GW, at a given time. One S-GW could connect to multiple eNBs, and one eNB could connect to various S-GWs.

Moreover, this entity works as a mobility anchor point for inter-eNodeB and intra-3GPP handover without changes in the S-GW. Concerning inter-eNodeB handovers, the S-GW is responsible for notifying the source eNB after switching the path and not receiving traffic for handed over UE. When the UE is in idle mode, the S-GW buffers the downlink packets to this user and initiates a network triggered service request procedure by the MME. When the UE moves, the S-GW might change, which is selected by the related MME.

Other roles include [15]:

- lawful interception
- packet routing and forwarding;
- transport level packet marking in the Uplink(UL) and Downlink(DL) (based on the metrics of the associated EPS bearer) .

2.1.3.4 Packet Data Network Gateway (P-GW)

The P-GW is the entity that communicates the data plane traffic to and from external PDNs through the SGi interface. The network usually includes one or more P-GWs for each type of external PDN connection, such as the Internet, the network operator's servers, or the IMS. Each PDN is identified by an access point name (APN). Also, the P-GW can provide connectivity to UEs using non-3GPP access networks. The P-GW

is also responsible for allocating the IP addresses to the UEs. Each mobile device is assigned to a default P-GW [13].

The main functions include [16]:

- per-user based packet filtering (by e.g., deep packet inspection);
- lawful interception;
- the transport level packet marking in the UL and DL is based on the quality of service class identifier (QCI) of the associated EPS bearer;
- UL and DL service level charging, rate enforcement and gating control;
- DL rate enforcement based on the accumulated maximum bit rates (MBRs) of the aggregated of service data flows (SDFs) with the same guaranteed bit rate (GBR) QCI;
- DHCP (server and client) functions.

Furthermore, this entity includes functions UL and DL bearer binding, i.e., represents an association between an SDF and a bearer in the access network to transport that SDF. Some functionalities, like SDF detection, policy enforcement, and flow-based charging, are supported by the entity, namely Policy and Charging Enforcement Function (PCEF), which is located at the gateway [17]. The S-GW and P-GW entities, according to 3GPP standards, could be implemented as separated entities or as a single entity, removing the S5/S8 interface [15].

2.1.3.5 Policy and Charging Rules Function (PCRF)

PCRF is the entity responsible for policy control decision concerning SDF detection, gating, QoS policies and flow based charging towards the PCEF, which resides in the P-GW. All the security procedures required by the operator are applied before this entity accepts service information from the application function (AF). This entity, also, has to decide how a specific subscription profile repository (SPR) shall be treated in the PCEF, and ensure that the PCEF user plane traffic mapping and treatment is by the user's subscription profile. So, its inputs are the SPR, the AF, and the predefined rules. The SPR is a logical entity containing all subscription-related information needed for subscription-based policies, for example, the subscriber's allowed services, and the information on the subscriber's allowed QoS. The AF is an element that extracts session information from the server application and provides it to the PCRF. The PCRF authorizes QoS resources. It uses the service information received from the AF and the subscription information received from the SPR to calculate the proper QoS authorization. Also, it may take into account the request QoS received from the PCEF through Gx interface [10], [17].

2.1.4 Communication Protocols

All interfaces mentioned previously are associated with a protocol stack, which the network elements use for data and signaling messages. The protocol stack consists of two planes: the user plane, which handles the user data traffic, and the control plane, which handle signaling messages that are just for control between the network elements. Figure 2.5 and figure 2.6 present the protocols associated to user plane and control plane, respectively, and each of their interfaces. The main functions of the different protocols are described below [11], [13], [18].

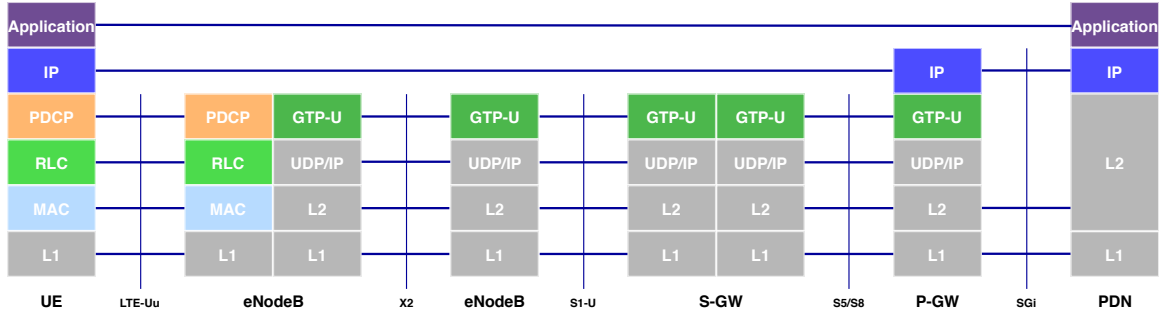


Figure 2.5: User plane protocol stack[15].

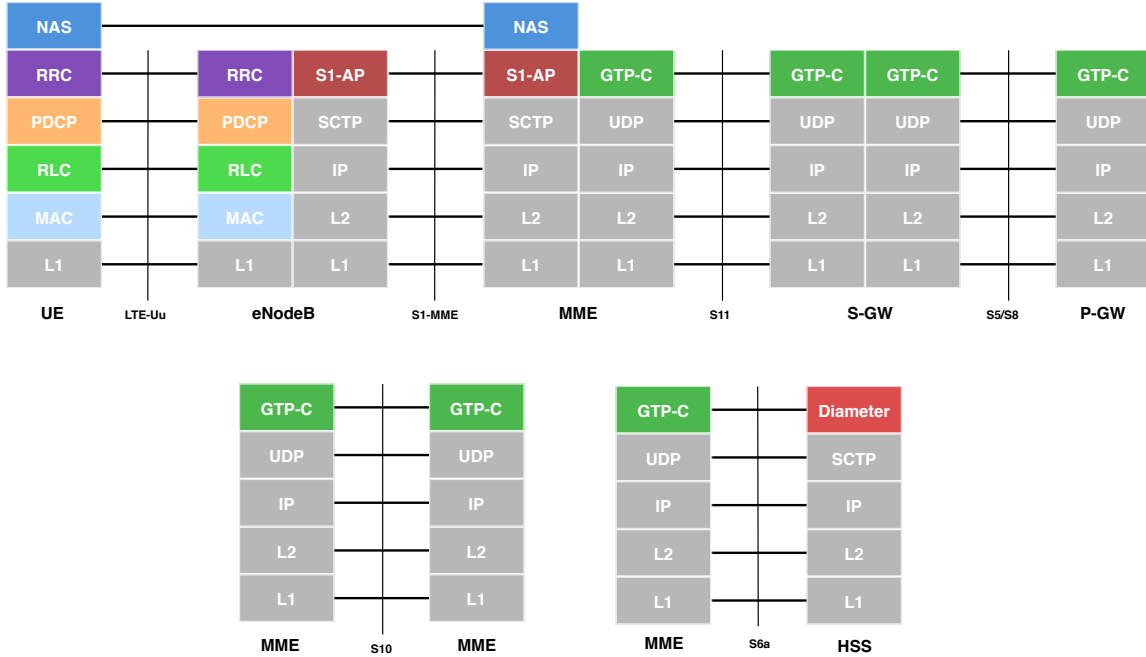


Figure 2.6: Control plane protocol stack[15].

2.1.4.1 Radio Resource Control Protocol (RRC)

The radio resource control (RRC) sublayer is responsible for handling the access network-related procedures, including the broadcast of system information that the device needs

to communicate with a cell (non-access stratum (NAS) and access stratum (AS)). Also, it is responsible for the transmission of paging messages from the MME to the device to notify about new connection requests, configure radio bearers, and mobility functions such as handover, and UE cell (re)selection.

2.1.4.2 Packet Data Convergence Protocol (PDCP)

The packet data convergence (PDCP) sublayer manages RRC messages in the control plane and IP packets in the user plane. Regarding the user plane, it performs header compression and decompression, ciphering and deciphering, and retransmission during handover. For the control plane it involves ciphering and integrity protection.

2.1.4.3 Radio Link Control Protocol (RLC)

The radio link control (RLC) sublayer is responsible for concatenation, segmentation of IP packets, and tracks packets that were sent or received. It has three modes of transmission: Transparent Mode (TM), Unacknowledged Mode (UM), and Acknowledged Mode (AM).

2.1.4.4 Medium Access Control Protocol (MAC)

The medium access control (MAC) sublayer handles the mapping between logical and transport channels. It also handles uplink and downlink scheduling, multiplexing/demultiplexing of logical channels, and error correction through hybrid automatic repeat request (HARQ). The MAC supplies services to the RLC in the form of logical channels.

2.1.4.5 Physical Layer

The physical layer (PHY) provides services to the MAC sublayer by using transport channels. Besides, it handles coding/decoding, modulation/demodulation, among others.

2.1.4.6 S1 Application Protocol (S1AP)/Non-Access Stratum (NAS)

The MME controls the eNB into its pool area through the S1AP. The main functions related to the S1AP are E-RAB management functions (setting up, modifying and releasing E-UTRAN Radio Access Bearers (E-RAB) that are triggered by the MME), initial context transfer function, mobility functions, UE capability Info Indication function, paging, NAS signaling transport function between the UE and MME, and location reporting which allows MME to know UE's current location [19]. S1AP utilizes Stream Control Transmission Protocol (SCTP).

The NAS protocol establishes a logical connection between the UE and the MME. The eNB transmits the NAS messages between LTE-Uu and S1-MME interfaces. The main functions are:

- the support of mobility of the UE;

- the support of session management procedures to establish and maintain IP connectivity between the UE and a P-GW.

The NAS procedures are grouped into two categories: EPS mobility management (EMM), which is related with mobility over E-UTRAN access, authentication and security, and EPS session management (ESM), which offers support to the establishment and handling of user data [20].

2.1.4.7 Diameter

The diameter protocol aims to provide authentication, authorization, and accounting (AAA) for applications such as network access or IP mobility and runs over Transmission Control Protocol (TCP) or SCTP. It gives the ability to exchange messages and delivery Attribute Value Pairs (AVPs), capabilities negotiation, error notification, and extensibility, which provide the addition of new applications, commands, and AVPs. All delivered data is in the form of AVPs. The use of the AVPs by this protocol is to provide [21]:

- user authentication information, which enables the Diameter server to authenticate the user;
- service specific authorization information, between clients and servers, to allow whether a user's access request should be granted;

The S6a interface is responsible for enabling the transfer of subscriber related data between the HSS and MME, which is defined in [22].

2.1.4.8 GPRS Tunneling Protocol (GTP)

The GTP is a tunneling protocol with a version for control plane (gtpv2) and a version for user plane (gtpv1). Both versions run over the User Datagram Protocol (UDP). To separate traffic into different communications flows GTP tunnels are used. A GTP tunnel in each node is identified with a Tunnel Endpoint Identifier (TEID), an IP address and a UDP port number. All data traffic is transported by using bearers based on GTP, and there is at least one tunnel for each UE attached. Between the S-GW and the P-GW, the interface could be called S5 or S8, according to the scenario. If it is a roaming scenario, the interface is called S5. Otherwise, if it is a non-roaming scenario, the interface is called S8, where typically the S-GW is in the visited network, and the P-GW is in the home network.

2.1.5 Connection Procedures

The following sub-sections describe the user connections procedures such as the authentication and attachment and the way that the user can request a service to the network.

2.1.5.1 The EPS Bearer

The EPS bearer is a tunnel responsible for transporting all data traffic between the UE and the P-GW, with a specific QoS. An EPS bearer consists of one or more service data flows such as streaming video application, and each service data flow comprises one or more packet flows like video streaming. All the packets that flow into a specific EPS bearer has the same QoS.

Whenever a mobile connects to a PDN, a default EPS bearer is created and remains established until the EPS session is finished. At the same time, the mobile receives an IP address to communicate with that network. The default bearer has a default QCI and a maximum bit rate. After the establishment of the default bearer, if an application or service needs particular QoS policies, a dedicated bearer is created. The dedicated bearer offers better QoS than the default bearer and can have a GBR.

Each EPS bearer is associated with a traffic flow template (TFT) that consists of a set of packet filters, and that filtering could be a port number or IP address.

The EPS bearer covers three different interfaces, so it cannot be implemented directly. The solution for this problem is broken down in three lower-level bearers, namely the radio bearer, the S1 bearer, and the S5/S8 bearer. Each of these has specific QoS policies. The radio bearer is implemented with a proper configuration of the LTE-Uu interface protocols, while the S1 and S5/S8 bearer are implemented using GTP tunneling. The E-RAB is a combination of the radio bearer and S1 bearer, which are illustrated in the figure 2.7 [13], [15].

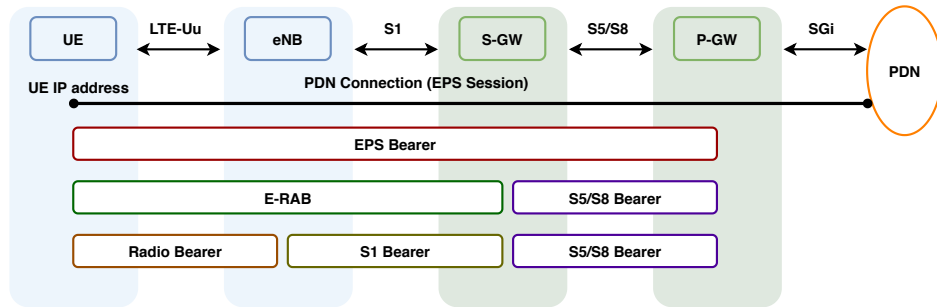


Figure 2.7: EPS Bearer Architecture [11]

2.1.5.2 LTE authentication

The authentication consists of the process to verify if a user is an authorized subscriber to the network that he is trying to access. The EPS Authentication and Key Agreement (AKA) corresponds to the mutual authentication and key agreement procedure used throughout E-UTRAN, between the UE and the MME.

After an attach request received from the UE, the MME knows the identity of the user through its IMSI, and it requests an authentication vector (AV) from the HSS. The HSS generates AVs, which are composed by the next keys:

- Integrity Key (IK)
- Cipher Key (CK)
- Random Number (RAND)
- Expected Response (XRES)
- Authentication Token (AUTN)

After receiving the AV, the MME sends the RAND and AUTN to the UE. Then, for the received keys, the UE computes the response and sends it back to the MME. After that, MME compares the received response from the UE with the XRES received from the HSS. Then, if the responses match, the authentication will be successful. All this exchange of keys is possible through the NAS protocol (see section 2.1.4) [23], [24].

2.1.5.3 Attach Procedure

The EPS attach procedure for an user is described in figure 2.8.

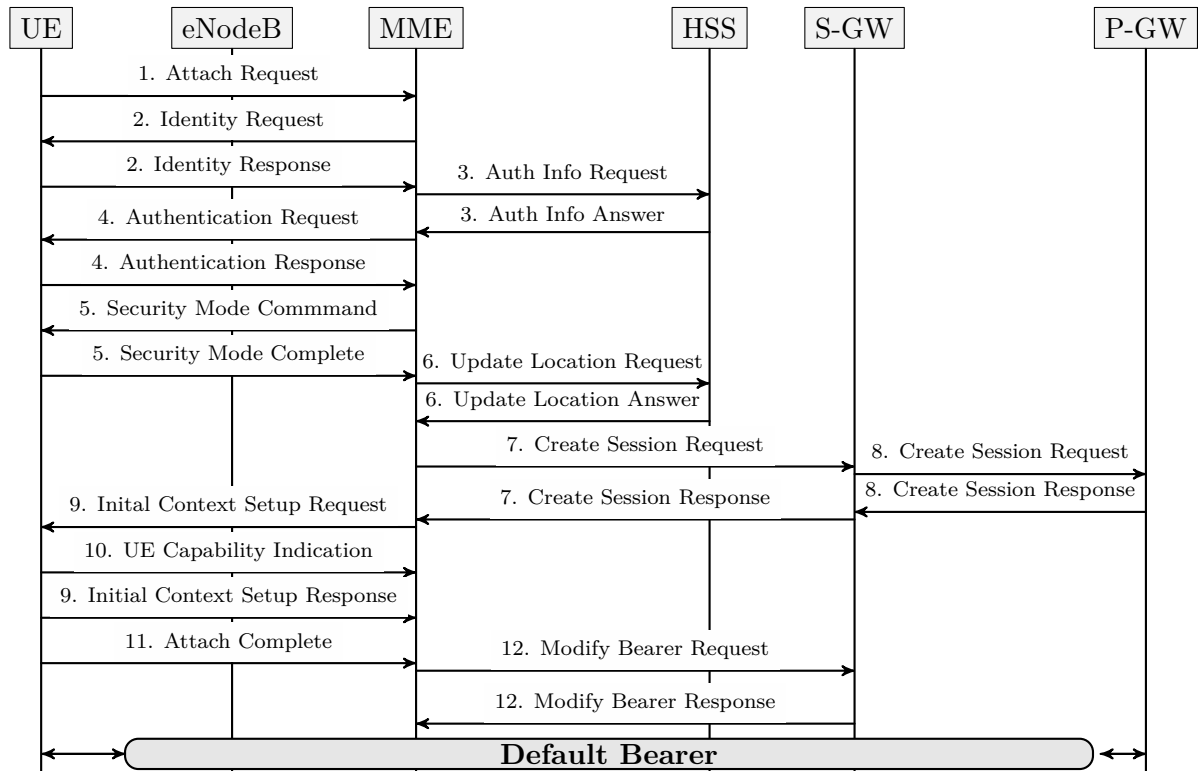


Figure 2.8: EPS attach procedure

1. The eNB forwards the Attach Request received by the UE in a S1-MME control message (Initial UE message) including the GUTI of the UE received from the last attach and the PDN connectivity request.

2. When the UE is unknown in the MME, the MME sends an Identity Request to the UE to request the IMSI. The UE answers with Identity Response (IMSI).
3. The MME sends an Authentication Information Request message to the HSS, requesting AVs for the UE that the IMSI is known.
4. The MME sends an Authentication Request to the UE with the information (RAND and AUTN) to generate AV. The UE responds with RES. Upon the MME receives the message it compares the RES value generated by the UE and the XRES sent from the HSS, to authenticate the user.
5. After the user authentication is completed, the MME starts the NAS security setup procedure. The MME selects algorithms of ciphering and integrity received from the UE from the Attach Request message. After selecting the algorithms, it informs the UE by sending a Security Mode Command message. When the UE receives the message, it generates NAS security keys through the algorithms selected by the MME and executes an integrity validation. If the message passes the integrity check, the UE sends a Security Mode Complete to the MME to inform that NAS security was successfully generated.
6. The MME sends an Update Location Request message to the HSS, which includes the IMSI and MME ID, to register the UE and acquire the subscription information of the UE. The HSS registers the MME ID of the MME where the user is located. As response (Update Location Answer), the HSS sends the subscription information of the user. The subscription information permits the MME to create the EPS session and the default EPS bearer for the subscriber.
7. Now, all the information is available for the MME to set up the default EPS bearer. The MME begins by selecting a P-GW regarding the APN received from the HSS user subscription. The MME sends a Create Session Request to the S-GW, which includes relevant subscription information and destination P-GW.
8. The S-GW receives the message and sends it to the P-GW. This message includes a GTP tunnel endpoint. The P-GW creates a GTP tunnel endpoint and sends the Create Session Response to the S-GW, which includes the IP address allocated for the UE and QoS of the default bearer. When the S-GW receives the TEID from the P-GW, the S5/S8 bearer is established. The S-GW forwards the message to the MME and creates a GTP tunnel endpoint for the S1 bearer.
9. The MME updates the information related to the UE and sends it to the eNB through the Initial Context Setup Request message, such as the IP address allocated. After receiving the message, the GTP tunnel endpoint is created for the S1 bearer and the TEID is sent to the MME.
10. The UE sends all the details of its capabilities to the MME .
11. The eNB sends an Attach Complete message to the MME.

12. The MME updates the S1 bearer and sends the information via Modify Bearer Request message to the S-GW. And the S-GW sends an acknowledged to the MME. The bearer is now established

2.1.6 LTE implementation platforms

There are several open-source software-based platforms available, wherein each of them holds different characteristics and implementation, offering simulation or monitoring in real-time. The following sections describe some open-source solutions.

2.1.6.1 *OpenLTE*

OpenLTE is an open source implementation of the 3GPP LTE specifications. Its main focus is the transmission and reception of the downlink. However, the current goal is to expand the capabilities of the GNU Radio applications and inserting capabilities to a simple eNB application. These build a simple eNB in a straightforward EPC, written in Octave, C++ and Python.

In this project there is no UE implementation, and many features are undeveloped or unstable. Furthermore, it requires a huge amount of processing power and also, a very low latency. If there is any delay in the processing, the system will not respond in time and will lose samples [25].

2.1.6.2 *srsLTE, srsUE, srsENB*

srsLTE¹ is an open source LTE library developed by Software Radio Systems², written in C, following the 3GPP Release 10. Besides, this project is composed by srsUE and srsENB, written in C++. This library proposes a basic light-weight implementation of the EPC.

srsUE and srsENB are a complete software radio LTE UE and eNB, respectively, built on the srsLTE library. srsUE application presents all layers from PHY to IP. srsUE can connect to any LTE network and provides high-speed mobile connectivity through the standard interface. Transmitting and receiving radio signals are a requirement by a Software Defined Radio (SDR) like Ettus Research USRP. srsENB can connect with any EPC. When this connection occurs, a local LTE cell is created. Furthermore, this project uses some security functionalities and NAS messages from OpenLTE project [26], [27].

2.1.6.3 *OpenEPC*

OpenEPC provides a full implementation of the EPC, following the 3GPP Release 8 to 12. The current version available, includes all the components that compose the

¹srsLTE: <https://www.srslte.com/>

²Software Radio Systems: <https://www.softwareradiosystems.com/>

3GPP architecture including the interfaces with different access technologies and service platforms [28]. However, this implementation is not an open source solution.

2.1.6.4 NextEPC

NextEPC³ is an opensource implementation of the EPC of LTE networks supporting 3GPP Release 13. This project presents an implementation of the MME, HSS, S-GW, P-GW, and PCRF; however, it does not present any implementation of eNB neither UE, which is a limitation of testbed [29].

2.1.6.5 GR-LTE

GR-LTE is an open-source project, whose aim is to provide a modular environment for a LTE DL, reaching this by giving signal processing blocks into the GNU Radio framework. This implementation could be considered as an UE, and does not present a deployment of eNB or EPC [30].

2.1.6.6 Open-Source LTE Deployment (OSLD)

The OSLD aims are to provide open-source SDRs and shared deployment of software for wireless communications systems. The project consists on a modular LTE library for mobile terminals and base stations. Therefore, it uses the open-source SDR framework ALOE. This project was terminated and does not provide an implementation of the EPC [31].

2.1.6.7 AMARISOFT LTE

Amarisoft LTE⁴ is the most complete platform that implements 4G system, but it requires a paid license to use it. This platform provides LTE release 14 compliant EPC and eNB, Multimedia Broadcast Multicast Services (MBMS) gateway and IMS server. The EPC handles UE procedures such as, attach, authentication, radio bearer establishment, etc. The component of UE is developed in LTE release 8 support features up release 14, allowing simulation of hundreds of UEs. It also, provides a component of the 5G system, New Radio compliant release 15 [32].

2.1.6.8 OpenAirInterface (OAI)

OAI is an open-source platform developed to emulate 4G/5G networks following the 3GPP standards, written in C and C++. Initially, this project started to be developed by EURECOM⁵, but now it is managed by the OpenAirInterface Software Alliance (OSA)⁶. This platform offers a software-based implementation of the LTE network and it

³NextEPC: <https://nextepc.com/>

⁴Amarisoft LTE: <https://www.amarisoft.com/>

⁵Eurecom: <http://www.eurecom.fr/en>

⁶OpenAirInterface: <https://www.openairinterface.org/>

comprises the entire protocol stack of 3GPP standards both in E-UTRAN and EPC. Furthermore, it can be used to customize and build the 4G network (UE, eNB, and EPC) on a computer (Intel x86 processors).

Commercial UEs or OAI UEs can be connected to the OAI eNB to test distinct configurations and monitor the network in real-time. It also provides a hardware platform supporting drivers and firmware such as USRP, BladeRF, and EXMIMO board [33]–[35]. Moreover, it provides an environment with diverse of built-in tools such as soft monitoring and debugging tools, realistic emulation modes, protocol analyzer, and configurable logging system for all layers [35].

As mentioned before, the OAI is an implementation of the 3GPP specifications and provides a full implementation of the protocol stack. Figure 2.9 shows the LTE protocol stack implemented by OAI. According to [36], some features of the E-UTRAN are the integrity check and encryption using Advanced Encryption Standard (AES). Standards S1AP and GTP-U interfaces offer communication with the core network.

The EPC features are:

- reuses standards compliant of GTPv1u and GTPv2c protocols from the open-source software implementation of EPC namely nwEPC⁷.
- NAS integrity and encryption using AES.
- UE procedures handling: attach, authentication, service access, radio bearer establishment.
- transparent access to the IP network.

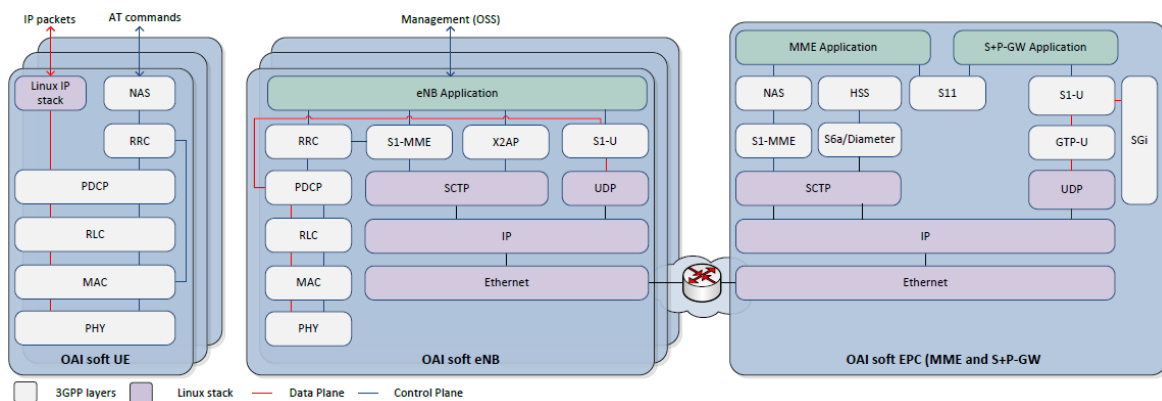


Figure 2.9: OpenAirInterface protocol stack (source [36]).

The OAI software is obtained from the Eurecom’s Gitlab and OpenAirInterface’s GitHub. It is on the Eurecom’s Gitlab where most information regarding this platform is provided⁸. This includes its tutorials, the software or hardware that they recom-

⁷nwEPC: <https://sourceforge.net/projects/nwepc/>

⁸OpenAirInterface5G-wiki: <https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/home>

mend to use, the main configuration files that we need to modify, the recommended simulator/emulator that we need to use according to our objectives, among others.

Another recommendation of this community is regarding the code. OAI recommends to use the code of master branch instead of the development branch. The development branch consists of recently updated version code, and the master branch stable version of the code, updated once every 2-3 months. Furthermore, when the master branch is updated, a tagged release is created, and this tag consists of the latest stable.

This community is very active, and anyone can communicate through the mailing list. All the questions (problems, bugs discovery, doubts) published are visible for all subscribers allowing them to clarify other people with the same question.

The OAI source code splits into `openairinterface5g`⁹ (eNB RAN + UE RAN) and `openair-cn`¹⁰ (EPC), as represented in figure 2.10.

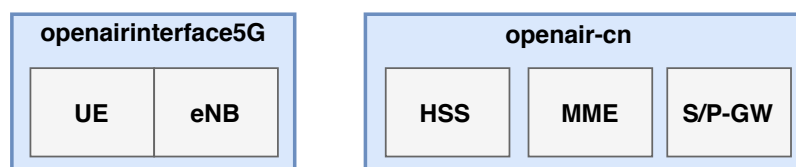


Figure 2.10: Brief representation of the modules implemented by OpenAirInterface

openairinterface5g

The `openairinterface5g` provides the source code for the UE and eNB RAN. The source code is organized in the following way¹¹:

- `cmake_targets`: folder that contains the script to compile the code (simulation, emulation, and real-time platform) and the generated build output files.
- `common/utls`: OAI utilities, such as the Inter-task Interface (ITTI)
- `openair1`: source code of PHY layer.
- `openair2`: implementation of the protocol stack of layer 2, such as RLC, MAC, PDCP, RRC and X2 Application Protocol (X2AP).
- `openair3`: implementation of the S1AP, NAS and GTPv1 for eNB and UE.
- `targets`: top-level wrappers for unitary simulation for PHY channels, system-level emulation and real-time eNB and UE.

The eNB needs a configuration file with specific parameters filled while it is running. The parameters of the configuration file are divided into six main sections [37]:

- **Main parameters**: configuration of the base station identity, Tracking Area Code (TAC), Mobile Country Code (MCC), and Mobile Network Code (MNC).

⁹OpenAirInterface5G: <https://gitlab.eurecom.fr/oai/openairinterface5g>

¹⁰Openair-cn: <https://github.com/OPENAIRINTERFACE/openair-cn>

¹¹OAI5G source code: <https://gitlab.eurecom.fr/oai/openairinterface5g/blob/v0.6.1/README.txt>

- PHY parameters: configuration regarding the physical layers like frequency and band.
- Special Radio Bearer (SRB) parameters: configuration of the SRB like retransmission timer.
- MME parameters: configuration of the MME parameters like IP address.
- Network interfaces: configuration of the eNB S1 and S1-MME IP address and interface names.
- Log configuration: select the logger's level and verbosity by taking all the layers and components of the network into account.

openair-cn

The `openair-cn` repository provides the source code for the main elements of the core network: S-GW, MME, P-GW, and HSS. The S-GW and P-GW integrate only one entity, the S/P-GW.

HSS is the network element that contains the database and uses `freeDiameter`¹², an open-source protocol that implements the Diameter protocol. Moreover, it uses MySQL¹³ database to store all the information regarding the user subscriber. It needs some configurations regarding the access to the database, such as the username, password and database name. Besides, all the network subscribers need to be inserted into the database, including the IMSI, International Mobile Equipment Identity (IMEI), Mobile Station International Subscriber Directory Number (MSISDN). It is connected to MME via S6a thread.

MME also uses S6a thread like HSS. This also needs some configurations, such as maximum number of UEs and eNBs, S1AP outcome timer, the Globally Unique MME Identifier (GUMMEI) and the Tracking Area Identity (TAI) parameters, S11 and S1-MME interfaces, and logging.

S/P-GW is the combination of the elements S-GW and P-GW and do not communicate through S5/S8 interface. It uses the GTP protocol to communicate. A configuration file of this entity is also provided and is divided into S-GW and P-GW. The S-GW configuration refers to S11 and S1-U interfaces, ITTI message queue size and logging. The P-GW configuration is regarding the SGi interface, DNS address, and a pool of IP addresses available for UEs.

Despite being a very complete implementation, OAI is also a very complex system. The project presented is not well documented and organized, and code is sometimes confusing, which is hard for new users to understand or customize.

¹²`freeDiameter`: <http://www.freediameter.net/>

¹³MySQL: <https://www.mysql.com/>

2.2 5G NETWORK

5G network has been designed to provide a more flexible and scalable network technology, capable of connecting everyone and everything, everywhere. The first version of the 5G specifications system was introduced in release 15 by the 3GPP community, which includes the 5GC [38] and New Radio (NR) [39].

Initially, 3GPP community had some 5G architecture options where there were multiple possibilities for the core network architecture. The work had progressed, and only two main architectures were considered, shown in the figure 2.11.

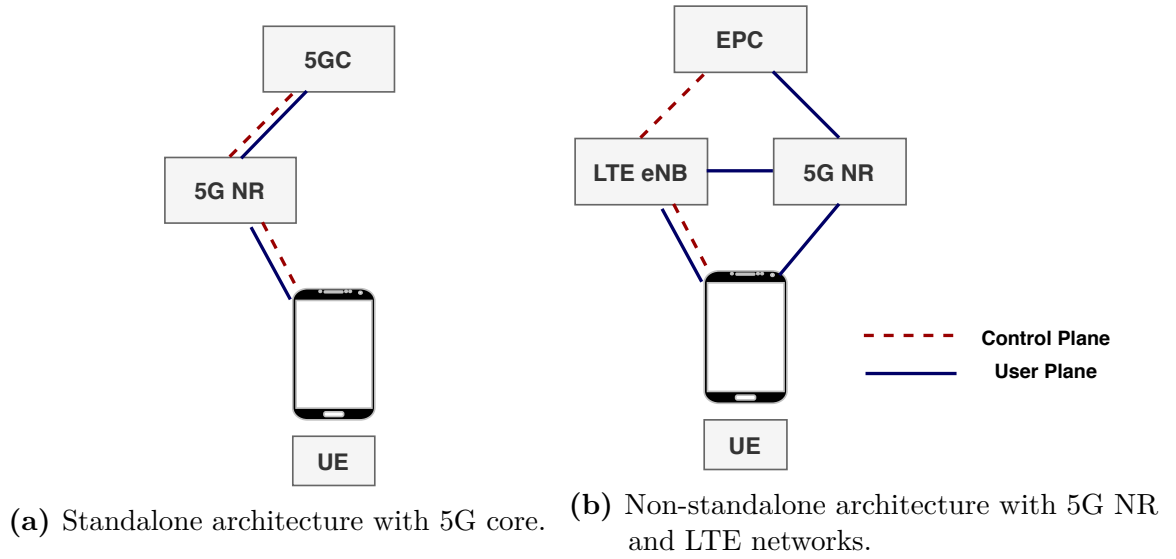


Figure 2.11: Main architecture options proposed by 3GPP for 5G evolution.

In figure 2.11a shows an architecture of the 5G NR with 5GC in Standalone (SA) mode. This architecture enables all the functionalities provided by the 5GC, like Service Based Architecture (SBA). Meanwhile, figure 2.11b shows the integration of the 5G radio systems in LTE networks, i.e., Non Standalone (NSA) mode. In this case, it uses LTE as the anchor for connection and connecting through the existing EPC. Only the user plane of the 5G radio is considered, which is then used with dual connectivity with LTE[40].

The 5G system is described to support data connectivity and services enabling deployments using new techniques such as Network Function Virtualization and Software Defined Networking. In order to allow independent scalability, evolution and flexible deployments, this new system maintains the principle of separating the Control Plane from the User Plane. Another principle for the design of 5G is to minimize the dependencies between the Access Network (AN) and the Core Network (CN) through the converged CN with a common interface (AN - CN) which integrates different access types such as 3GPP and non-3GPP access. 5G presents two options for the architecture:

(i) the first one is to be similar to LTE network namely reference points; (ii) the second consists of a service-based interface (SBI), which is used in the control plane, where it only exists one interface in network functions (NF), and these interfaces connect via bus.

2.2.1 Core network

The following sections describe the main functionalities of the 5G core network functions [38], [40], [41]. Figure 2.12 depicts the non-roaming of the 5G network architecture where service-based interfaces are used into the control plane.

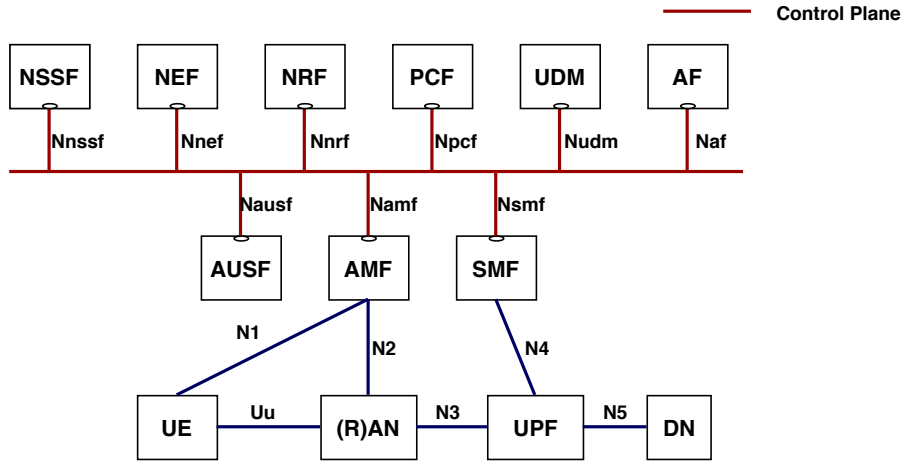


Figure 2.12: 5G Architecture [38]

2.2.1.1 Access and Mobility Management Function (AMF)

AMF is responsible for termination of RAN control plane interface (N2) and NAS (N1), NAS messages ciphering and integrity protection, lawful intercept, registration management, connection, reachability and mobility, and Session Management Function (SMF) selection. It provides transport for session management (SM) messages between UE and SMF, transparent proxy for routing SM messages, access authentication and authorization, security anchor function (SEAF), security context management (SCM), and EPS bearer ID allocation for interworking with EPS. In addition, the AMF supports functions associated with non-3GPP ANs like authentication of UEs connected over Non-3GPP InterWorking Function (N3IWF).

2.2.1.2 Session Management Function (SMF)

The SMF provides SM (session establishment, modification, release), allocation and management of the IP address to UEs, selection and control of User Plane Functions (UPF), termination of SM parts of NAS messages, control and coordination of charging

data collection at the UPF, DL data notification, and control part of policy enforcement and QoS.

2.2.1.3 User Plane Functions (UPF)

The UPF performs packet routing and forwarding, packet inspection, QoS handling for user plane, traffic usage reporting, acts as external Packet Data Unit (PDU) session point of interconnect to data network, and is an anchor point intra-/inter-radio access technology (RAT) mobility.

2.2.1.4 Policy Control Function (PCF)

The PCF applies unified policy frameworks to govern network behaviour. Further, it is responsible for accessing subscription information for policy decisions in a Unified Data Repository (UDR) and provides policy rules to Control Plane functions.

2.2.1.5 Network Exposure Function (NEF)

The NEF securely exposes capabilities and events provided by NFs, like AFs and edge computing. Furthermore, it translates internal and/or external information, and handles masking of network and user sensitive information towards external AFs according to the network policy.

2.2.1.6 Network Repository Function (NRF)

The NRF supports service discovery and maintains the NF profile of available NF instances and their supported services.

2.2.1.7 Unified Data Management (UDM)

The UDM is responsible for the generation of 3GPP AKA authentication credentials, user identification handling, access authorization based on subscription data and subscription management.

2.2.1.8 Authentication Server Function (AUSF)

The AUSF is like an authentication server, given that it authenticates the UE.

2.2.1.9 Application Function (AF)

The AF supports application influences on traffic routing, NEF access, and interaction with policy framework for policy control.

2.2.1.10 Unified Data Repository (UDR)

The UDR is a common database for all types of the data structure standardized such as subscription data, policy data, structured data for exposure and application data.

2.2.1.11 Network Slice Selection Function (NSSF)

The NSSF is responsible for selecting the Network Slice instances to serve the UE, determining the allowed and configured Network Slice Selection Assistance Information (NSSAI), and determining the AMF set used to serve the UE.

As mentioned before, 5G system has a completely separate control and user planes. The control plane presents more network elements than the user plane, wherein the UPF is the only network element responsible for processing all the user plane traffic. Table 2.1 presents the comparison between network element of 4G and 5G system.

4G	5G
HSS	UDM
MME	AMF
MME, S-GW and P-GW (control plane)	SMF
P-GW and S-GW (user plane)	UPF
PCRF	PCR

Table 2.1: Comparison of the network elements between 4G and 5G system.

2.3 BROKERS

Broker, also known as message broker or data bus, is a software that enables systems to communicate with each other and exchange messages. It allows independent services to communicate with others in different languages or implemented on various platforms. One or more producers send those messages to the broker. The broker acts as an intermediary between systems, so it will not process the message; it will route the message for one or more message queues. The message queue will store and order all the received messages until the consuming application can process them, namely consumers. All the received messages are stored in a message queue for the order that they have arrived. After consumers process the message, the message is discarded from the message queue [42].

The next subsection presents three open-source software solutions of message brokers.

2.3.1 Software Solutions

2.3.1.1 RabbitMQ

RabbitMQ is a messaging broker, that acts as an intermediary for messaging. It enables software applications to connect and exchange data between them. Besides, it supports a variety of messaging protocols like Advanced Message Queuing Protocol (AMQP)¹⁴

¹⁴AMQP: <https://www.amqp.org/>

and Message Queue Telemetry Transport (MQTT). There are client implementations for almost any programming language, which allows to use it in different platforms [43].

AMQP aims to create a complete functionality between clients and servers. For that, it defines a set of components and standard rules for connecting these. The main types of components are[44]:

- exchange - the entity that receives a message from the publisher and routes it to a message queue defined previously
- message queue - the entity that stores messages until they are consumed by a client
- binding - connects an exchange with a queue using binding key

As illustrated in figure 2.13, the producer never sends any message directly to the message queue. For this reason, the producer generates and sends the message to the exchange. The exchange is an entity that receives messages from the producers and

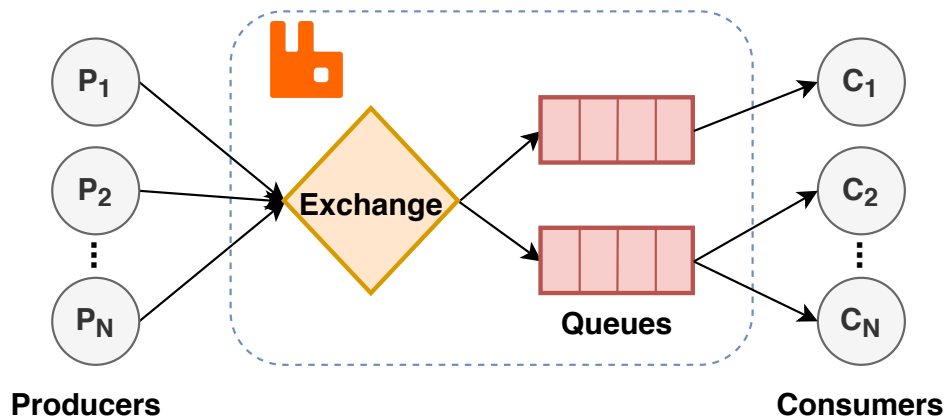


Figure 2.13: Simplified RabbitMQ architecture.

forwards them to the queues. When the exchange receives the message, it should know what to do with it, for example, should it append the message to zero, one or more queues? There is a need to create rules that are defined by exchange type (direct, fanout, topic). After the exchange type is created, it is necessary to inform the exchange entity of which queue it should send the message. This relationship is named binding. Consumers have a persistent connection to the broker, providing the information of which queues they are subscribed to. The broker forwards the message to the consumer [45].

RabbitMQ also offers support for Remote Procedure Call (RPC) communication. In general, the process consists on sending a message and waiting for a response. This way, the publisher entity starts with the creation of the private callback queue. When a request occurs, this entity sends a message with two more informations: the private callback queue created to receive the answer, the field `reply_to`, and the correlation id,

which is an identifier value for each request created. The message sent by the publisher is set to a defined queue. The consumer is waiting to receive messages on this queue. After the consumer receives the message it will do its job and send a response message using the queue from `reply_to` parameter. In the end, when the publisher receives the answer, it will verify the correlation id property. If it matches it will return a response to the server [46]. Figure 2.14 illustrates the RPC communication using RabbitMQ.

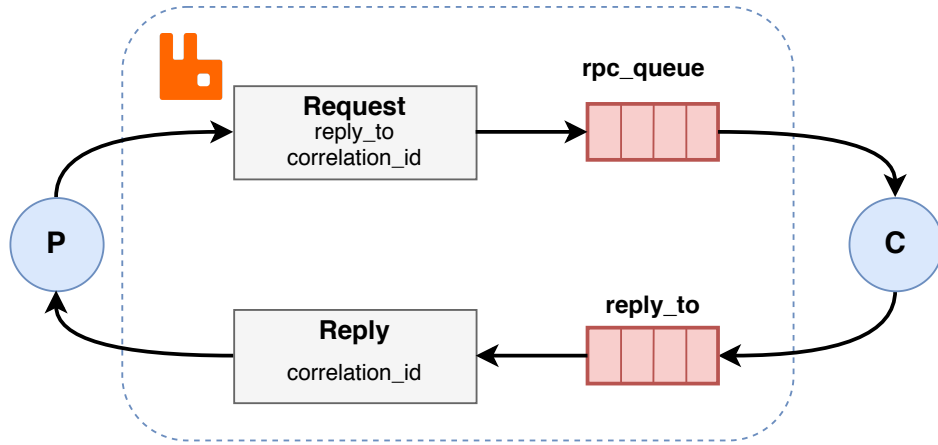


Figure 2.14: RabbitMQ RPC system.

RabbitMQ does not save messages after they are successfully consumed [47].

2.3.1.2 Eclipse Mosquitto

Eclipse Mosquitto¹⁵ is a message broker that provides client and server implementation of the MQTT¹⁶ protocol. MQTT is a light-weight publish and subscribe messaging protocol, presenting low network overhead. This project is composed by [48]:

- mosquitto server
- mosquitto_pub and mosquitto_sub client utilities
- a MQTT client library

The open-source client implementations of MQTT are provided by the Eclipse Paho¹⁷ in different languages of programming.

The MQTT protocol is comprised by two components: clients and the broker as shown in figure 2.15. The client could be a publisher or a subscriber wherein a publisher publishes messages to a specific topic and a subscriber subscribes a particular topic. The broker has the responsibility to control all the received messages and forwarding them to the subscribers that are listening to the corresponding topic [49].

While sending a message to the broker using a specific topic and the subscriber is not ready to subscribe to it, the message is not stored at the broker until the subscriber

¹⁵Eclipse Mosquitto: <https://mosquitto.org/>

¹⁶MQTT: <http://mqtt.org/>

¹⁷MQTT Client: <https://www.eclipse.org/paho/>

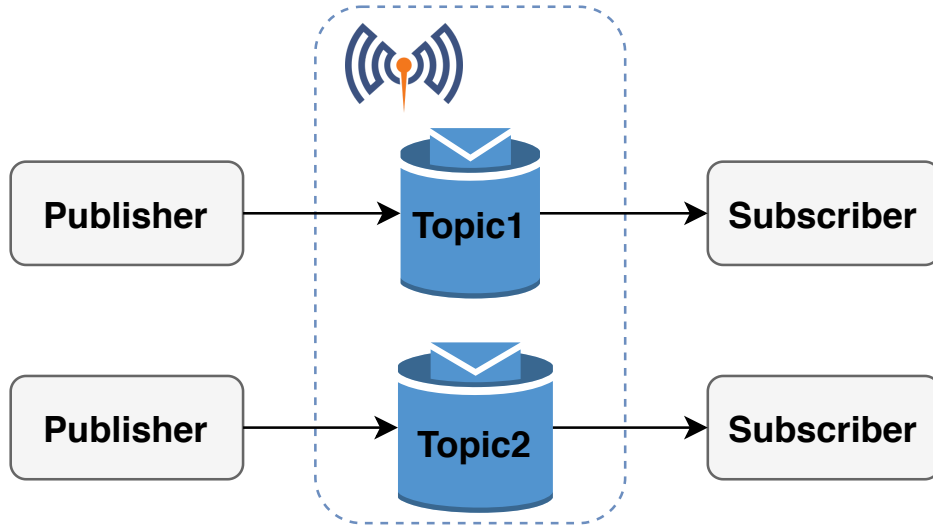


Figure 2.15: Simplified MQTT architecture using Mosquitto broker.

is prepared. This way, MQTT provides three QoS levels to send the message: QoS0 - at most once, QoS1 - at least one time, QoS2 - exactly once [50].

2.3.1.3 *Kafka*

Apache Kafka¹⁸ is a distributed streaming platform developed by LinkedIn and donated to the Apache Software Foundation. It was designed to provide high throughput in a publish and subscribe model [47].

Kafka has a component known as Kafka Cluster that stores data streams, which are sequences of messages produced by applications and sequentially consumed by other applications. Kafka cluster supply publishes and subscribes service, as illustrated in figure 2.16. Producers send messages into Kafka, and consumers read those messages from Kafka. All exchanged messages are saved in Kafka servers, namely brokers, and disposed in topics. The topic consists of an agglomeration of messages, called log. This way, when a message is sent for this topic, it is appended at the end of the log.

¹⁸Kafka: <https://kafka.apache.org/>

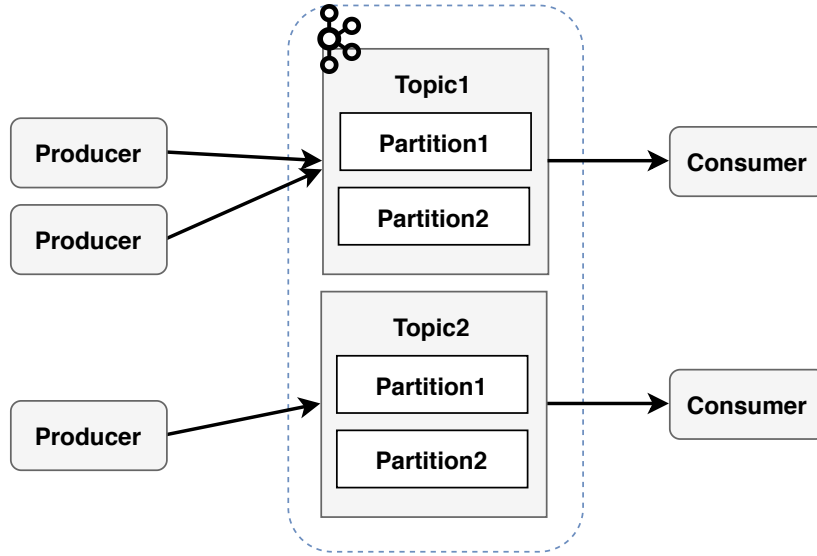


Figure 2.16: Simplified Kafka architecture.

Furthermore, topics are divided into partitions. When a producer sends a message to a specific topic, it has to define the partition related to this message. However, the producer client could not use partitions to send messages. By default, each message sent is assigned with a key, and messages with the same key will be sent to the same partition. Kafka only guarantees the order of received messages at the same partition in the same topic. Topics could have zero, one, or more consumers that subscribe to a message [47], [51].

Kafka is used due to this long term storage messages. It saves all received messages on its disk. If Kafka does not have any configuration about the long term storage and consumers, consumers can reply to the message when needed [47].

2.4 SUMMARY

The first section, section 2.1, of this chapter starts by addressing the evolution of the network to 4G. Subsections 2.1.1, 2.1.2, and 2.1.3 have descriptions of the 4G network components. Furthermore, the protocol stack as well as connections procedures are explained, in subsections 2.1.4 and 2.1.5, respectively. Section 2.1.6 is presented some software platforms that integrate the 4G networks.

The evolution of the 5G network and its architecture are described in section 2.2.

Finally, section 2.3 describes some software solution for the broker implementation.

Architecture and Specifications

To reach the final architecture, there is a need to make choices in terms of which LTE software platform fits the requirements, what type of changes are needed, and how to integrate the message broker into the existing architecture.

This chapter starts by identifying the problem statement in section 3.1. Then, section 3.2 presents the requirements needed. The first step is to choose the LTE platform. Then, some software platforms are analyzed and we select the one that fits our purpose. In section 3.2.2, the architecture of the platform chosen is displayed. In section 3.2.3, some requirements of the message brokers are introduced. The next section 3.3 describes the proposed solution. Finally, section 3.4 gives an overview of broker integration.

3.1 PROBLEM STATEMENT

Data traffic demanding high data rates is rising. The network operators have to upgrade and expand their networks to answer these demands continuously. However, all these exchanges on the network are expensive, and the revenue is lower. In an attempt to solve this problem, 5G network was developed and proposed by the 3GPP community offering more flexibility and scalability. 5G network keeps the same idea as 4G to maintain both user and control plane split. Furthermore, the main difference consists of a new core architecture wherein there are many network functions for the control plane and only one for the user plane. Despite that, one of the architecture consists in all the network functions of the control plane do not connect directly, they only have one interface using a bus to communicate, following a Service-based interface approach as observed in web-based technologies. In fact, this increasingly absorption of web-based aspects by mobile network technologies is what drives cloud-native enablements [52]. The purpose of this thesis is to further explore this capability and realize the communication

bus between network core elements via a message broker. Brokers are commonly used in cloud-based environments, allowing the same messages to be sent to different destinations, even when operating at different time frames (e.g., storing an event in a database versus consuming to act upon it). So, this thesis's work will evaluate the effect of using message brokers to make the signaling routing between the core entities, removing the dedicated interfaces of 4G.

3.2 REQUIREMENTS AND SPECIFICATIONS

This thesis focuses on the study of integration of 5G-like aspects, such as the use of SBI between network core functions, but in a 4G environment. For this reason, there is a need to choose a platform that implements the concepts of the 4G network. Furthermore, we need to select some open-source solutions for message brokers that could be implemented and integrated with the chosen LTE platform.

3.2.1 Selection of LTE implementation

Some of the LTE implementation platforms were described in section 2.1.6, now we need to choose one of them. But, before selecting it, there are two requirements that we need to take into account:

- full implementation of the LTE network (modules, protocol stack, and authentication)
- UE emulator

These requirements are fundamental for this thesis because the aim is to evaluate the buses on the control plane during the UE's attachment procedure, when it tries to connect to the network.

To decide which framework to use in this thesis, we need to compare the platforms described previously in chapter 2. Table 3.1 offers an overview of the main requirements defined.

OpenEPC and Amarisoft LTE are not good choices because they do not offer free services. For the development of this work, it is essential to have full LTE implementation and an UE emulator. Due to this reason, except srsLTE and OAI, the other platforms are not viable for the necessary implementation.

OAI and srsLTE are platforms with many similarities in features, and both provide development of the main components of the 4G networks. However, while OAI includes a full implementation of EPC, eNB, and UE, srsLTE consists of a light-weight EPC implementation. When this thesis work started, srsLTE did not provide the emulation UE option, and this option only appeared in June 2019. For this reason, the decision relied on the full implementation LTE and UE emulator option, OAI.

Platform	Requirements		
	Implementation of the LTE	UE emulator	Cost
OpenLTE	No UE	No	Free
srsLTE, srsUE, srs eNB	Yes, but light-weight EPC	Yes, June 2019	Free
OpenEPC	Only EPC	No	No Free
NextEPC	Only EPC	No	Free
GR-LTE	Only UE	No	Free
OSLD	No EPC	No	Free
Amarisoft LTE	Yes	No	No free
OpenAirInterface	Yes	Yes	Free

Table 3.1: Comparison between LTE implementations platforms.

3.2.2 OAI

OAI currently provides a standard-compliant implementation of release 8.6 with a subset of release 10 LTE for the UE, eNB, MME, HSS, S-GW, and P-GW on standard Linux-based computing equipment (Intel x86 processors). Figure 3.1 represents the network elements that are currently implemented by OAI. However, they are working in the implementation of Release 14 and, at the same time, initiating the first steps for 5G networks. The S-GW and P-GW are implemented in the same module, namely S/P-GW, eliminating the S5/S8 interface.

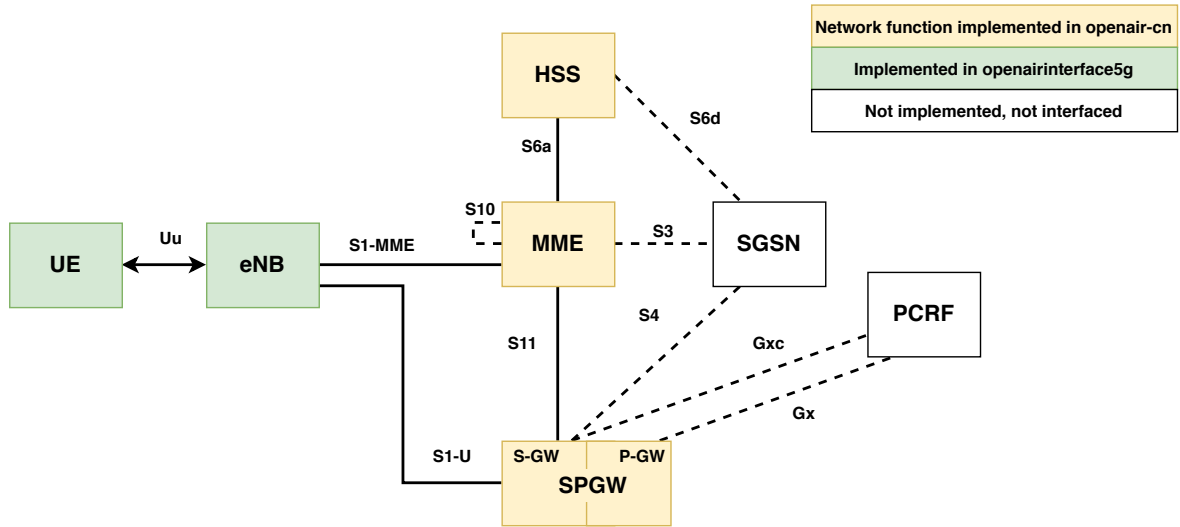


Figure 3.1: LTE network elements implemented by OpenAirInterface [53].

The emulation is one of the requirements of this thesis, and this platform provides

it. To emulate a user connected to the network, we need to use the OAISIM program. By default, only one user is configured; however, multiple UEs can be configured as well as multiple eNBs.

The first step is to set the Public Land Mobile Network (PLMN) list that UE will be able to recognize [54]. Each PLMN is composed of MNC and MCC.

Secondly, all the information regarding a particular user has to be filled. This information consists of:

- Home Public Land Mobile Network (HPLMN): network operator of UE.
- IMEI: unique number to identify devices.
- Mobile Subscription Identification Number (MSIN): subscribe identification of the UE. The IMSI is the concatenation of the HPLMN and MSIN.
- MSISDN: number used to identify the device number internationally.

Finally, the USIM parameters and permanent data of UEs have to be defined. For that it is necessary to generate the binary output files (files hidden in the filesystem):

- .ue.nvramX
- .ue_emm.nvramX
- .usim.nvramX

X is the index of the UE. By default, these binary files are generated during the build script. However, if the UE's exchanges are after the build script, there is the need to replace them with a specific configuration file. The full UE configuration must be in the database; otherwise, the UE could not establish connection to the network.

During the connection between the emulator and the OAI core, virtual interfaces were created. These were assigned according to the number of users configured to connect to the network. However, during the course of this dissertation, it was verified that the existing emulator was designed with simple signaling conformance testing in mind and not for performance testing. As a result, some stability problems while connecting to the OAI core were experienced. These problems occur because the emulator is not stable when more than three users are configured to run. Such event leads to the emulator disconnecting without apparent reason and performing a different behavior in each execution.

3.2.3 Brokers

One of the requirements is to develop code in the same language of OAI, C code, as all the changes that we are going to make are the OAI platform. For this reason, we need to choose brokers that provide libraries and clients in C code.

The brokers that we are going to integrate and evaluate throughout this thesis were described previously in section 2.3.1. These are the most popular and fill the requirements. We are going to integrate these brokers with different types of services.

That means, RabbitMQ using RPC service and Mosquitto and Kafka using topics. The reason why we chose different services is to evaluate if there is any impact on using them.

3.3 PROPOSED SOLUTION

Given the goals described in chapter 1, the aim is to analyze the operation of a SBI between 4G core network functions, not only approximate information exchanges to what is done in 5G, but to evaluate as well the integration with a message broker.

3.3.1 Architecture

In order to meet the proposed architecture, three steps have to be taken: study the software platform, adapt the source code of the message brokers, and integrate them into the OAI software platform. As previously discussed, the aim is to evolve the 4G network to the 5G's core network concepts, and for that, there is the need to know the code, structure, and workflow of the platform chosen to implement the brokers correctly.

Starting by analyzing the OAI platform, and know its architecture, the integration of the message broker would be between the modules eNB and MME, and between all the components in the core network, as illustrated in figure 3.2.

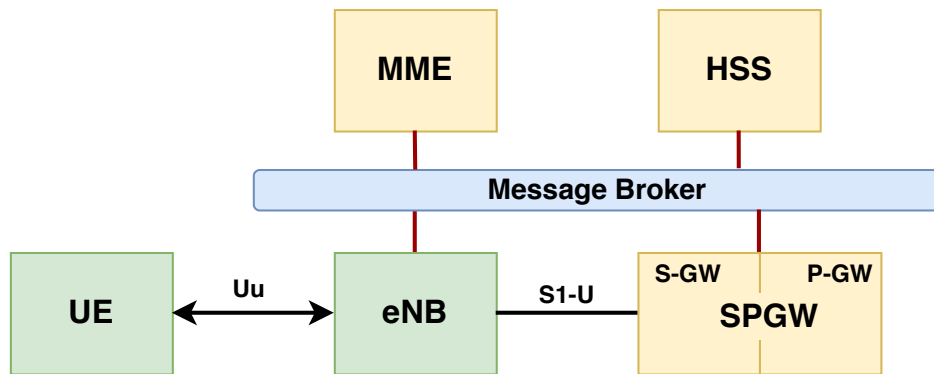


Figure 3.2: Illustration of the integration of the broker between the core elements and eNB.

When we were analyzing in more detail the code as well as the structure of all the components, we noticed that the provided code is complex. To integrate the message broker between all the components, many changes would have to be made, and in some cases, a reformulation of almost entire modules. The complexity in eNB code is less than the EPC, but it still implies a lot of changes. In EPC, the HSS module, for example, uses specific functions of the open source protocol free diameter to communicate, which increases the complexity. Due to this, we chose to integrate the message broker only between MME and S/P-GW, as presented in figure 3.3.

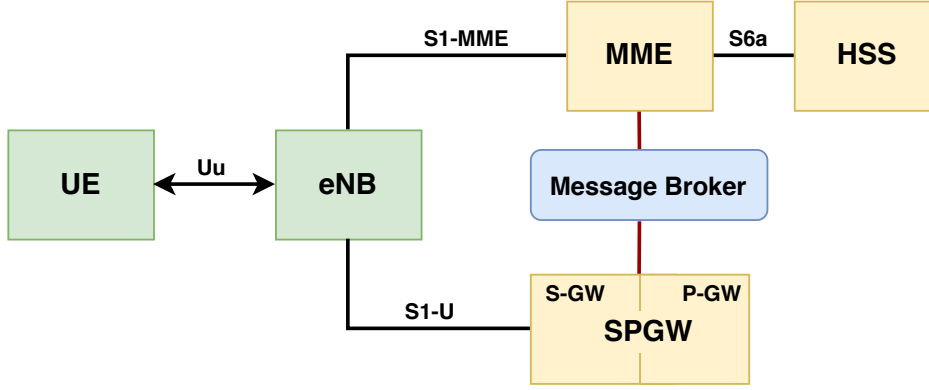


Figure 3.3: Proposed solution architecture.

With the integration only between these two modules, we expected to have an overview of how using broker-enabled bus on the control plane affects the network performance. To evaluate the presence of the bus in the control plane, we are going to integrate three message brokers, and study which message broker is the most suitable for control signaling routing.

3.4 INTRODUCING BROKERS

To have an approximation of the 5GC architecture presented in section 2.2.1, the evolution of the EPC is necessary. For that, it is required to introduce web-based communication capabilities between the network core elements and, in this case, to add the feature of a message broker into the EPC defined by 3GPP and described in section 2.1.

The network function HSS does not have changes, therefore, it operates just as specified by 3GPP. The network functions MME and S/P-GW need some modifications; however, the MME only has to be modified to communicate with S/P-GW. For these network functions, new methods needed to be implemented so that they are able to send and receive the adequate information, throughout the broker. This information includes:

- The buffer of the sent message.
- Size of the sent message.

Both network functions are sending and receiving messages through the broker. When a message is received, the entity will decipher the message and check if this message is of its responsibility to process.

When the program starts, both entities (MME and S/P-GW) exchange some information with the broker entity. This information is related to the IP address and port that communicate with the broker, the topic or queue that they are subscribed and, in some cases, the ID of the client.

3.4.1 Attachment Procedure

The attachment procedure is similar to LTE; however, the difference consists of one more entity between the MME and S/P-GW, wherein the messages are exchanged through publishing and subscribing methods. Figure 3.4 describes the attachment procedure with the introduction of the broker. These procedure steps are the same as for LTE, presented in section 2.1.5.3, up until step 6. So, from step 6, the process changes and it is described below.

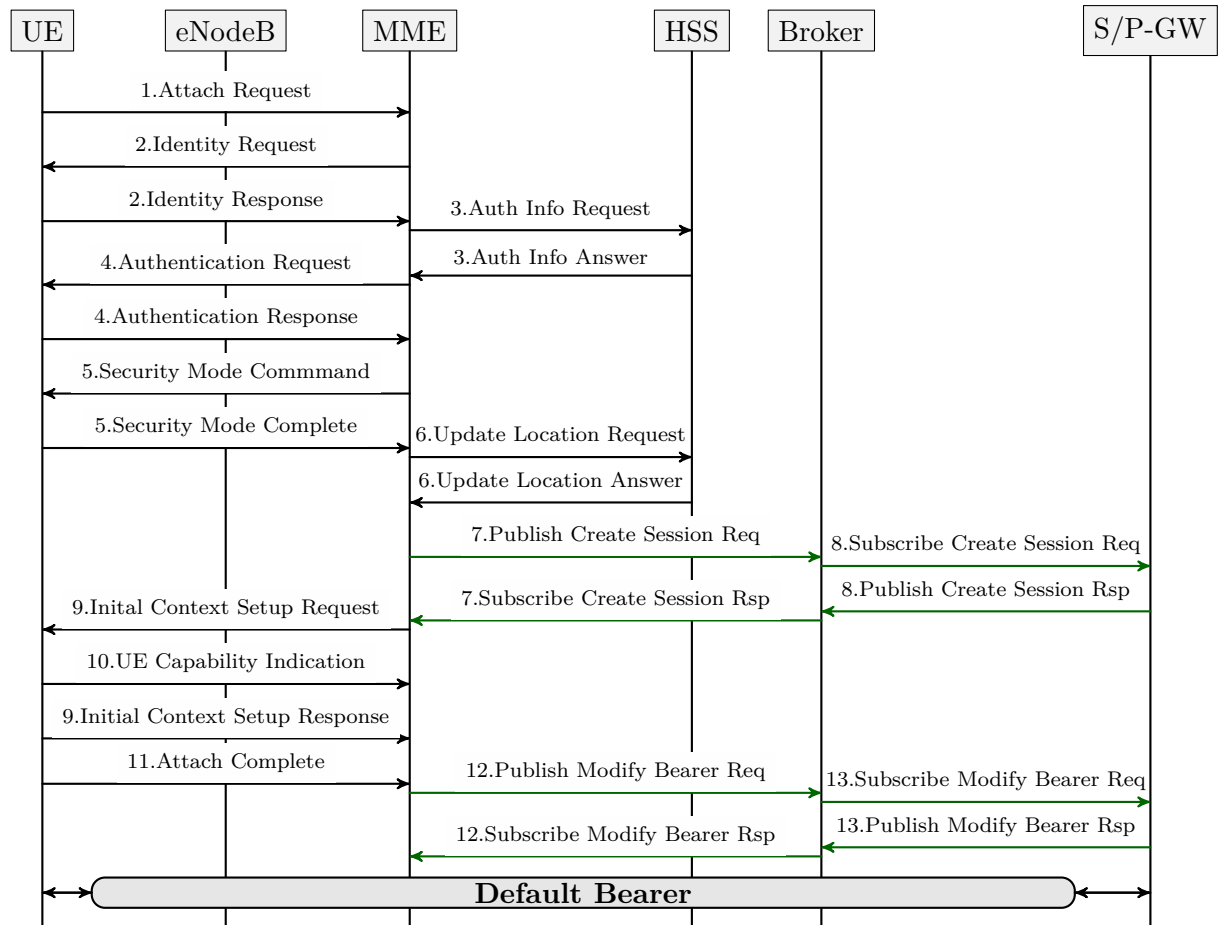


Figure 3.4: Attachment procedure using a broker.

7. The MME entity requests a Create Session to the S/P-GW, through the broker. For that, it will send a publish to the broker, identifying the topic. This topic refers to the topic that the subscriber of the other entity is listening to. After sending the message, it will be waiting to receive a response from the broker.
8. When the broker receives the publish, it will forward the message to the proper topic in the subscriber of the S/P-GW entity. The message arrives, and it will be decoded. After that, this entity sends a response message back (Create Session) through the broker, publishing it in another topic.

12. After the MME updates the S1 Bearer, it will send a publish message, Modify Bearer Request, via the broker to inform it of the exchanges. Then, it will wait to receive an acknowledge message.
13. The broker entity receives another publish from MME, in this case, a Modify Bearer message, and forwards it to the subscribe entity in the S/P-GW. It will decode and send an acknowledge by publishing a Modify Bearer Response.

3.5 EXECUTION PROCEDURE

In addition to the attachment procedure, the execution will also be considered, i.e., the user connection time until it is disconnected from the network. The procedure for LTE and using message brokers will be described below.

3.5.1 Execution procedure for LTE

Considering the LTE attachment procedure, the execution procedure is similar but with one more message being exchanged. Figure 3.5 describes the execution procedure.

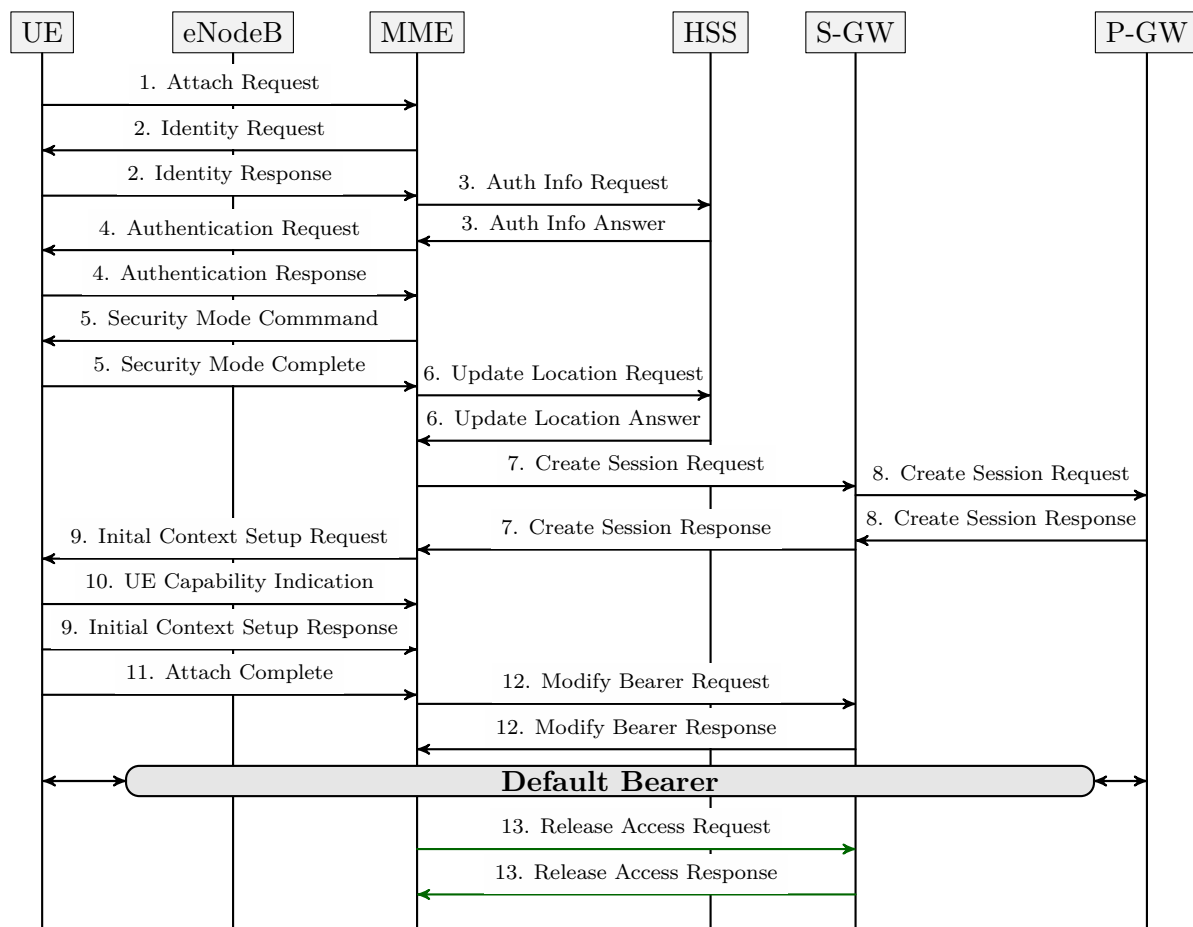


Figure 3.5: EPS execution procedure

The execution steps are the same as LTE's, presented in section 2.1.5.3, up until 12. The stages introduced from that point are described below.

13. When a user is disconnected from the network the eNB sends a shutdown message to MME. After receiving this information, MME requests the S-GW for the release of resources associated with the eNB, sending a Release Access Request. It will inform that no downlink traffic can be delivered. After S-GW receives this message it will send an acknowledge to MME through Release Access Response.

3.5.2 Execution procedure for message brokers

The execution procedure is similar to the LTE, but in this case, there is one more entity. Figure 3.6 illustrates the execution process using a broker. The procedure's steps are the same as for attachment using the broker, described in section 3.4.1, up until step 13. From step 13, the process is presented bellow.

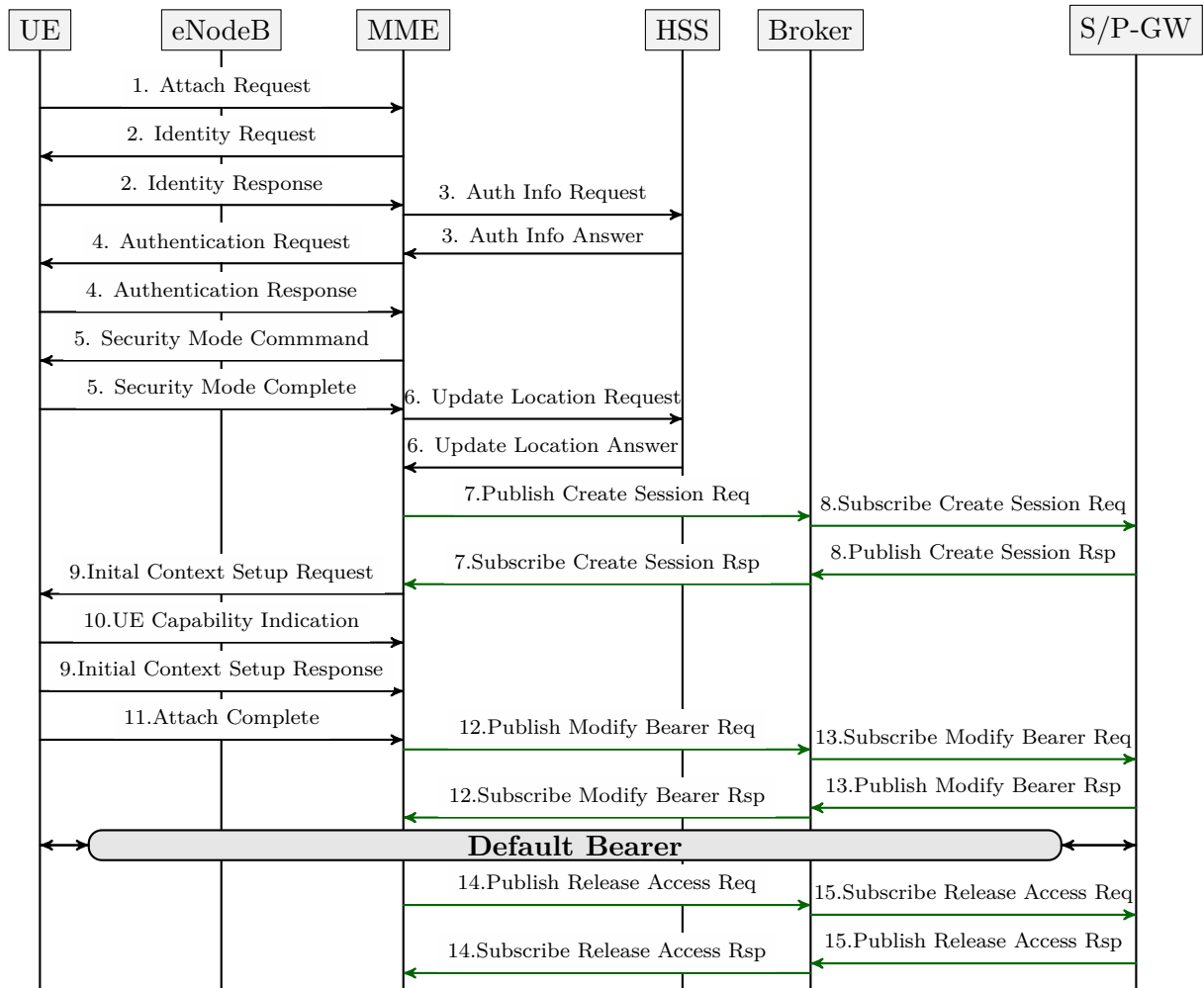


Figure 3.6: Execution procedure using a broker.

14. After a shutdown message is received from eNB, MME requests a Release Access to the S/P-GW, through the broker. This way, it will send a publish message to the broker, to inform that no more downlink traffic can be delivered. Then, it will wait to receive an acknowledge message.
15. The broker entity receives one more publish from MME, and forwards it to the subscribe entity in the S/P-GW. It will decode and send an acknowledge by publishing a Release Access Response.

3.6 SUMMARY

This chapter addressed the essential steps of the development of this thesis: problem identification, necessary requirements description, and the proposed solution and its architecture.

In the first section, section 3.1, it was enunciated that the 4G network is not scalable and causes significant investments on behalf of the network operators to face the user's demand continuously. For this reason, the 3GPP community proposed another architecture that is expected to be more scalable and flexible, given the lower costs of network operators. However, this new architecture is also a first step towards cloud-native mobile networks, which allows the consideration of introducing even newer concepts. So, it is necessary to evaluate their impact.

Section 3.2 has the aim to introduce the requirements needed for selecting the platform and the message brokers. Subsection 3.2.1 defines the two requirements that the LTE platform needs to fulfill to reach the goals: full implementation of the LTE components, protocol stack and authentication. It also provides the tool to emulate UE. After comparing some platforms, the chosen one was OAI because it offers all the requirements needed and does not have any cost associated. Subsequently, in subsection 3.2.2, it was provided the OAI architecture and the tool to emulate the UE is also described. Lastly, on subsection 3.2.3, it was defined the broker's requirements to implement with the chosen platform: client source code in C and installation of the proper library.

After having the details and requirements needed, section 3.3 provides the proposed solution, which is to integrate the message broker between the modules of the control plane. Subsection 3.3.1 presents two architectures: the first one for a total implementation of the message broker in the control plane and the other, which is the proposed solution.

Section 3.4 it was presented the signaling for UE attachment using a message broker. Finally, in section 3.5 it was described the execution procedure for LTE and using a message broker.

Implementation

This chapter presents a detailed implementation of the proposed solution. It starts by presenting the characteristics of the software used in the virtual machines, in section 4.1. Sections 4.2 and 4.3 describe the main steps to configure the principal components of the network provided by OAI. Section 4.4 details the steps of the implementation of the different brokers used. Finally, section 4.5 presents the procedure to fetch the data from the captures.

4.1 OVERVIEW

Taking into account OAI's architecture, there is the need to define the proper resources as well as the distribution modules. Figure 4.1 illustrates the implementation of each module of the network developed by OAI and the proposed solution. All these modules

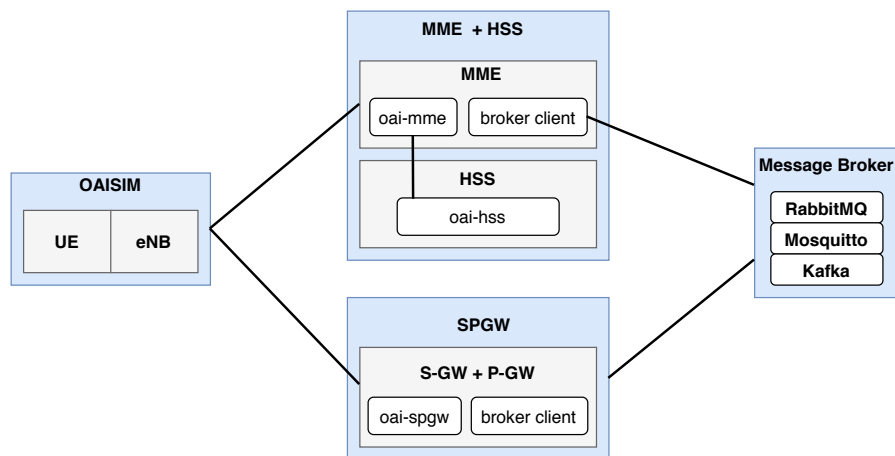


Figure 4.1: Architecture implementation in Openstack cloud environment.

were created in a cloud environment, Openstack¹.

4.1.1 Virtual Machine Specifications

Each module represented in figure 4.1 describes a VM. In the table 4.1 the resources associated to each VM like Operating System (OS) and kernel version are presented.

VM	#CPUs	RAM(GB)	OS	Kernel
OAISIM	4	8	Ubuntu 16.04 LTS	4.4.0-174-lowlatency
HSS+MME	4	8	Ubuntu 16.04 LTS	4.4.0-173-generic
S/P-GW	1	4	Ubuntu 16.04 LTS	4.4.0-173-lowlatency
Broker(s)	4	8	Ubuntu 18.04 LTS	4.15.0-91-generic

Table 4.1: Resources used by each VM.

4.2 RADIO ACCESS NETWORK (RAN)

The eNB and UE form the RAN, and the program OAISIM manages both. With OAISIM, it is possible to emulate several eNB and UE in the same process. The following subsections present their implementation.

4.2.1 Evolved NodeB (eNB)

For the eNB implementation, it is necessary to use the source code of the project `openairinterface5g`. It applies a 3GPP specification compliant eNB deploying it through simulation and emulation programs.

4.2.1.1 Hardware Setup

First, the Ubuntu 16.04 LTS with the low-latency kernel was installed. Then, all power management in the Basic Input/Output System (BIOS) (sleep states and c-states) and CPU frequency scaling were disabled. Also, hyperthreading in the BIOS should be disabled. The kernel also needs c-states and p-states to be turned off, and the governor flag is set to performance [55]. All these configurations allow that CPU to be at its maximum speed all the time, reducing the response time.

4.2.1.2 Software Setup

For the implementation of the OAISIM, the tag v0.6.1 of the `openairinterface5g` was used. First, we need to clone the git repository to the OAISIM machine[56]. Then, we need to install the necessary dependencies and compile the code through an automated script. Also, with this script the OAIUE and OAIeNB were installed.

¹<https://www.openstack.org/>

Furthermore, we need to configure it by using one of the configuration files provided. In our case the configuration file used is `enb.band7.generic.oaisim.local_mme.conf` and the most relevant parameters are described in table 4.2. The configuration file also needs to be filled with the MME IP address and network interfaces for S1-MME and S1-U interfaces. To use multiples machines with the program OAISIM, it is only necessary to increase the parameter `eNB_ID` in the configuration file.

Parameter	Value
<code>eNB_ID</code>	0xe00
Tracking Area Code(TAC)	1
Mobile Country Code(MCC)	208
Mobile Netwok Code(MNC)	93

Table 4.2: OAISIM file configuration.

4.2.2 UE Setup

The first steps for configuring UEs are modifying the UE's configuration file and compiling it with the proper command and to the appropriate folder.

The OAISIM program allows the emulation of multiples UEs. However, when more than three UEs are configured to run simultaneously, the emulator is not stable, i.e., when a fixed number of users is configured for multiple runs, the number of connected users are not the same. For this reason, we chose to configure a maximum of three users.

To configure multiple users we need to use the file `openairinterface5g/openair3/NAS/TOOLS/ue_eurecom_test_sfr.conf` and configure the same parameters to the PLMN that were configured in the MME and OAISIM configuration files. After the parameters are configured, we also need to generate the binary output files.

4.2.3 Run eNB and UE

To run the oaisim program we need to use the following command[56]:

```
$ sudo -E ./run_enb_ue_virt_s1
```

By default, this command only allows to emulate one UE. To change it, we need to modify the source code of the `run_unb_ue_virt_s1` file and change the parameter `-u`, which refers to the number of users, for the correct quantity of users that we want.

Now with all configured, the oaisim is ready to run. However, the components of the core network need to be configured and initialize their processes before the oaisim.

4.3 CORE NETWORK

The Core Network provides the main components of the EPC. To implement them, the tag v0.5 of the `openair-cn` was used. The implementation is described in the following subsections.

4.3.1 HSS + MME

HSS + MME VM implements two components: HSS and MME, and both are connected to the localhost interface. The first step is to clone the repository and run the adequate automated scripts to install all the dependencies of the HSS and MME.

4.3.1.1 Configuring and building HSS

After installing HSS it is also necessary to do some configurations. Table 4.3 presents the main parameters of the hss files that need to be changed. Then, the database `oai_db`, provided by OAI, was inserted into the HSS database, and most parameters were filled allowing the system to work as expected.

hss.conf		hss_fd.conf	
MYSQL_server	"127.0.0.1"	Identity	"hss.openair4G.eur"
MYSQL_user	"root"	Realm	"openair4G.eur"
MYSQL_pass	"*****"		
MYSQL_db	"oai_db"		

Table 4.3: Main configuration of the HSS files.

First, in the table `mmeidentity` the local hostname of MME and its realm were inserted: `openair4g.eur`. Second, in the table `user` it was inserted information about users subscribers. Table 4.4 is an example of the most important parameters that need to be configured. Finally, in table `pdn` all the IMSI configured previously were inserted, allowing them to connect to the APN

	IMSI	MSISDN	IMEI
UE#0	208930100001100	33638030000	35609304079200
UE#1	208930100001101	33638030001	35609304079201

Table 4.4: Example of user subscribers.

4.3.1.2 Configuring and building MME

Beyond the typical configuration of network interfaces, PLMN and GUMMEI parameters are also required. Also, it is in this file where the maximum number of UEs and eNBs connected are configured. The main configurations are presented in table 4.5.

GUMMEI_list	MNC	93
	MCC	208
TAI_list	MNC	93
	MCC	208
MAXUE		16
MAXENB		4

Table 4.5: MME file configuration.

4.3.2 S/P-GW

S/P-GW VM implements the S/P-GW entity. As mentioned before in chapter 3, the S-GW and P-GW entities are merged in the OAI project. The main configuration is presented in one file, “spgw.conf”, but it is only necessary to configure the proper IP address for its interfaces. After that, run the automated scripts to install all the dependencies necessary. After all the modules are configured, they are ready to run.

4.4 ADAPTATIONS TO HSS+MME AND S/P-GW

Previous sections have shown the configuration of all modules of the 4G network implemented by OAI. To achieve the proposed solution, the source code of both OAI MME and OAI S/P-GW had to be changed to implement publisher and subscriber methods of the different message brokers, substituting the peer-to-peer communication. For this, a new module was implemented and integrated into the source code. The module was written in C. The implemented files are different regarding the message broker, but the changes in the source code of the MME and S/P-GW are similar.

When developing the source code changes, the main idea was to have defined structures that can provide the needed information in a structured way. The aim is these structures can be used by message brokers to exchange messages.

The first step was to analyze the source code and understand how it was implemented and worked. All the code of the OAI between MME and S/P-GW work through tasks to exchange message between them. For this reason, a similar approach was followed.

This way, the first change on the source code was in function `nwGtpv2cCreateAndSendMsg` in the `NwGtpv2c.c` file, defined as shown in appendix

A, section “Changes on nwGtpv2cCreateAndSendMsg function”. It is responsible for sending the message to the other entity. For that, it fills a structure with the correct parameters of this specific message to send. A similar implementation was done for the message brokers. A new structure, `publisher_send_msg_t` (defined in appendix A, section “broker information structure”), was created and filled with the parameters needed for the other entity to know what message arrives, which message it will process, and send a response back. After that, a message and a task were defined, allowing a specific message (request/response) to be sent to the new module of the message broker.

When we started to implement the publisher/subscriber, we thought it would be the best way to initialize it. Checking other implementation in the source code, we noticed, for example, that UDP has defined a function to create the ITTI associated to the TASK and also received as argument a function that will be in a loop waiting for messages in this task. This way, we created a function in the publisher/subscriber file to initialize it, and it was called in the main function of the MME and S/P-GW.

A function, `publisher/subscriber_intertask_interface`, is passed as an argument during the initialization of the task for publisher and subscriber. This function enables the task and waits for a specific message to arrive. This message comes with particular information about the broker to links, such as port number, IP address, binding key, and exchange type. All this information was filled in a function called `init_publisher/subscriber`, in the `s11_mme_task.c` file where MME handles all messages to exchange with S/P-GW. After all information was filled, it was sent for the task defined previously. This implementation is shown in appendix A, section “Init broker connection in s11_mme_task.c” and “Init publisher”. Figure 4.2 illustrates

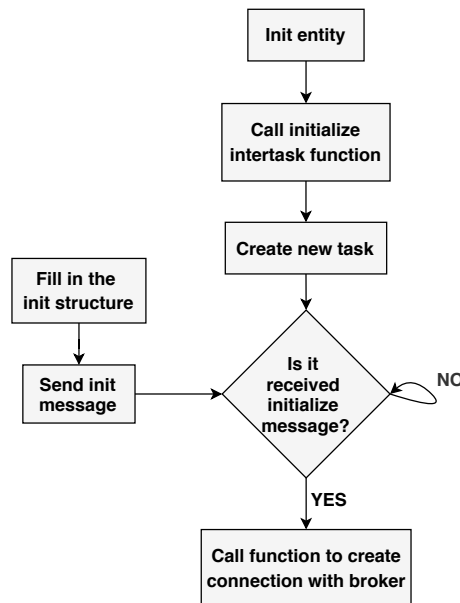


Figure 4.2: Generic initialization of the publisher and subscriber.

this process. Next subsections describes the different implementations of the message brokers.

4.4.1 RabbitMQ

For RabbitMQ the method RPC was implemented, i.e., when the publisher sends a message to the subscriber, the publisher awaits for the subscriber's response.

To implement it, it was used the source code of the project `RabbitMQ-C client`² with tag v0.9.0. The files `amqp_listen.c` and `amqp_rpc_sendstring_client.c` from the folder `examples` were adapted to be integrated into the OAI source code.

The `publisher.c` and `publisher.h` files were implemented and integrated on the OAI MME entity, and the `subscriber.c` and `subscriber.h` on the OAI S/P-GW entity. Both entities will have the behavior of publisher and subscriber. Appendix B presents a description of the source code for both files. In the next two subsubsections all the implementation steps are detailed.

4.4.1.1 Publisher/Subscriber

With all the configurations defined, the connection with the server broker is now possible to do. As explained before, the function `publisher/subscriberr_intertask_interface` is in loop waiting for the message `INIT_CONNECTION`. When it is received, a new function is executed `publisher/subscriber_connection`. This function as the role to establish a connection to the server, creates a private reply queue, publishing and subscribing.

Establishment of the connection

The first step is to create a channel that is used to connect the RabbitMQ server. The second is to create a socket with this channel and open it with the defined IP address and port. Finally, to complete the connection, the login with the broker is required, through the function `amqp_login`. This function sets the virtual host to connect to the broker, limiting the number of channels of the connection and the maximum frames to request. In the end, a channel is open, which enables the RPC.

Create queue

With the implementation of the RPC method, the publisher creates a private queue to receive the response message from another entity. For this, the function `amqp_queue_declare` is called. It declares a new queue associated with the channel created previously. On the other hand, the subscriber also creates a queue but with a

²RabbitMQ C client: <https://github.com/alanxz/rabbitmq-c>

different aim, i.e., to not send a response message like a subscriber. In this case, it was created a queue to bind with the binding key and exchange type.

Publish the message

The function `amqp_basic_publish` is used to send new messages, but it was filled in different ways according to the entity it will be implemented.

In the MME entity, the request messages are going to be exchanged through the publish method, wherein it is filled with the exchange, binding key, and the bytes of the structure to send. Nevertheless, before, we need to wait for new messages to send. For that, a loop is created associated with the task created previously, and when it receives new messages in this task provided from the function `nwGtpv2cCreateAndSendMsg`, the publish function is called.

On the other hand, in the S/P-GW entity, the publish function is called to send a response to the MME entity. However, the function is filled with the private queue, the correlation id, and the structure with the answer.

Subscribe the message

After MME publishes its message, it waits for an answer. In this case, the answer is for example, regarding to `create_session_response` message. When it receives the message, the information arrived is decoded by the structure `amqp_frame_t`. After that, it is assigned to the task of the MME that is waiting to receive the responses.

The S/P-GW module is waiting to receive request messages. When it gets the message, it has the information regarding the reply queue and correlation id to use in the publish method.

The behavior of this implementation is illustrated in figure 4.3.

4.4.2 Mosquito MQTT

To Mosquitto the source code for C client refers to the project `Paho C Client MQTT`³ and the tag v1.3.0 was used. The files `publisher.c` and `subscriber.c` from the example of the documentation⁴ were adapted and implemented. This implementation follows the topic method.

³C-Client MQTT: <https://github.com/eclipse/paho.mqtt.c>

⁴Paho: <https://www.eclipse.org/paho/files/mqtt/doc/MQTTClient/html/index.html>

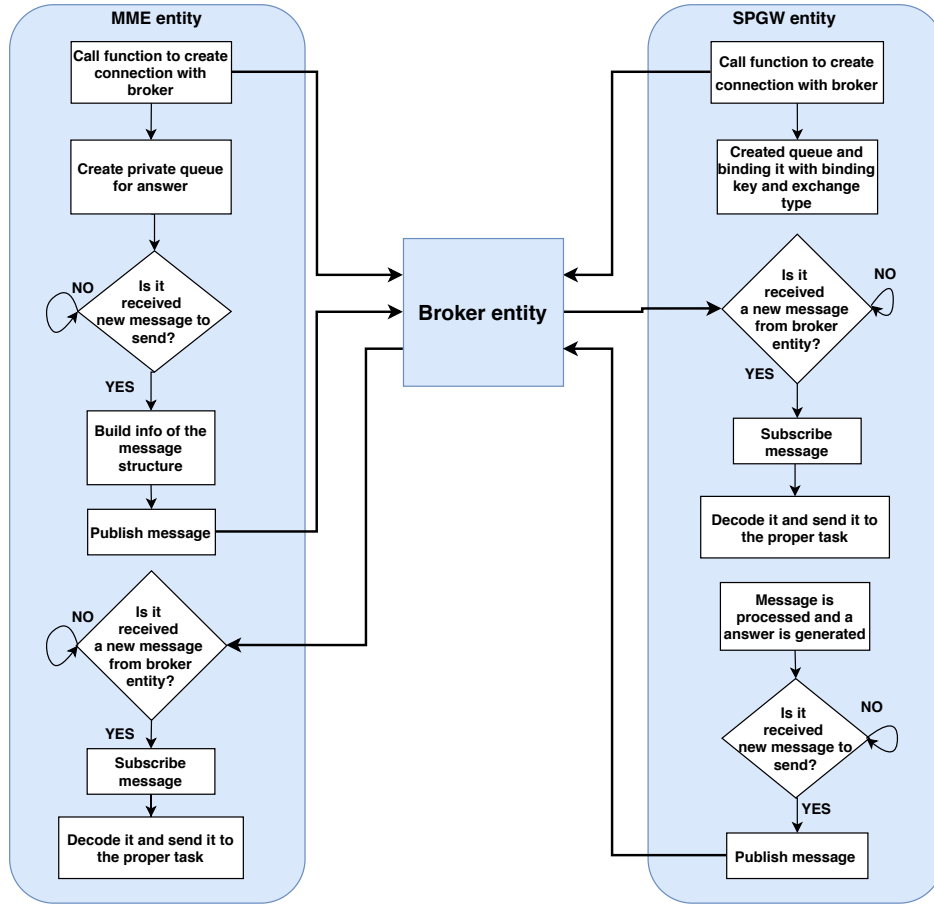


Figure 4.3: RabbitMQ flow using RPC method.

For both entities (MME and S/P-GW) the `publisher.c` and `subscriber.c` were implemented because both entities are sent and received messages. This implementation is provided in appendix C.

The structure to initialize the process has different fields. This way, the structure `publisher_init_t` and `subscriber_init_t` have the parameters IP address + port number, `clientID` and `topic`.

The first steps are the same for both publisher and subscriber methods, the creation of the client and establishing the connection with the broker.

Create the client

To create the client, we need to call `MQTTClient_create()`. This function creates a MQTT Client ready for connection to the defined server. First, we need to create a `MQTTClient` handle, and this will be populated with a valid client in case the successful return of this function. Second, this function receives as arguments the created client, address and `clientID`.

Establishment the connection

To establish the connection the previously created client is passed as argument in the function `MQTTClient_connect()`. This function tries to connect the client to the MQTT server through the specified options.

Now, with the client created and the connection established the method publisher or subscriber is called. Below they are described.

4.4.2.1 Publisher

A loop is waiting to receive a specific message and a structure that comes with it. When it receives it, the structure of MQTT related to storing the message `MQTTClient_message` is filled with the bytes and size of the received structure. After that, the publisher is available to send the message to a specific client for a defined topic, `MQTTClient_publishMessage`. However, this function also has an argument for a token. This token tells the publisher if the message was sent or not.

4.4.2.2 Subscriber

The subscriber is waiting to receive a message from the publisher in a specific topic, `MQTTClient_subscribe()`. The `MQTTClient_setCallbacks()` function sets the callback function to a particular client. One of this arguments is a pointer to a message arrived callback function. This way, this function is adapted to decode the message and send it to a specific task. After sending it, the memory allocated is freed.

The generic flow of the Mosquitto MQTT is illustrated in figure 4.4.

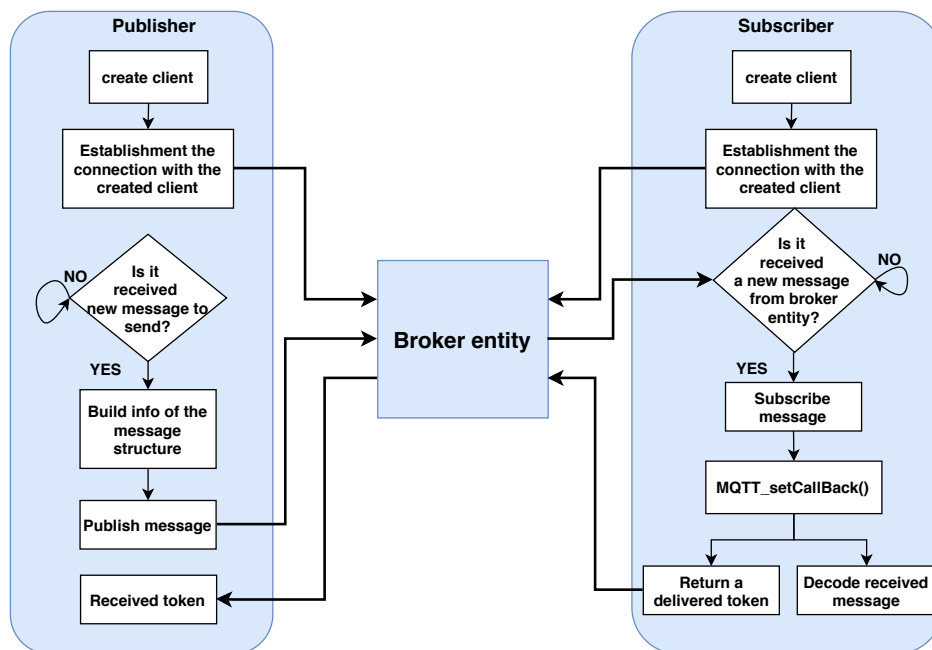


Figure 4.4: Mosquitto MQTT flow.

4.4.3 Kafka

The Kafka broker was implemented with the source code for client C from the project *Apache Kafka C*⁵. The `consumer.c` and `producer.c` from the folder `examples` were adapted and integrated into the OAI code. The files `publisher.c` and `subscriber.c` were created and implemented with the Kafka C client. The implementation of both publisher and subscriber are described in appendix D.

The next step after establishing the connection is to call the function `publisher_connection` and `subscriber_connection` for the publisher and subscriber, respectively. For the kafka C client, the first thing to do is to create the client and configure it. This way, a temporary configuration is created with the function `rd_kafka_conf_new()`. Then, this configuration is assigned to the address via `rd_kafka_conf_set()`.

4.4.3.1 Publisher

With the client configuration, a producer instance is created through `rd_kafka_new()`. When a message is received in a loop, the `rd_kafka_producev()` function is called. In this function, the instance created before is used. If this function returns an error associated with queueing full, it waits until the previous message is sent, and after a new attempt, the message is published.

4.4.3.2 Subscriber

When all the configurations are done for the client, the subscriber creates a consumer instance (`rd_kafka_new()`). With this instance created, the temporary configuration created is set to NULL. After that, all messages are redirected to the main queue, allowing the consumption of these messages (`rd_kafka_poll_set_consumer(rk)`). All received topics are associated to a subscription (`rd_kafka_topic_partition_list_new()`). Then, the subscriber is ready to subscribe the topic, and the message is decoded.

The figure 4.5 describes the flow of Apache Kafka.

⁵Kafka C client: <https://github.com/edenhill/librdkafka>

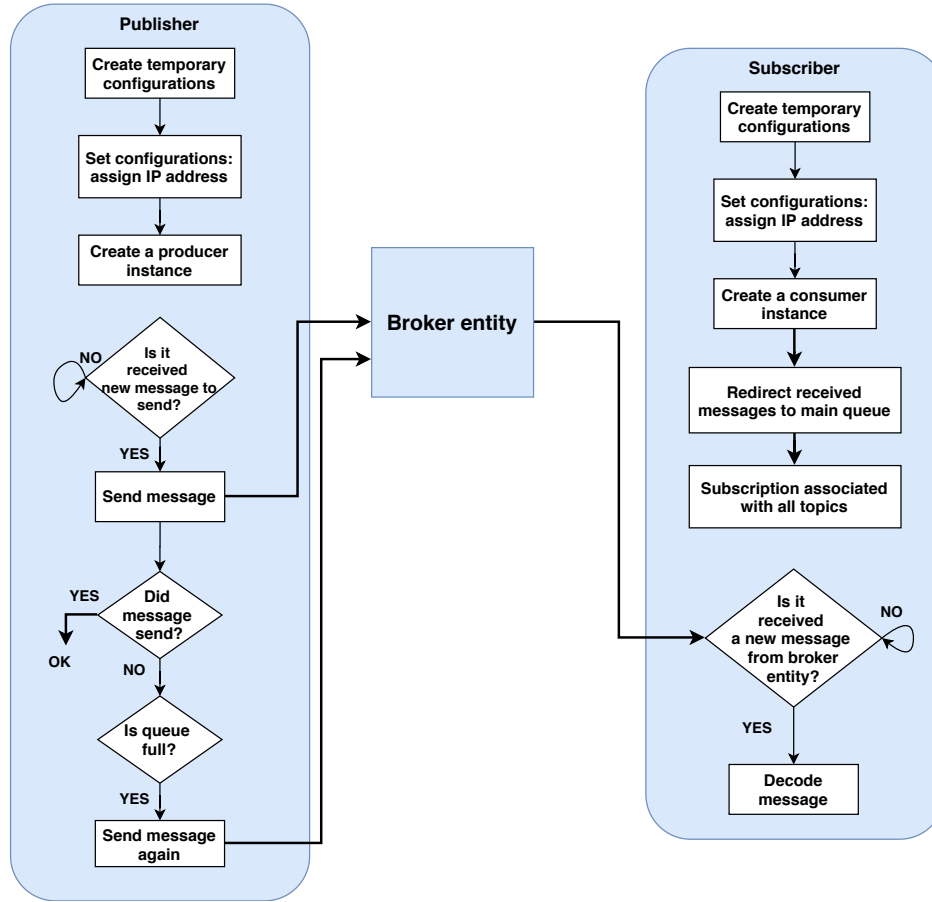


Figure 4.5: Kafka flow.

4.4.4 Message broker VM

The message broker VM is only used to deploy the server for the different message brokers implemented. Table 4.6 describes the versions used for the server brokers.

Broker	Version
RabbitMQ	3.8.3-1
Kafka	2.12-2.4.1
Mosquitto	1.6.9
MQTT protocol	3.1.1

Table 4.6: Version of server brokers.

4.5 EXTRACTION DATA FROM CAPTURES

This experience was tested multiple times both for a version with and without brokers. Since the aim is to run various users starting at the same time, we implemented a Flask⁶ server with the scripts to run the proper modules. In each VM, a server was

⁶Flask: <https://flask.palletsprojects.com/>

implemented and a subprocess was configured to run the specific scripts. Also, in the server, a tcpdump⁷ capture was initiated in the specific interfaces. An example of this configuration is depicted in figure 4.6.

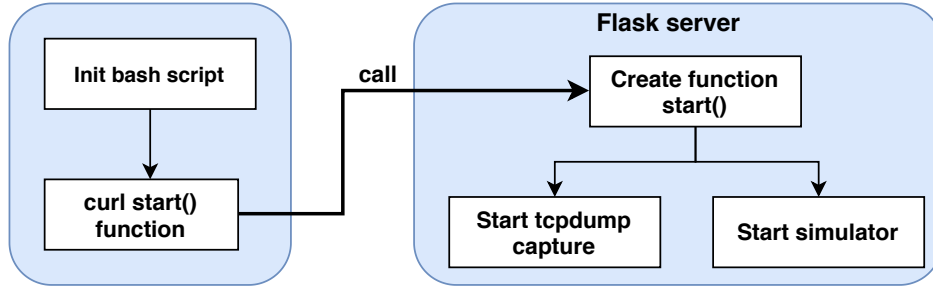


Figure 4.6: Implementation of the Flask server.

With this configuration, we try not only to start the emulators at the same time but also to stop them. One of the aims is to stop the emulators after the attachment and measure its execution time. Despite some delay, we can send a kill command, also defined in the server, to stop the emulation when the function is called. After the runs are concluded, we need to extract some data from its captures. For that, it was used a wrapper for tshark, allowing the parse of the python packets using Wireshark dissectors, PyShark⁸.

Three different functions were defined and their source code is presented in appendix E. The first one is the function to extract data from the S1AP protocol. For this, all the captures were analyzed, and the messages were saved in a dictionary. However, to assign the message to a particular user, i.e., defined by the IMSI, but the IMSI is only defined at the first message received. After analyzing the protocol fields, an ID is created for each user that connects to the MME. This way, with the ID, we could assign all the messages to a specific user.

The second function has the role to extract data from the diameter protocol. The request messages have one of the fields filled with the IMSI of the user; however, the response message does not have this parameter. It has a parameter called “hopbyhopid” which is defined in both request and response message.

Finally, the last function aims to extract messages of the GTP protocol. Only the first message of this protocol has the IMSI. So, to associate the others’ messages with this IMSI, the tunnels’ headers were analyzed. Between the different messages exchanged, they shared the same TEID. And with that, it was possible to associate the different messages to the specific IMSI.

⁷tcpdump: <https://www.tcpdump.org/>

⁸PyShark: <https://pypi.org/project/pyshark/>

4.6 SUMMARY

The proposed solution was developed with success. The first step was to configure the OAI modules, which is the base of the integration of the message brokers.

Section 4.2 describes all the necessary steps to configure eNB and UE modules, and the steps of the core network are defined in section 4.3. With all these configurations OAI is ready to execute and it is possible to understand all the flow of the exchanged messages.

With the study of the exchanged messages and all the code structure it was possible to integrate the different message brokers, described in section 4.4.

Lastly, section 4.5 describes the process to fetch data from the tcpdump captures.

5

CHAPTER

Results

This chapter presents the tests for assessing the effect of the brokers' integration. Section 5.1 presents the main scenario where the architecture can be tested. Section 5.2 presents the strategy to collect the data. The signaling impact of the 3GPP messages and the messages exchanged using brokers are analyzed in section 5.3. A detailed analysis of each pair of messages is provided in section 5.4. Sections 5.5 and 5.6 present the attachment time and execution time for the different implementations.

5.1 SCENARIO

The solution implemented for this thesis was, as stated previously, to integrate different message brokers in the control plane. To do so, an open-source solution, OAI, was used. Different types of message brokers were deployed in this platform and their usage impact on the network was evaluated.

The main scenario is to connect different amounts of users with the various implemented message brokers and compare their behavior with the version without message broker. The user runs over the OAI emulator; however, as the OAI emulator is not stable with more than three users, we decided to define two users per emulator and connect five emulators to the OAI core network. The user is defined to run twice: one with different types of message brokers and another without them. The user connects to the OAI core network, and when it is successfully connected to the network, we send a command kill to the emulator. To send the command to kill the emulator's machines, it was necessary to have installed in all VMs a Flask server to control the flow remotely. Due to this, we can kill all the emulator machines with few milliseconds of difference between them. In the runs that follow, we increase the number of users by one until a maximum of ten users. All these experiences ran 100 times. Figure 5.1 presents the scenario's workflow.

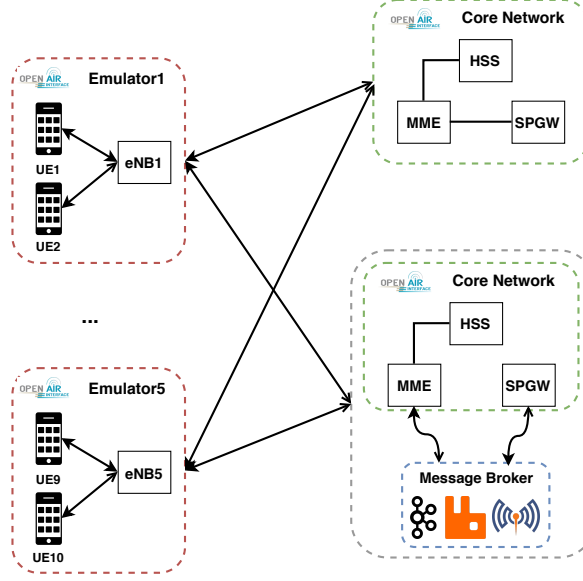


Figure 5.1: OAI with and without message broker scenario.

5.2 DATA GATHERING

To evaluate the integration of the different messages brokers, we need to compare their performance with the original version of the 4G core architecture, i.e., original EPC. For this, in all interfaces of the architecture, we captured packets using the tcpdump tool. However, in the interfaces connected to the message brokers, instead of the tcpdump tool, we use the C library `<sys/time.h>` to output the logs of the MME and S/P-GW machines as the messages are sent and received. With the data gathered we can analyze the attachment time, execution time, and each entity's processing message delay.

5.3 SIGNALING IMPACT

This section aims to analyze the size of the signaling messages in the implemented architecture and their impact on the control plane.

5.3.1 Messages defined by 3GPP

The first step is to analyze the control messages defined by 3GPP. The table 5.1 shows the control plane messages by interface and their size. On the other hand, section 2.1.4 presents the protocols used by each interface of the EPC.

Interface	Message	Size(bytes)	Payload(bytes)
S1-MME	InitialUE, Attach Request, PDN connectivity	158	96
	Authentication Request	142	62
	Authentication Response	138	59
	Security Mode Command	118	39
	Security Mode Complete	134	56
	Attach Accept	274	193
	UE Capabilty Information	126	46
	Attach Complete	182	101
S6a	Authentication Information Request	342	276
	Authentication Information Answer	358	292
	Update Location Request	330	264
	Update Location Answer	610	544
S11	Create Session Request	188	142
	Create Session Response	159	113
	Modify Bearer Request	85	39
	Modify Bearer Response	60	14
	Release Access Bearer Request	64	18
	Release Access Bearer Response	60	14

Table 5.1: Size of messages defined by 3GPP.

5.3.2 Size of messages using brokers

With the integration of different brokers, the direct interfaces between MME and S/P-GW disappear, and the exchange is going to be done through publish and subscribe methods. In table 5.2 is presented the size as well as the payload of each request and response messages for each broker.

Broker	Message	Size(bytes)	Payload(bytes)
Kafka	Request Message	1234	1164
	Response Message	129	41
Mosquitto	Request Message	1124	1055
	Response Message	1124	1055
RabbitMQ	MME -> Broker	1212	1146
	Broker -> MME	1260	1194
	S/P-GW -> Broker	1222	1156
	Broker -> S/P-GW	1250	1189

Table 5.2: Size of messages sent through the different brokers.

Analyzing the results, we can see that Mosquitto has the same size for both request and response messages, but they differ for Kafka. On the other hand, for RabbitMQ, the

message's size is different between the exchanged messages through the entities. During the request message, RabbitMQ sends an information to the subscriber client about where it should send the response (private queue), i.e., for which queue it should send the response. When a subscriber sends the response, the field related to `routing_key` is filled with the previously created private queue, which is composed of more bytes. For this reason the response messages (S/P-GW \rightarrow Broker and Broker \rightarrow MME) present higher size than request messages.

5.3.3 Throughput

This subsection aims to provide the mean throughput generated by the messages per-interface (S1-MME, S6a, and S11) of the control plane during the execution. To measure the throughput, the first step consists of filtering the messages by protocol per-interface. After that, taking into account each message's size and the time between the first and last messages, the mean throughput was calculated. Table 5.3 illustrates these values for each number of users connected to the network during the execution time for original EPC and the different message brokers. Analyzing the original EPC,

#User	Throughput (kbps)					
	Original EPC			Message Brokers		
	S1-MME	S6a	S11	Kafka	Mosquitto	RabbitMQ
1UE	15.7(± 0.64)	55.66(± 4.27)	0.663(± 0.057)	4.66(± 0.092)	7.52(± 0.429)	8.06(± 0.493)
2UE	12.94(± 0.91)	44.03(± 7.66)	0.86(± 0.105)	5.95(± 0.791)	9.74(± 2.769)	11.24(± 4.314)
3UE	13.27(± 0.86)	44.37(± 4.85)	0.864(± 0.104)	6.03(± 0.927)	9.86(± 2.332)	10.61(± 1.832)
4UE	13.14(± 0.93)	51.43(± 5.14)	0.977(± 0.395)	5.96(± 0.491)	9.43(± 0.809)	11.37(± 3.054)
5UE	13.67(± 0.53)	57.56(± 3.2)	0.977(± 0.189)	6.26(± 1.165)	9.68(± 1.799)	12.08(± 4.182)
6UE	13.12(± 0.24)	58.41(± 3.05)	1.037(± 0.275)	6.48(± 1.407)	10.22(± 1.885)	11.28(± 1.743)
7UE	14.23(± 0.18)	59.47(± 1.30)	1.003(± 0.114)	6.54(± 1.113)	10.26(± 2.697)	12.38(± 4.291)
8UE	13.15(± 0.14)	55.74(± 2.18)	1.029(± 0.181)	6.49(± 1.018)	9.89(± 1.3)	12.45(± 6.265)
9UE	13.19(± 0.11)	54.99(± 1.73)	1.066(± 0.582)	6.27(± 1.189)	10.41(± 0.946)	11.83(± 1.556)
10UE	13.17(± 0.11)	52.14(± 2.65)	1.022(± 0.181)	6.61(± 0.961)	11.29(± 2.634)	12.07(± 1.935)

Table 5.3: Mean throughput per interface for different implementation.

S1-MME and S6a interfaces start with a higher value, and for the following users, their values present slight variations. On the other hand, the S11 interface tends to increase its values with the rise of users. For message brokers, their results usually remain closer with the increase of users.

The only interface that we can compare with the integration of the messages brokers is the interface between MME and S/P-GW entities, S11 interface. When comparing the original EPC (S11 interface) values with the different messages brokers, we observe that the values and the standard deviation are higher than original EPC. The execution

time using brokers are higher than the original EPC as well as the size of messages, as presented in table 5.2. Due to this message brokers' results are higher.

5.4 DELAY ON THE CONTROL PLANE

This section will provide the processing time for each pair of messages in each entity for the control plane for the different users connected. To calculate each pair of message's processing time, we need to consider the time between a request and a response. Table 5.4 presents all pairs of messages that we are going to consider.

Pair Message	messages
Attach Req - Attach Complete	msg1
Authentication Info Req - Authentication Info Ans	msg2
Authentication Req - Authentication Rsp	msg3
Security Mode Command - Security Mode Complete	msg4
Update Location Req - Update Location Ans	msg5
Create Session Req - Create Session Rsp	msg6
Initial Context Req - Initial Context Rsp	msg7
Initial Context Req - UE Capabilty Indication	msg8
UE Capabilty Indication - Initial Context Rsp	msg9
Modify Bearer Req - Modify Bearer Rsp	msg10
Release Access Req - Release Access Rsp	msg11
Request MME -> Broker - Request Broker -> S/P-GW	msg12
Response S/P-GW -> Broker - Request Broker -> MME	msg13

Table 5.4: Assign a delta message to the different pairs of messages.

The first step is to analyze the processing time for the original EPC and after for the different brokers implemented. For the original EPC, we have to take into account the diagram of the exchange message present in the section 3.5.1 with one more pair of messages (Release Access Request/Response).

5.4.1 Delay between UE+eNB and MME entities

Figure 5.2 represents the processing time for the pair of messages exchanged with UE+eNB and MME entities for original EPC. With the presented results, we can see that, in general, the processing time in the MME is a bit higher than in the UE+eNB. These results are expected because the MME needs to wait for the response sent by the user. However, from a certain number of users, the values tend to keep constant.

For the first pair of messages msg1, in UE+eNB, the values increase with the rise of the users; however, from seven/eight users till ten, time tends to keep constant. On the other hand, in MME, there are slight variations with the rise of users, but the values, in general, remain close to 800ms.

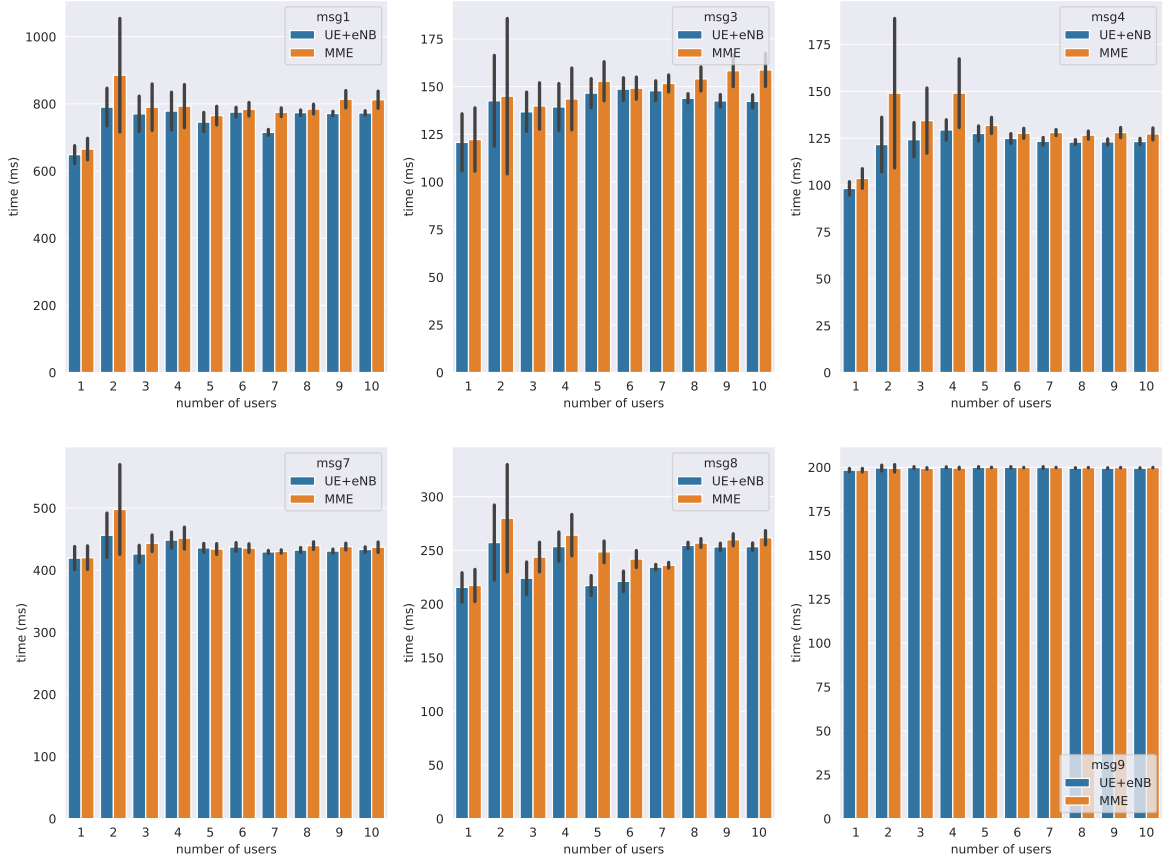


Figure 5.2: Delay time for messages exchanged with UE+eNB and MME for original EPC.

The second and third pairs of messages, msg3 and msg4, respectively, have slight variations on the results with the rise of the users in both entities. For msg3, in MME, the values increase with the rise of users. On the other hand, in UE+eNB, the values tend to decrease. For msg4 initially, the values increase for both entities, but from four users till ten, time tends to keep constant, around 125ms.

To analyze the fourth pair of messages, msg7, we need to take into account that another message is exchanged between itself, the UE Capability Indication. The msg8 and msg9 present the difference between this new message and the Initial Setup Request and Response, respectively. Through these results, we can observe that msg8 starts with slight variations, but from six/seven users, the values keep constant and close for both entities around 250 ms. For msg9, the values stay uniform, around 200ms, for both entities with the rise of users.

Analyzing these three pairs of messages, we can see why the msg7 has high values. Msg7 initially has little variations, but from five users, its values tend to keep constant for both entities.

Furthermore, there is a peak between two and three users, presented in general for all pairs of messages. During the execution of the emulator, we can see there are delays

related to the messages processing. It is also possible to identify different behaviour in each execution when more than one user is configured, i.e., in each execution for two users, for example, the behavior of initializing each user is different. Initially, the emulator establishes the connection by sending a message to MME. After that, MME sends an acknowledgment to the emulator, and it continues with its execution. The emulator will initialize the pre-configured users, and when a user is enabled, the first message is exchanged. However, when one more user is configured to run, the behavior of the emulator changes. To understand why this happens, the captures for more than one user were analyzed. It was observed that the process to enable the users is not the same. With more than one user, the emulator could enable the users with small differences between them, i.e., a few milliseconds after the first, or could enable the second user in the middle of the first one's execution. All the messages will be processed for the order that they arrived in the MME entity. Due to this and considering the different behavior, we noticed changes on execution times as well as standard deviation. For example, in the case that two users were configured, if the second user was initialized in the middle of the processing of the first user, its execution would be interrupted. That means the first user could have more processing time and, consequently, as the behavior is not the same, the standard deviation changes too. Due to this, we observe a peak from one to two users. Given the observed peak between two and three users, we need to take into account that there are two VMs with the emulator, one with two users configured and the other with one. With the gathered results it was possible to see that the mean time is approximately between the mean from one and two users, i.e., for one user the emulator has better behavior than with more. In the end, we can observe a decrease in the mean time due to the normal behavior of the emulator and inconsistencies when the emulator is problematic. However, analyzing the behavior for the following users: when the emulator has the maximum number of users (two users) defined there are small increase in the mean time. Summarizing, it can be seen that, for fewer amounts of users the emulator can display some high variability of the signaling delay. Such variability decreases with the increase of the users number, as the amount of signaling increases as well, thus reducing the impact of such variation.

5.4.2 Delay between MME and HSS entities

The pair of messages exchanged between HSS and MME are msg2 and msg5 as presented in figure 5.3. Both pair of messages demonstrate slight variations with the rise of users; however, these variations are not greater than one millisecond.

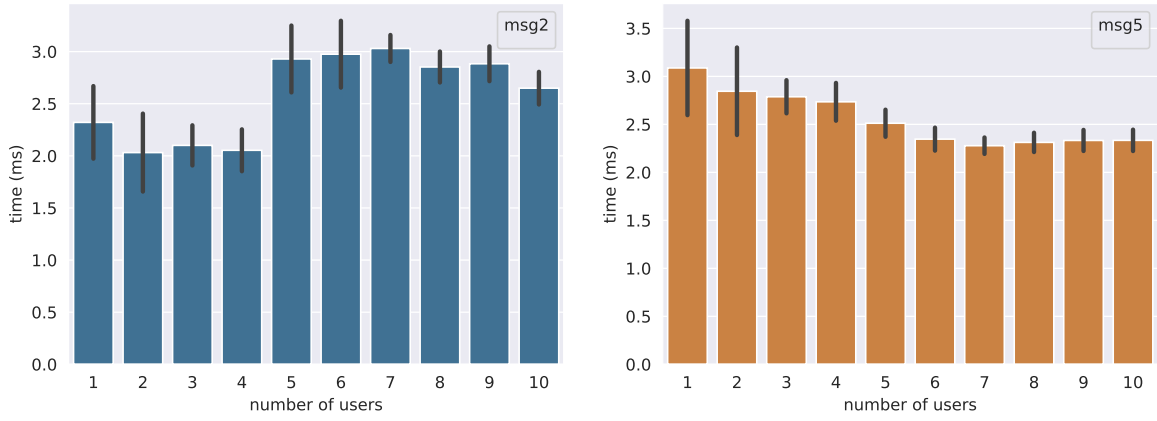


Figure 5.3: Delay time for each pair of messages exchanged between HSS and MME entities.

5.4.3 Delay between MME and S/P-GW entities

Now, we are going to compare the messages exchanged between the MME and S/P-GW entities for original EPC and the different implemented brokers. In general, the values are higher in MME than in S/P-GW. The first pair of messages to be analyzed is msg6, presented in figure 5.4. Comparing original EPC with the different brokers

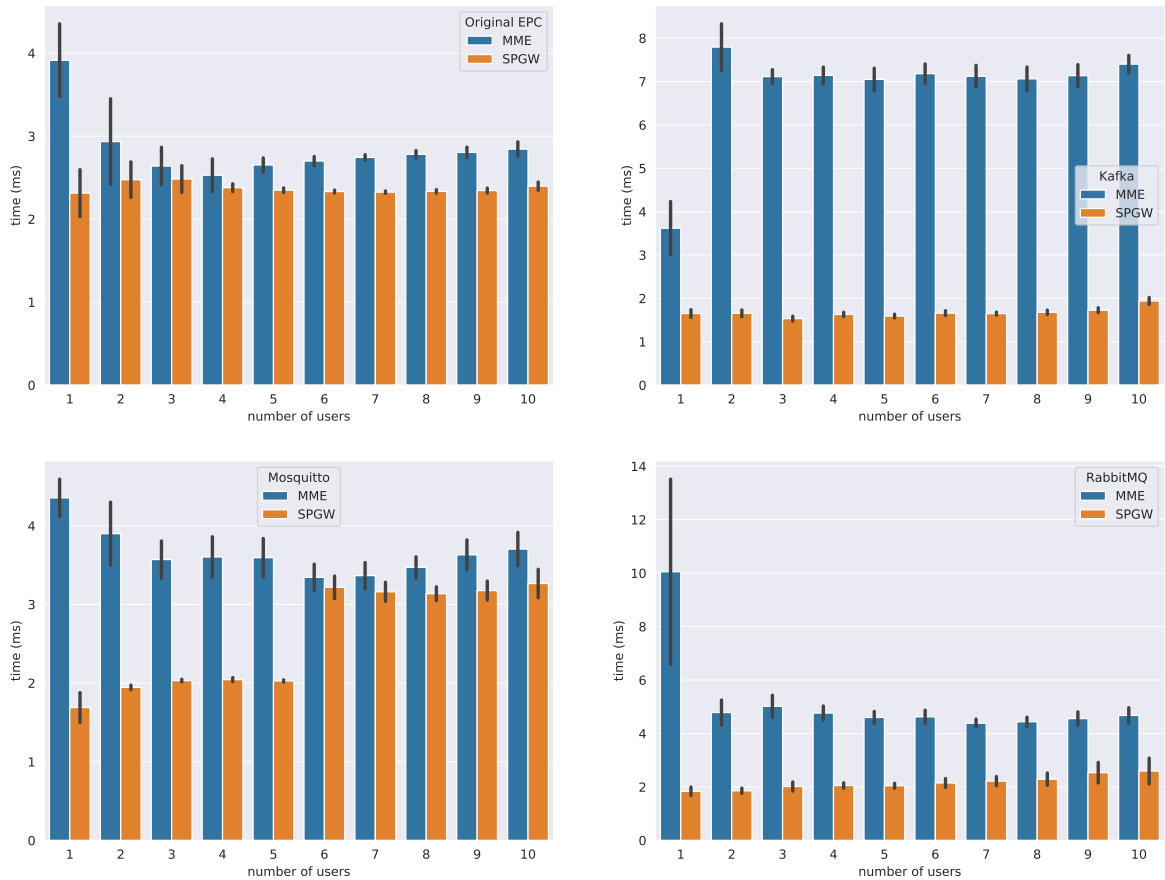


Figure 5.4: Delay time comparison for the message pair msg6 for different implementations.

implemented, we noticed a peak value for one user when looking into the remaining. For original EPC, with the rise of users, the values have slight variations in the MME entity; however, in the S/P-GW entity, values tend to be constant from four users. All implemented message brokers present slight variations with the rise of users. Kafka broker presents a big difference between MME and S/P-GW with a tendency to keep its values between 7ms and 8ms in MME and closer to 2ms in SPGW. On the other hand, Mosquitto tends to keep its values close, and RabbitMQ presents slight variations with the rise of users for both entities. We can conclude for this pair of messages that the performance of message brokers are higher than the original EPC. However, Mosquitto and RabbitMQ present values close to original EPC, where Mosquitto has the best performance. Nevertheless, these variations are not greater than two milliseconds.

The second pair of messages is msg10, illustrated in figure 5.5. Original EPC has

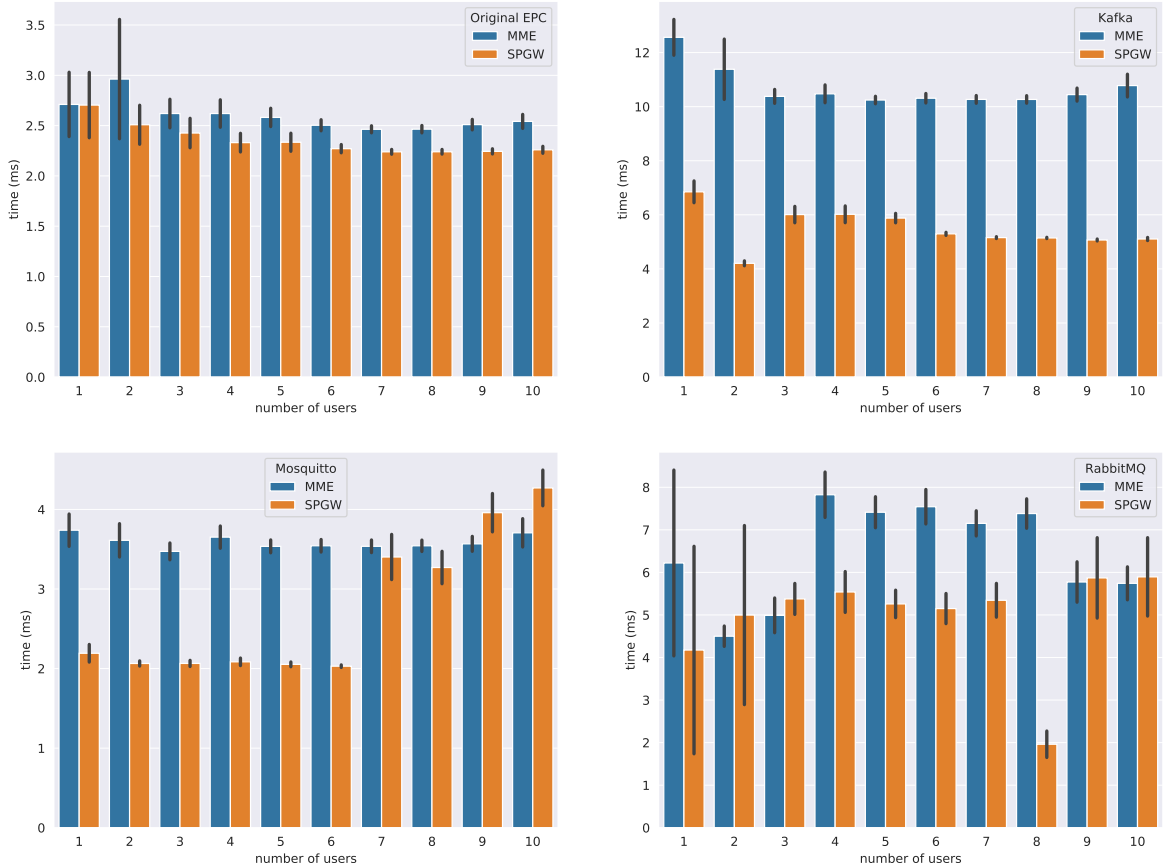


Figure 5.5: Delay time comparison for the message pair msg10 for different implementations.

its values close to both entities. On the other hand, Kafka presents a big differences for both entities, although for MME the values are higher. Mosquitto initiates with higher contrast between entities but tends to keep close, and from nine users, the S/P-GW values increase and they are higher than MME. RabbitMQ presents variations with the rise of users for both entities. It starts with MME higher than S/P-GW with a big

difference, but from nine users, the values keep close but S/P-GW's are still higher. For this pair of messages the variations are not higher than three milliseconds, in general, Mosquitto has the implementation that offers closer values when compared with original EPC.

The last pair is msg11, shown in figure 5.6 presenting a big difference in their values for MME and S/P-GW entities. For original EPC, the values tend to decrease

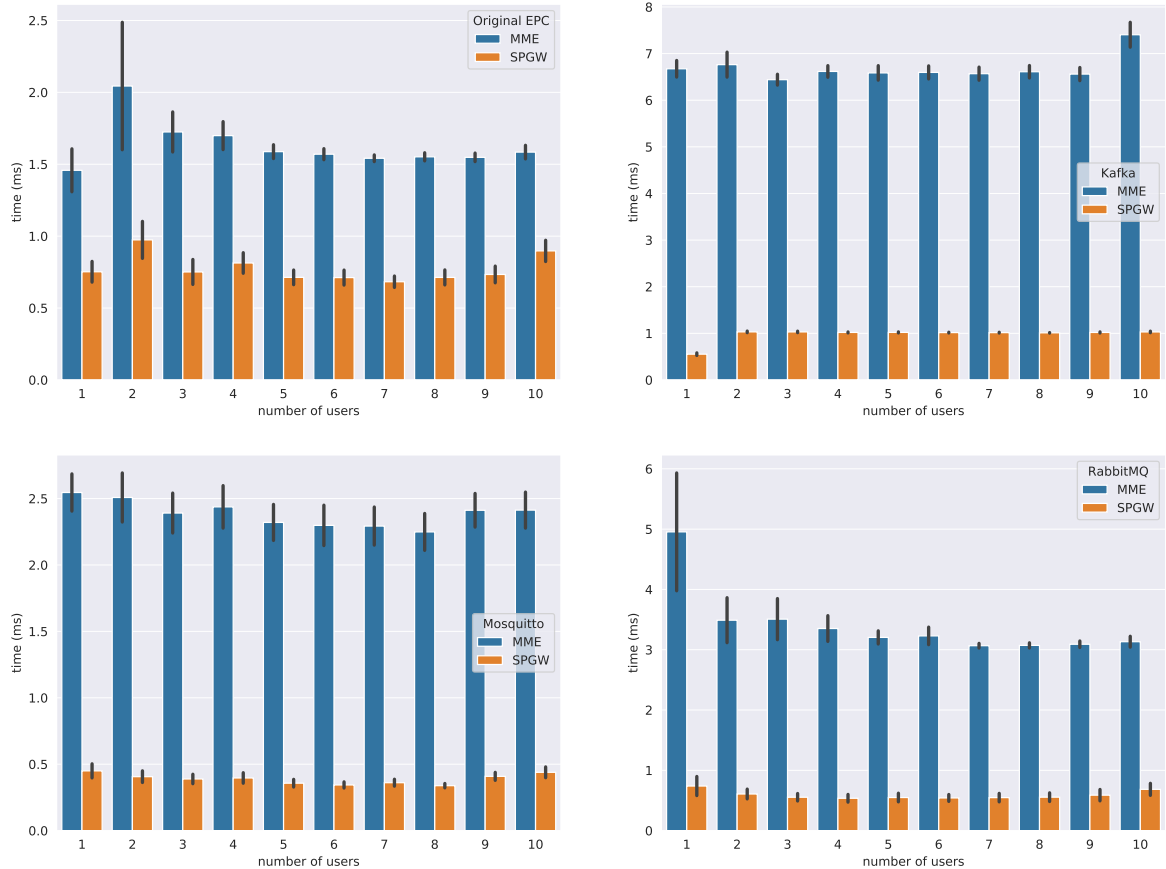


Figure 5.6: Delay time comparison for the message pair msg11 for different implementations.

with slight variations. In Kafka broker, we verify in both entities to keep their values constant, around 600ms and 100ms, for MME and S/P-GW, respectively. However, for ten users in MME, the value increases. Mosquitto presents variations with the rise of users. On the other hand, RabbitMQ tends to decrease its values in MME; however, in S/P-GW, the values start by getting lower, keeping constant from three to eight users and increase slightly until ten users. For this pair of messages, we can observe some variations around two milliseconds, hence, mosquitto broker has, once again, values closer to the original EPC.

After analyzing these three pairs of messages (msg6, msg10, and msg11), we observed that with message brokers, their values are above or, in some cases, closer to the original EPC. These values are expected due to the presence of one more entity between MME

and S/P-GW. For this reason, the delay increases since there are delays on the network, with these delays happening due to message processing or on the software. Nevertheless, with the presence of a new entity, the variations are not greater than three milliseconds. These variations represent a small value that does not affect the execution time of the messages with the broker's presence.

5.4.4 Delay in broker entity

For the messages exchanged through the broker, we need to consider not only the processing time in MME and S/P-GW entities but also the broker's processing time. Through the packets captured we can only see the payload of each message and for this reason, we are going to analyze, in general, the processing time since the request message arrives the broker until the broker forwards it to the corresponding entity (with corresponding topic/queue). This process is illustrated in figure 5.7. For the three different brokers we have meager variations for both the request and response message. However, Mosquitto broker presents lower values when compared to Kafka and RabbitMQ. On the other hand, RabbitMQ and Kafka have their values very closely when comparing their request and response messages.

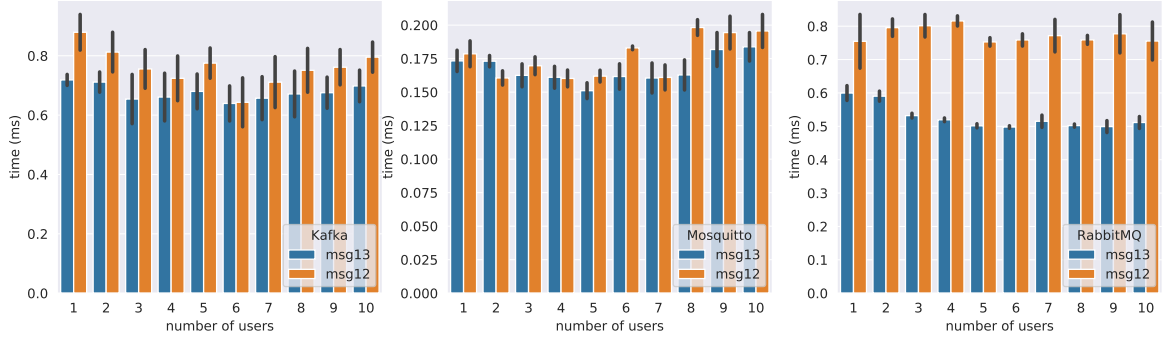


Figure 5.7: Delay time for each request and response messages in broker entity.

5.5 ATTACHMENT TIME

After analyzing the processing time for each pair of messages, we will evaluate the overall impact during the attachment time for original EPC. Then, this behavior will be compared with the proposed implementation.

5.5.1 Original EPC

To determine the attachment time it is considered the procedure described in section 2.1.5.3. This experience was done as described previously, in section 5.1, by connecting one UE, two UEs and so on till a total of ten UEs simultaneously. The obtained times do not consider the first pair of messages exchanged between eNB and

MME, since they are related to the initialization of the connection establishment. Due to this reason, we only consider the first message sent by the user.

Figure 5.8 illustrates the mean times for each number of users attached. Analyzing this figure's results we observed that the lowest value is associated with one user, 644(± 23.4)ms. From user one to user two, there is a significant increase in the mean time, approximately 300ms. We expected that with one user, the attachment time is

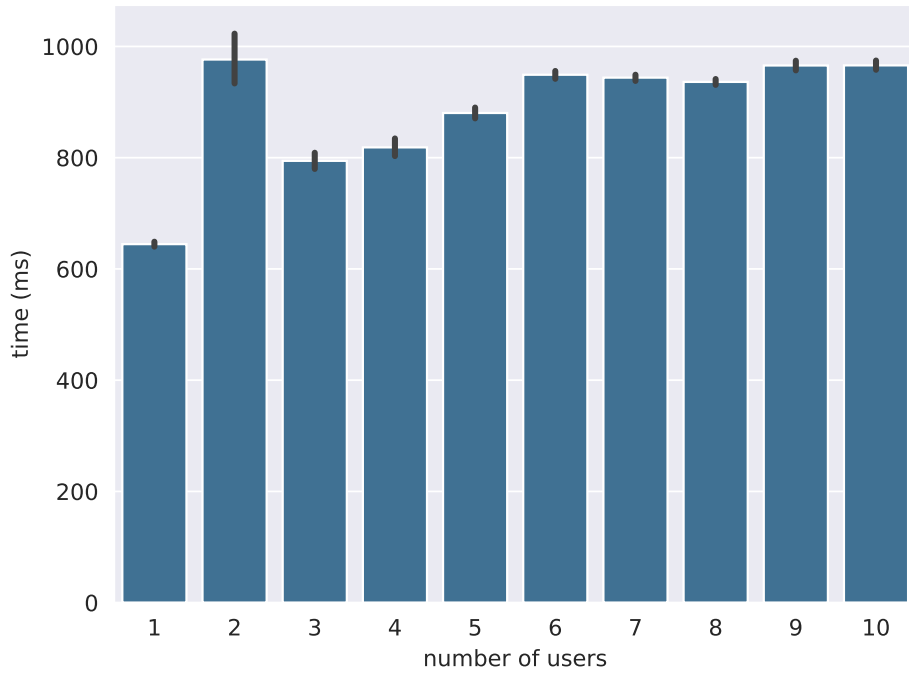


Figure 5.8: Attachment time for original EPC

lower, and with more, the mean time would increase. When comparing one to two users, this increases not only due to the existence of one more user but also because there are more exchanges between the peer-to-peer interfaces and more processing in the emulator. When one user sends the initial message a few milliseconds later, the other user sends your message. However, the processing of the messages is not going to be firstly for the first user that sends the first message but, in fact, intercalated with the other users. For this reason, we observe an increase in the mean time for one user with the rise of users. Despite the tendency to increase, there is a tendency to keep from six to ten users around 900ms, with slight variations.

5.5.2 Message Brokers

The following step was to measure the attachment time of the proposed architecture in section 3.4.1.

Figure 5.9 demonstrates the attachment time for the different brokers implemented. In general, we verified two peaks time, the lower for one user, and the higher for two

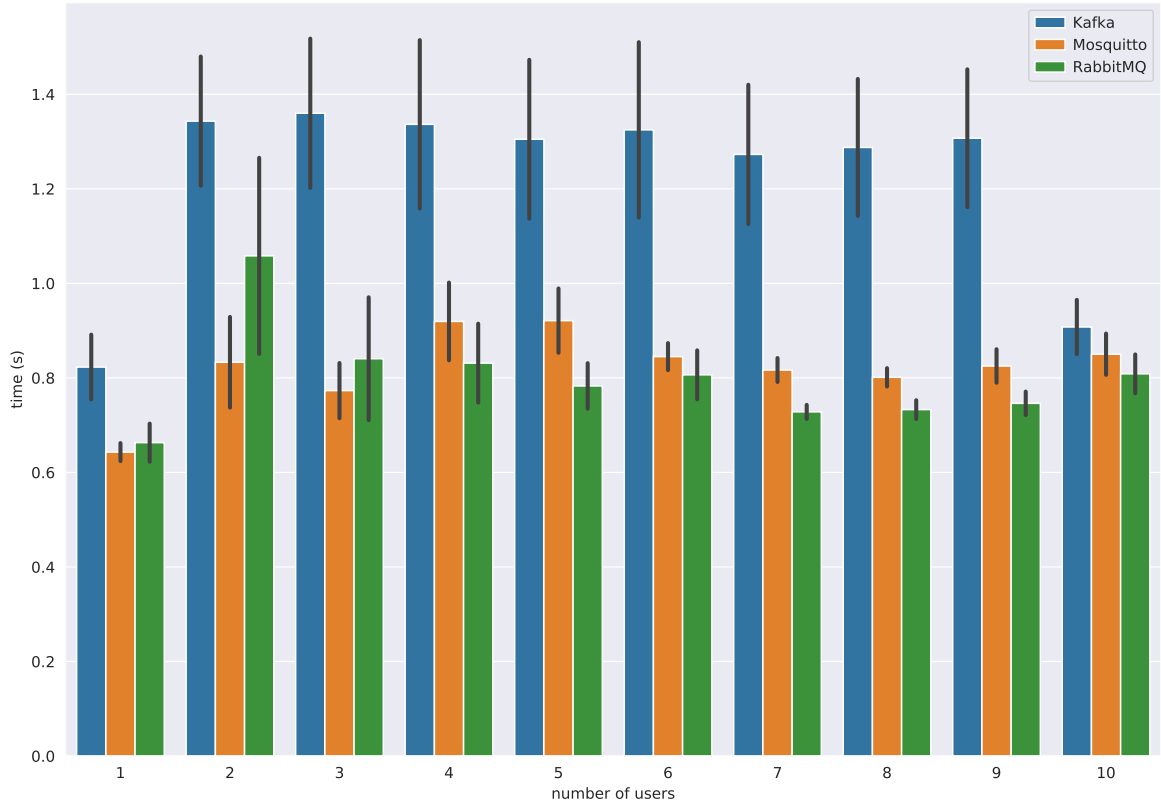


Figure 5.9: Attachment time comparison between message brokers.

users. These peaks are present for all message brokers. However, there is a tendency to decrease attachment time with an increase in the number of users.

Kafka has a different behaviour at the beginning by always presenting higher mean time in contrast to the other brokers. From two until nine users, the mean time presents slight variations, around 1.3s. We can only verify lower values, 800ms and 900ms, for one and ten users, respectively. Mosquitto broker presents slight variations with the rise of users, and from six/seven users tends to keep the values uniform, around 800ms. RabbitMQ broker initiated with higher values when comparing with the other implemented message brokers. Nevertheless, from four users its values tend to keep lower than the others.

5.5.3 Comparison between original EPC and message brokers

Figure 5.10 presents the comparison between original EPC and the different brokers implemented.

We expected that the mean time with brokers was higher than the original EPC because since there is a new entity between MME and S/P-GW. We can observe that only Kafka broker has worse performance than the other brokers compared with original EPC. On the other hand, Mosquitto and RabbitMQ have, in general, lower mean times

with the rise of the users when comparing to original EPC.

The proposed solution only has to take into account a new entity between MME and S/P-GW, due to this, only two pair of messages are exchanged using message brokers for attachment time. Previously, we observed that these two pairs of messages (msg6 and msg10) for Kafka broker present the highest values, and Mosquitto broker has the lowest values. Considering these results, we expected that the attachment time for Kafka is higher, and Mosquitto is lower. However, these values are a small percentage of all the processing time. The emulator used does not have the same behavior throughout all emulations done. Due to this, we observed different responses for different implementations.

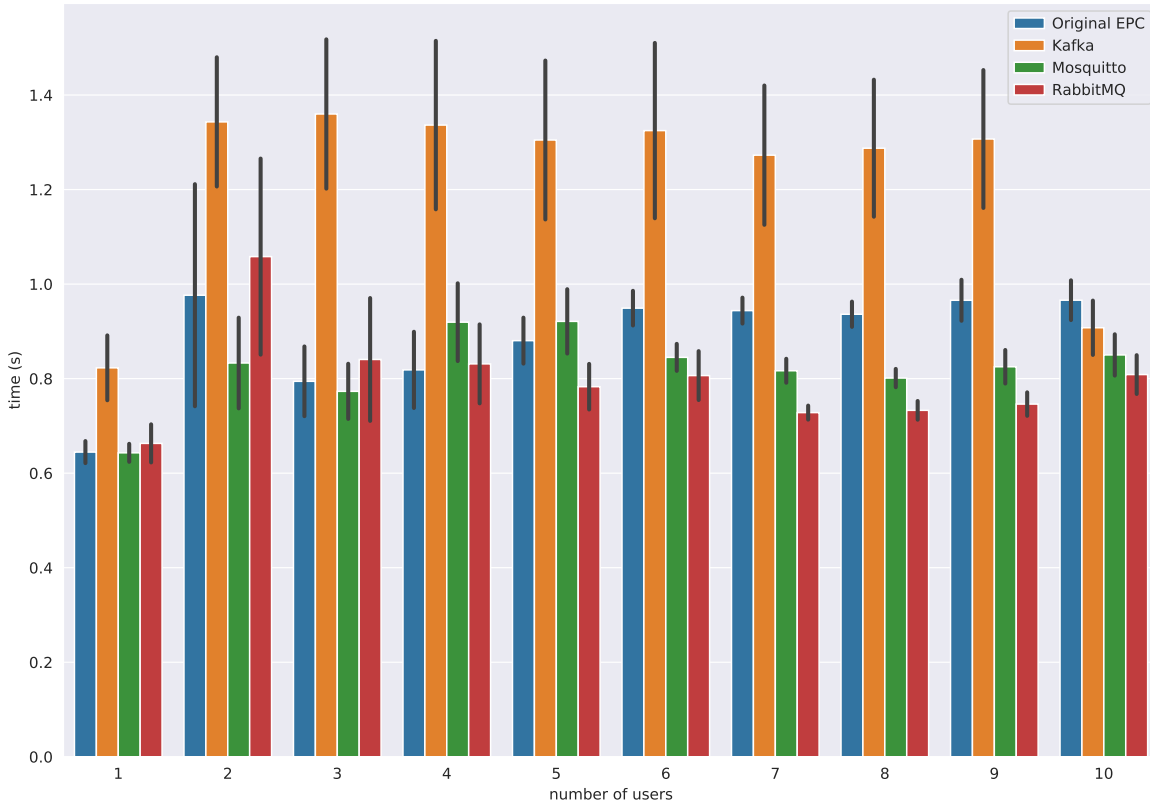


Figure 5.10: Attachment time comparison between original EPC and message brokers.

5.6 TOTAL TIME

This section has aims to provide the total mean time of the original EPC and message brokers. That means the time since the user sends the first message until disconnects from the network. During the experience, the number of connected users will be shown, whereas the attached appears only at the end. All this information is given from the MME entity. So, when the MME has the attached users, it sends a kill command to the

emulator. For this reason, we analyzed the time between the first message sent from the user and the last sent from S/P-GW to MME (Release Access Request/Response).

5.6.1 Original EPC

As mentioned before, we are going to analyze the total mean time, described in section 3.5. For this, we need to first analyze it for the original EPC, presented in figure 5.11.

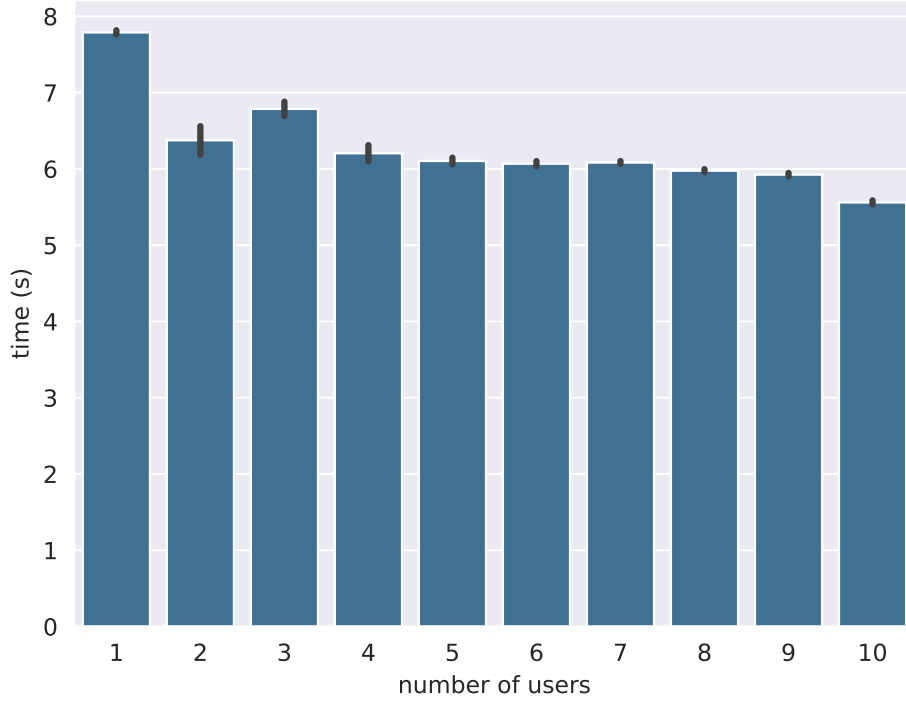


Figure 5.11: Total time for original EPC.

Considering the results of this figure, we observed a peak time for one user, $7.78(\pm 0.14)$ s. However, for the other number of users, there is a decrease in time when compared with one user attached. Also, we noticed that there is a tendency to decrease the mean time from three users with the values being between 5.5s and 6.7s.

All these values are higher because there is some delay to send the kill command. We need to consider the time of MME knowing all the attached users, sending the kill command to the OAISIM VM, and the processing of the command in it.

When comparing with figure 5.8, there is also a big difference from one to two users, but in this case, it is the opposite. Some conditions could influence the final results, such as delay on the network, delay of processing from the different VMs, and the processing of the kill command. Despite all the users are running in different VMs, there is a tendency to keep a better performance with its rise.

5.6.2 Message Brokers

The following step is to analyze the total mean time for different brokers implemented, presented in section 3.5.2. Figure 5.12 shows the overall mean time between message brokers.

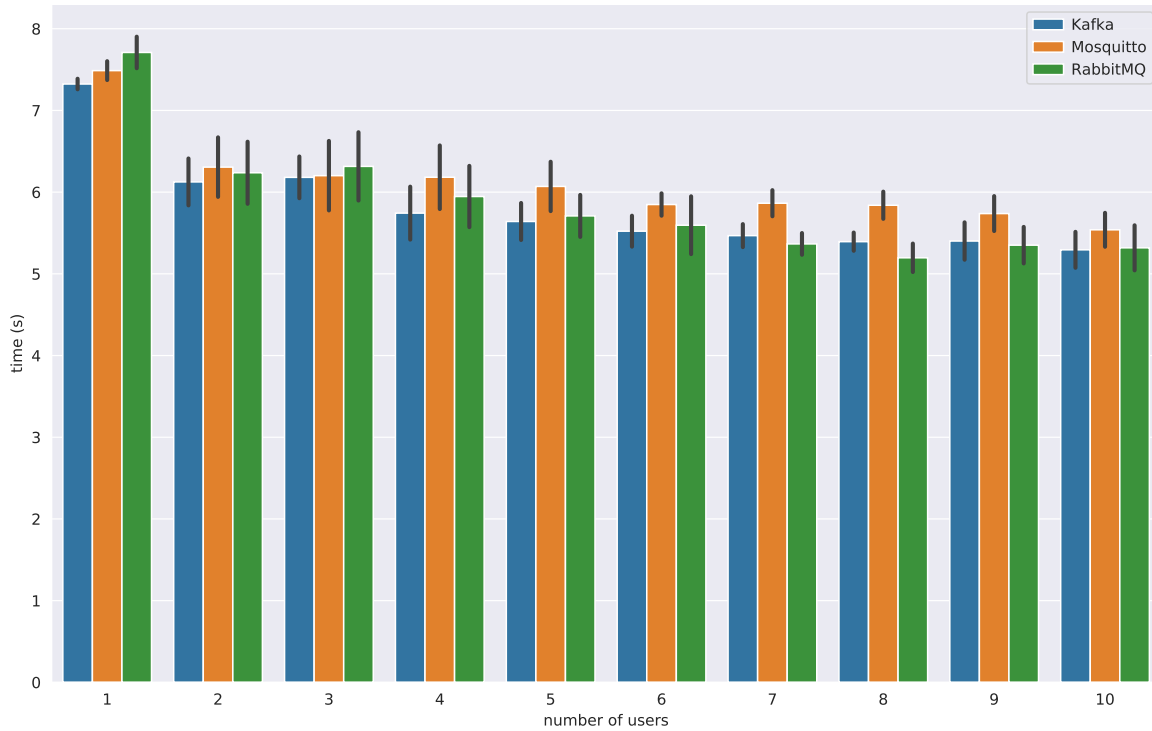


Figure 5.12: Total time using message brokers.

Observing the results, we noticed a higher peak value from one user. In general, there is also a tendency to decrease the total mean time with the rise of users.

All brokers have similar behavior with the increase of users, i.e., their values tend to decrease. Mosquitto broker presents higher values when compared with the other brokers. Although it decreases its values slightly with the rise of users, presenting its lower value for ten users, $5.53(\pm 0.2)$ s. Kafka broker presents its lower values when compared with Mosquitto and RabbitMQ, with its lowest point being for ten users, $5.29(\pm 0.22)$ s. In some cases RabbitMQ's values are lower.

5.6.3 Comparison total time between original EPC and message brokers

Figure 5.13 shows the comparison between original EPC and message brokers. When comparing with the attachment time, in section 5.5.2, as expected, the total time is higher than the attachment time wherein there is more delay in processing the attached users and send the kill command. Furthermore, for the attachment time, we saw that Kafka broker has values higher than the original EPC; on the opposite, the total time does not present it.

For this experience, we have to take into account that brokers only are responsible for exchanging messages between MME and S/P-GW entities, corresponding a few pairs of messages when comparing the pairs of messages exchanged between OAISIM and MME entities. Whereas for the attachment time, only two pairs of messages are exchanged, apart from the total time where one more pair is exchanged. With all these considerations and given that there is some delay to the broker process the received message and forwarding to the correct topic, we can observe that the total times for the different brokers implemented are very close to each other.

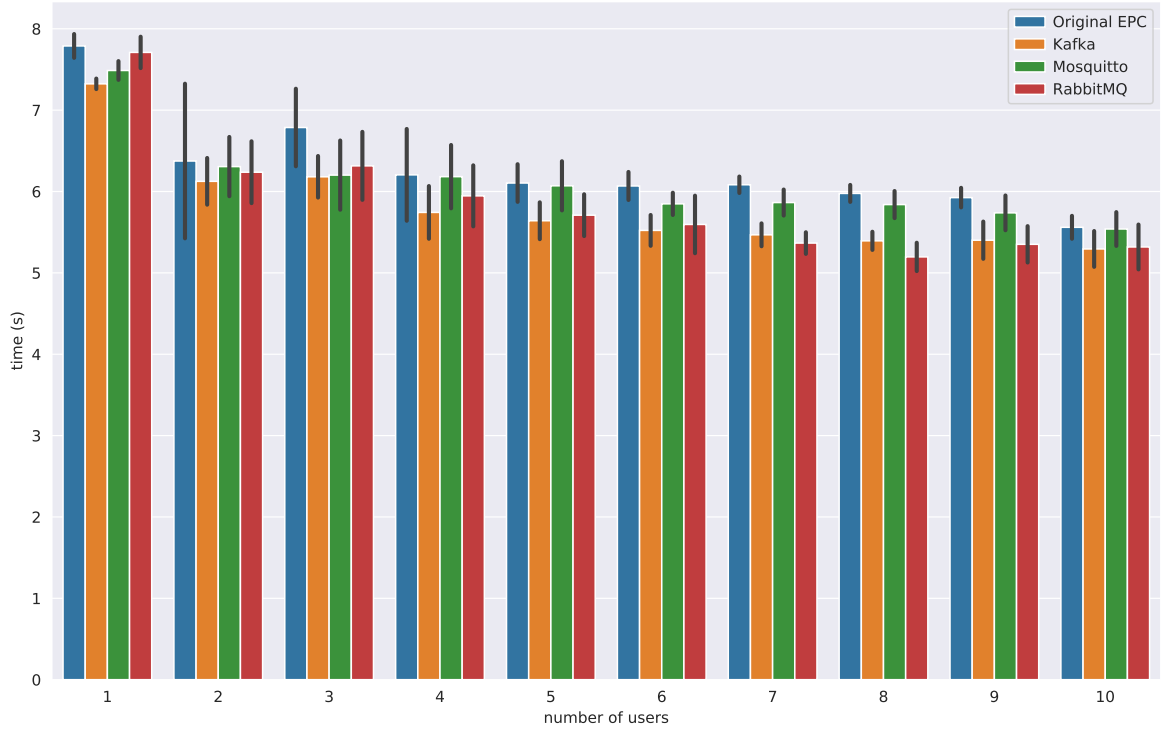


Figure 5.13: Total time comparison between original EPC and message brokers.

So, we have some delays throughout the execution due to emulator or delay caused by the broker. However, only three pairs of messages are exchanged through the broker, but they present better performance. Then, we expected with more messages exchanged using a broker, the total time also decrease or could slightly rise, concerning original EPC.

5.7 LATENCY

After a user attachment, an interface was created for each user. This section aims to study if the different implementations in the control plane affect the data plane. This way, the ping tool was used to generate Internet Control Message Protocol (ICMP) requests every second and wait for a reply. It was generated four packets with 48 bytes

of payload and their mean time was measured. For this experience, five pings were executed for the different implementations and for each number of users connected to the network. The results are presented in figure 5.14.

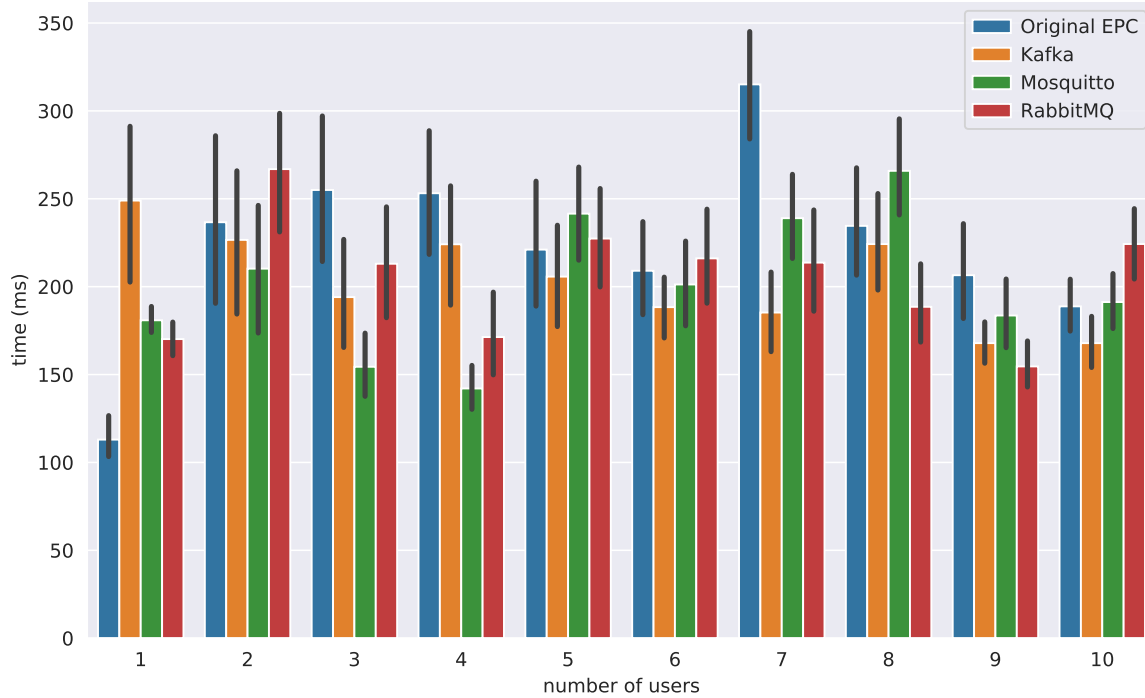


Figure 5.14: Comparison between original EPC and different brokers in terms of latency.

Analyzing the data from the table, we can see significant variations with the rise of users for the different implementations. However, in general, the results from the brokers are lower than the original EPC. Taking into account some delays in the network that influenced the results presenting higher means and, consequently, standard derivations, we can conclude that the implementations do not affect the data plane.

5.8 SUMMARY

This chapter started by, in section 5.1, defining the main scenario of the implemented solution. To test the scenario, it was necessary to run it multiple times for the different implementations and each number of users. This process was described in section 5.2.

During testing the scenario, the results were collected. All exchanged messages and their generated traffic were analyzed in section 5.3.

The effect of using multiple users for different implementations and distinct pair of messages was analyzed in detail in section 5.4. When comparing to the processing times for original EPC, the values are higher in the MME entity than the other entities. However, some pairs of messages tend to stay uniform from half of the users. Another

comparison made was the pair of messages exchanged between MME and S/P-GW, using message brokers. All pairs of messages have higher mean times for message brokers than original EPC. However, Mosquitto has the implementation that offers closer values to original EPC, for all messages exchanged between these entities.

The attachment time was analyzed for each implementation in section 5.5. With the rise of users, the values tend to increase for all implementations. However, the results for messages brokers are lower than the original EPC. In section 5.6 it was presented the total mean time of execution for the different implementations. Analyzing their results, it was possible to notice an increase in mean time for all implementations when comparing with the attachment procedure. With the rise of users it was verified a decrease in the total mean time. As for attachment, the brokers implemented have values lower than original EPC.

As pointed out before, it was verified that the available emulator tool used was designed considering simple signaling conformance testing, and not performance testing. As a result, the emulator presented some variations for some executions and according to the defined number of users. The results presented in this chapter were the best gathered from this tool. This tool has difficulties connecting to users with their increase, and for this reason, we observed some delays and standard deviations.

The last section provided an evaluation of the effect of these implementations on the data plane. With the presented results for the different implementations and taking into account delays at the network, these implementations do not have an effect on it.

Conclusions

This work presented a first assessment of progressing intercommunication mechanisms between mobile network core entities from dedicated interfaces into the broker-based design, going one step beyond the current pursuit of cloud-native deployment. Using the OAI platform it was possible to study the impact of the proposed changes on the control plane. It also provided a study of the effect on the core network using multiple users.

The new architecture provides more flexibility and scalability than the 4G network, which is an advantage to the network operators, since they will be able to meet the user's demands while keeping the costs low.

The problem stated was faced by the evolved 4G network. By working with a software platform implementing the 4G network's concepts, the ideas of 5G could be tested. Projects like OAI implement all the LTE modules and provide a tool for emulation/simulation.

The solution aimed to go beyond the cloud-native inclination of 5G networks, and progress the information inter-exchange between core entities by coupling them to a message broker. The validation addresses measuring the attachment procedure of the different implementations.

To give an overview of that impact, the message brokers were implemented between MME and S/P-GW entities and its processing time for each pair of messages in each entity was measured. Through this analysis, MME is the entity that takes more time processing for each pair of messages. On the other hand, when compared to the pairs of messages exchanged through the implemented brokers, it was possible to understand that Mosquitto has better performance and its results are closer to the original EPC. Furthermore, the attachment and execution procedures were analyzed. Both with the integration of the message brokers presented better performance. Despite the separate

study of the pair of messages using message brokers were higher than the original EPC, these results only represent a small percentage of all exchanged messages. Comparing the attachment with execution time, only one more pair of messages is exchanged. Nevertheless, there is a big difference in the mean time between them. This difference did not occur only due to the existence of one more message but also because of the delay of disconnecting the user(s) from the network. Another observation was with the rise of users; in general, the results have a tendency to keep uniform or decrease. After all this research, a study was made to comprehend if the changes in the control plane affected the data plane. A ping tool was used, and all the implementations for all users were tested. It was then possible to conclude that the changes did not affect the data plane.

According to the results gathered, we observed that using message brokers we get better performance than the original EPC. However, the solution developed did not provide a full comparison between the new 5GC architecture concepts since we only implemented brokers in one connection. Nevertheless, when comparing the pair of messages exchanged using message brokers it was possible to observe that there is no impact on time related to the original EPC. Due to this, we can conclude that with the presence of one more entity, the broker, there is no impact on the network.

6.1 FUTURE WORK

It would be interesting to repeat the same tests done using physical devices and compare the results obtained with the results of this dissertation. Given the problem stated, it would be essential to have full integration of the broker in all the control plane. A development of MME and S/P-GW using a REST API was also envisioned; however, due to the complexity presented in all code provided by OAI, it was decided not to include it on this document. Another problem was with the platform used; despite presenting all the integration of the 4G modules and the emulation tool, the UE/eNB emulator does not have a stable behavior.

Once the OAI's 5G version is available, it would be interesting to perform the same tests and compare the results to the ones presented.

References

- [1] Cisco, “Cisco visual networking index: Global mobile data traffic forecast update, 2017-2022”, Feb. 2009, Cisco, White Paper.
- [2] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, “Nfv and sdn—key technology enablers for 5g networks”, *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, 2017.
- [3] C. Zhang, X. Wen, L. Wang, Z. Lu, and L. Ma, “Performance evaluation of candidate protocol stack for service-based interfaces in 5g core network”, in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2018, pp. 1–6.
- [4] A. Ghosh, A. Maeder, M. Baker, and D. Chandramouli, “5G Evolution: A View on 5G Cellular Technology Beyond 3GPP Release 15”, *IEEE Access*, vol. 7, pp. 127 639–127 651, 2019.
- [5] *About 3gpp*, <https://www.3gpp.org/about-3gpp>, [Online; accessed 1-June-2020].
- [6] M. Nohrborg, *LTE Overview*, <https://www.3gpp.org/technologies/keywords-acronyms/98-lte>, [Online; accessed 23-March-2020].
- [7] F. Firmin, *The Evolved Packet Core*, <https://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core>, [Online; accessed 23-March-2020].
- [8] J. Wannstrom, *LTE-Advanced*, <https://www.3gpp.org/technologies/keywords-acronyms/97-lte-advanced>, [Online; accessed 27-March-2020].
- [9] P. A. Atayero, M. Luka, M. Orya, and J. Iruemi, “3GPP Long Term Evolution: Architecture, Protocols and Interfaces”, *International Journal of Information and Communication Technology Research*, vol. 1, no. 7, pp. 306–310, Nov. 2011, ISSN: 2223-4985.
- [10] S. Sesia, I. Toufik, and M. Baker, *LTE – The UMTS Long Term Evolution: From Theory to Practice*, 2nd ed. Jul. 2011, pp. 1–611, ISBN: 9780470660256. DOI: 10.1002/9780470978504.
- [11] 3GPP, “Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2 (Release 10)”, 3rd Generation Partnership Project (3GPP), TS 36.300, Dec. 2014. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/36300.htm>.
- [12] —, “Characteristics of the Universal Subscriber Identity Module (USIM) application (Release 10)”, 3rd Generation Partnership Project (3GPP), TS 31.102, Dec. 2017. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/31102.htm>.
- [13] C. Cox, “System architecture evolution”, in *An Introduction to LTE: LTE, LTE-Advanced, SAE and 4G Mobile Communications*, 1st ed., Wiley Publishing, 2012, pp. 21–44, ISBN: 9781119970385.
- [14] 3GPP, “General on Terminal Adaptation Functions (TAF) for Mobile Stations (MS) (Release 10)”, 3rd Generation Partnership Project (3GPP), TS 27.001, Mar. 2011. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/27001.htm>.

- [15] —, “General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access (Release 10)”, 3rd Generation Partnership Project (3GPP), TS 23.401, Dec. 2014. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/23401.htm>.
- [16] —, “Network architecture (Release 10)”, 3rd Generation Partnership Project (3GPP), TS 23.002, Jun. 2014. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/23002.htm>.
- [17] —, “Policy and charging control architecture (Release 10)”, 3rd Generation Partnership Project (3GPP), TS 23.203, Dec. 2014. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/23203.htm>.
- [18] E. Dahlman, S. Parkvall, and J. Sköld, “Chapter 4 - radio-interface architecture”, in *4G LTE-Advanced Pro and The Road to 5G (Third Edition)*, Third Edition, Academic Press, 2016, pp. 55–74, ISBN: 978-0-12-804575-6. DOI: <https://doi.org/10.1016/B978-0-12-804575-6.00004-2>. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/B9780128045756000042>.
- [19] 3GPP, “Evolved Universal Terrestrial Radio Access (E-UTRA) ; S1 Application Protocol (S1AP) (Release 10)”, 3rd Generation Partnership Project (3GPP), TS 36.413, Sep. 2014. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/36413.htm>.
- [20] —, “Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3 (Release 10)”, 3rd Generation Partnership Project (3GPP), TS 24.301, Sep. 2014. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/24301.htm>.
- [21] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn, *Diameter Base Protocol*, RFC 6733 (Proposed Standard), Updated by RFC 7075, 8553, Internet Engineering Task Force, Oct. 2012. [Online]. Available: <https://tools.ietf.org/html/rfc6733>.
- [22] 3GPP, “MME Related Interfaces Based on Diameter Protocol (Release 10)”, 3rd Generation Partnership Project (3GPP), TS 29.272, Mar. 2018. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/29272.htm>.
- [23] D. K. Prakasha, “Authentication and key agreement in 3gpp networks”, vol. 5, Jul. 2015, pp. 143–154. DOI: 10.5121/csit.2015.51313. [Online]. Available: https://www.researchgate.net/publication/300141344_Authentication_and_Key_Agreement_in_3GPP_Networks.
- [24] 3GPP, “3GPP System Architecture Evolution (SAE); Security architecture (Release 10)”, 3rd Generation Partnership Project (3GPP), TS 33.401, Dec. 2015. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/33401.htm>.
- [25] *OpenLTE*, <https://sourceforge.net/projects/openlte/>, [Online; accessed 08-May-2020].
- [26] *srsLTE 20.04.1 Documentation*, <https://docs.srslte.com/en/latest/index.html>, [Online; accessed 08-May-2020].
- [27] *srsLTE*, <https://github.com/srsLTE/srsLTE>, [Online; accessed 08-May-2020].
- [28] *Open EPC core network dynamics*, <https://sites.google.com/a/corenetdynamics.com/openepc/home>, [Online; accessed 08-May-2020].
- [29] *nextEPC*, <https://github.com/nextepc/nextepc>, [Online; accessed 08-May-2020].
- [30] *GNU Radio LTE receiver*, <https://github.com/kit-cel/gr-lte>, [Online; accessed 08-May-2020].
- [31] *Open Source Long-Term Evolution (LTE) Deployment*, <https://sites.google.com/site/osldproject/home>, [Online; accessed 08-May-2020].
- [32] *Technology*, <https://www.amarisoft.com/technology/>, [Online; accessed 15-May-2020].

- [33] R. Defosseux, *FAQ*, <https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/FAQ>, [Online; accessed 17-May-2020], 2019.
- [34] S. Koh and S. Lee, “Implementation of openairinterface control software for 4g network”, in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2017, pp. 747–749.
- [35] N. Nikaein, R. Knopp, F. Kaltenberger, L. Gauthier, C. Bonnet, D. Nussbaum, and R. Ghaddab, “Demo: Openairinterface: An open lte network in a pc”, *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, Sep. 2014. DOI: 10.1145/2639108.2641745.
- [36] OpenAirInterface, *OpenAirInterfaceTM (OAI): Towards Open Cellular Ecosystem*, https://www.openairinterface.org/?page_id=864, [Online; accessed 17-May-2020].
- [37] Eurecom, *E-UTRAN User Guide*, Jul. 2015.
- [38] 3GPP, “System architecture for the 5G System (5GC) Stage 2 (Release 15)”, 3rd Generation Partnership Project (3GPP), TS 23.501, Mar. 2020. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/23501.htm>.
- [39] —, “NR; NR and NG-RAN overall description; Stage 2 (Release 15)”, 3rd Generation Partnership Project (3GPP), TS 38.300, Mar. 2020. [Online]. Available: <http://www.3gpp.org/ftp/Specs/html-info/38300.htm>.
- [40] A. Toskala and M. Poikselkä, “5g architecture”, in *5G Technology*. John Wiley & Sons, Ltd, 2019, ch. 5, pp. 67–86, ISBN: 9781119236306. DOI: 10.1002/9781119236306.ch5. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119236306.ch5>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119236306.ch5>.
- [41] S. Ahmadi, “Chapter 1 - 5g network architecture”, in *5G NR*, S. Ahmadi, Ed., Academic Press, 2019, pp. 1–194, ISBN: 978-0-08-102267-2. DOI: <https://doi.org/10.1016/B978-0-08-102267-2.00001-4>. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/B9780081022672000014>.
- [42] IBM, *Message brokers*, <https://www.ibm.com/cloud/learn/message-brokers>, [Online; accessed 20-May-2020], 2020.
- [43] *What can RabbitMQ do for you?*, <https://www.rabbitmq.com/features.html>, [Online; accessed 10-May-2020].
- [44] *Advanced Message Queuing Protocol, Protocol Specification*, <https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf>, [Online; accessed 10-May-2020].
- [45] RabbitMQ, *Publish/subscribe*, <https://www.rabbitmq.com/tutorials/tutorial-three-python.html>, [Online; accessed 23-May-2020].
- [46] —, *Remote procedure call (rpc)*, <https://www.rabbitmq.com/tutorials/tutorial-six-python.html>, [Online; accessed 1-June-2020].
- [47] P. Dobbelaere and K. S. Esmaili, “Industry paper: Kafka versus rabbitmq”, *ArXiv*, 2017. DOI: 10.1145/3093742.3093908.
- [48] R. Light, “Mosquitto: Server and client implementation of the mqtt protocol”, *The Journal of Open Source Software*, vol. 2, May 2017. DOI: 10.21105/joss.00265.
- [49] D. Soni and A. Makwana, “A survey on mqtt: A protocol of internet of things(iot)”, Apr. 2017.
- [50] N. Q. Uy and V. H. Nam, “A comparison of amqp and mqtt protocols for internet of things”, in *2019 6th NAFOSTED Conference on Information and Computer Science (NICS)*, 2019, pp. 292–297.

- [51] M. J. Sax, “Apache kafka”, in *Encyclopedia of Big Data Technologies*, S. Sakr and A. Zomaya, Eds. Cham: Springer International Publishing, 2018, pp. 1–8, ISBN: 978-3-319-63962-8. DOI: 10.1007/978-3-319-63962-8_196-1. [Online]. Available: https://doi.org/10.1007/978-3-319-63962-8_196-1.
- [52] T. Taleb, A. Ksentini, and B. Sericola, “On service resilience in cloud-native 5g mobile systems”, *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 483–496, 2016.
- [53] Eurecom, *EPC User’s Guide*, Jul. 2016.
- [54] L. Ariza, *How to run oaisim with multiple ue*, <https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/how-to-run-oaisim-with-multiple-ue>, [Online; accessed 21-May-2020], 2018.
- [55] R. Defosseux, *Open Air Kernel Main Setup*, <https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/OpenAirKernelMainSetup>, [Online; accessed 21-May-2020].
- [56] R. Gupta, *How to connect oaisim with OAI EPC*, <https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/T/howtoconnectoaisimwithoaiiepc>, [Online].

Appendix-A: oai-mme/oai-spgw source code modifications

A.1 BROKER INIT CONNECTION CONFIGURATIONS

```
typedef struct{
    uint32_t      port;
    char const    *address;
    char const    *exchange;
    char const    *bindingkey;
}publisher_init_t;
```

A.2 BROKER INFORMATION STRUCTURE

```
typedef struct{
    uint8_t      msgBuf [MAX_MSG_LEN] ;
    uint16_t     msgLen;
    uint32_t     peerIP;
    uint32_t     peerPort;
    uint32_t     teid;
}publisher_send_msg_t;
```

A.3 CHANGES ON NWGTPV2CCREATEANDSENDMSG FUNCTION

```
static NwRcT nwGtpv2cCreateAndSendMsg (...) {
    NwRcT      rc = NW_OK;
    uint8_t    *msgHdr = NULL;

    // Allocate memory to new structure
    publisher_send_msg_t  publisher_send_req__;
    publisher_send_msg_t  *publisher_send_req = &publisher_send_req__;
    MessageDef            *message_p;
    /*...*/

    //Fill struct with the proper arguments
    message_p = itti_alloc_new_message(TASK_S11, publisher_SEND_MSG);
```

```

publisher_send_req = &message_p->ittiMsg.publisher_send_msg;
memcpy(publisher_send_req->msgBuf, pMsg->msgBuf, sizeof(pMsg->msgBuf));
publisher_send_req->msgLen = pMsg->msgLen;
publisher_send_req->peerIP = peerIp;
publisher_send_req->peerPort = peerPort;

//send message to the broker task
rc = itti_send_msg_to_task(TASK_PUBLISHER, INSTANCE_DEFAULT, message_p);
return rc;
}

```

A.4 INIT BROKER CONNECTION IN S11_MME_TASK.C

```

static int init_publisher(void)
{
    MessageDef *message_p = itti_alloc_new_message(TASK_S11, PUBLISHER_INIT);
    message_p->ittiMsg.publisher_init.port = 5672;
    message_p->ittiMsg.publisher_init.address = "192.168.85.225";
    message_p->ittiMsg.publisher_init.exchange = "amq.direct";
    message_p->ittiMsg.publisher_init.bindingkey = "mmespgw";
    return itti_send_msg_to_task(TASK_PUBLISHER, INSTANCE_DEFAULT, message_p);
}

```

A.5 INIT PUBLISHER

```

static void * publisher_intertask_interface (void *args_p)
{
    /*...*/
    MessageDef *received_message_p = NULL;
    itti_receive_msg(TASK_PUBLISHER, &received_message_p);
    //wait for messages in task TASK_PUBLISHER
    while(1){
        if(received_message_p != NULL){
            switch(ITTI_MSG_ID(received_message_p))
            {
                //wait for receive init message
                case PUBLISHER_INIT:
                {
                    publisher_init_t *publisher_init_p =
                        &received_message_p->ittiMsg.publisher_init;
                    publisher_connection(publisher_init_p->address,
                        publisher_init_p->port, publisher_init_p->exchange,
                        publisher_init_p->bindingkey);
                }
            }
        }
    }
}

```

```

        /*...*/
    }
}

int publisher_init(void)
{
    if(itti_create_task(TASK_PUBLISHER, &publisher_intertask_interface,
        NULL) < 0){
        printf("ERROR, consumer pthread_create (%s)\n", strerror(errno));
        return -1;
    }
    return 0;
}

```


Appendix-B: RabbitMQ implementation

B.1 PUBLISHER AND SUBSCRIBER METHODS

B.1.1 Establishment the connection

```
amqp_socket_t *socket = NULL;
//Establish a channel that is used to connect RabbitMQ server
amqp_connection_state_t conn = amqp_new_connection();

socket = amqp_tcp_socket_new(conn);

status = amqp_socket_open(socket, address, port);

amqp_login(conn, "/", 0, 131072, 0, AMQP_SASL_METHOD_PLAIN, "admin", "admin");
amqp_channel_open(conn, 1);
```

B.1.2 Private reply queue

```
amqp_bytes_t reply_to_queue;
amqp_queue_declare_ok_t *r = amqp_queue_declare(conn, 1,
    amqp_empty_bytes, 0, 0, 0, 1, amqp_empty_table);
reply_to_queue = amqp_bytes_malloc_dup(r->queue);
```

B.1.3 Publish the message

```
//(...)
amqp_bytes_t publisher_send;
switch(ITTMSG_ID(received_message_p))
{
    case PUBLISHER_SEND_MSG:
    {
        publisher_send_msg_t* publisher_send_msg_p =
            &received_message_p->ittiMsg.publisher_send_msg;
        publisher_send_msg_spgw = publisher_send_msg_p;
        publisher_send = structSize(publisher_send_msg_spgw);
    }
}
```

```

    break;
}
//(...)
amqp_basic_publish(conn, 1, amqp_cstring_bytes(exchange),
    amqp_cstring_bytes(bindingkey), 0, 0, &props, producer_send)

```

B.1.4 Subscriber method

```

//(...)
amqp_basic_consume(conn, 1, reply_to_queue, amqp_empty_bytes, 0, 1, 0,
    amqp_empty_table);

//(...)
publisher_received_msg_t *publisher_rcv_msg =
    (publisher_received_msg_t *)frame.payload.body_fragment.bytes;
message_p = itti_alloc_new_message (TASK_PUBLISHER, PUBLISHER_PROCESS_MSG);
//(...)
ret = itti_send_msg_to_task (TASK_S11, INSTANCE_DEFAULT, message_p);

```

Appendix-C: Mosquitto implementation

C.1 SIMILAR METHODS FOR PUBLISHER AND SUBSCRIBER

C.1.1 Create the client

```
MQTTClient client;  
MQTTClient_create(&client, address, clientID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
```

C.1.2 Establishment the connection

```
MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;  
MQTTClient_connect(client, &conn_opts))
```

C.2 PUBLISH THE MESSAGE

```
MQTTClient_message pubmsg = MQTTClient_message_initializer;  
MQTTClient_deliveryToken token;  
//(...)  
while(1){  
    //(...)  
    if(){  
        switch{  
            case PUBLISHER_SEND_MSG:  
            {  
                publisher_send_msg_t *publisher_send_msg_p =  
                    &send_message_p->ittiMsg.publisher_send_msg;  
                publisher_send_msg_to_spgw = publisher_send_msg_p;  
                pubmsg.payload = (void*)publisher_send_msg_to_spgw;  
                pubmsg.payloadlen = sizeof(publisher_send_msg_t);  
                pubmsg.qos = QOS;  
                pubmsg.retained = 0;  
            }  
            //(...)  
        }  
    }  
}
```

```

MQTTClient_publishMessage(client, topic, &pubmsg, &token);
rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
}

```

C.3 SUBSCRIBE THE MESSAGE

```

message_received(void *context, char *topicName, int topicLen,
MQTTClient_message *message)
{
    MessageDef *message_p;
    subscriber_rcv_msg_t *subscriber_rcv_msg_p;

    subscriber_rcv_msg_t *subscriber_msg =
        (subscriber_rcv_msg_t *)message->payload;
    message_p = itti_alloc_new_message(TASK_SUBSCRIBER, SUBSCRIBER_RCV_MSG);
    subscriber_rcv_msg_p = &message_p->ittiMsg.subscriber_rcv_msg;

    itti_send_msg_to_task (TASK_S11, INSTANCE_DEFAULT, message_p);

    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
}
//(...)
MQTTClient_subscribe(client, topic, QOS);

MQTTClient_setCallbacks(client, NULL, connlost, message_received, delivered);

```

Appendix-D: Kafka implementation

D.1 CREATE THE CLIENT

```
rd_kafka_conf_t *conf;
conf = rd_kafka_conf_new();

rd_kafka_conf_set(conf, "bootstrap.servers", address, errstr, sizeof(errstr))
```

D.2 PUBLISH THE MESSAGE

```
rd_kafka_t *rk;

rk = rd_kafka_new(RD_KAFKA_PRODUCER, conf, errstr, sizeof(errstr));
err = rd_kafka_producev(rk, RD_KAFKA_V_TOPIC(topic),
                        RD_KAFKA_V_MSGFLAGS(RD_KAFKA_MSG_F_COPY),
                        RD_KAFKA_V_VALUE(buf, len), RD_KAFKA_V_OPAQUE(NULL), RD_KAFKA_V_END);
if(err){
    if (err == RD_KAFKA_RESP_ERR__QUEUE_FULL) {
        rd_kafka_poll(rk, 1000/*block for max 1000ms*/);
        goto retry;
    }
}
```

D.3 SUBSCRIBE THE MESSAGE

```
rd_kafka_t *rk;
rd_kafka_topic_partition_list_t *subscription;

rk = rd_kafka_new(RD_KAFKA_CONSUMER, conf, errstr, sizeof(errstr));
conf = NULL;
rd_kafka_poll_set_consumer(rk);

subscription = rd_kafka_topic_partition_list_new(topic_cnt);
err = rd_kafka_subscribe(rk, subscription);
//(...)
rd_kafka_message_t *rkmsg;
```

```
rkm = rd_kafka_consumer_poll(rk, 100);  
decode_received_message(rkm);
```

Appendix-E: Extraction data from captures

E.1 EXTRACTION S1AP CAPTURES

```
all_messages = defaultdict(dict)
enb_ue_id_imsi_map = dict()
for pkt in capture_slap:
    for i in range(0, len(pkt.get_multiple_layers("slap"))):
        slap_name = pkt.get_multiple_layers("slap")[i].
        get_field_value(pkt.get_multiple_layers("slap")[i].field_names[9])
        if slap_name == "InitialUEMessage":
            enb_ue_id = pkt.get_multiple_layers("slap")[i].enb_ue_slap_id
            enb_ue_id_imsi_map[enb_ue_id] =
                pkt.get_multiple_layers("slap")[i].e212_imsi

            all_messages[pkt.get_multiple_layers("slap")[i].e212_imsi][slap_name] =
                pkt.sniff_timestamp
        elif slap_name == "InitialContextSetupRequest"
        or slap_name == "UECapabilityInfoIndication"
        or slap_name == "InitialContextSetupResponse":
            enb_ue_id = pkt.get_multiple_layers("slap")[i].enb_ue_slap_id
            imsi = enb_ue_id_imsi_map[enb_ue_id]

            all_messages[imsi][slap_name] = pkt.sniff_timestamp
        elif slap_name == "UplinkNASTransport"
            or slap_name == "DownlinkNASTransport":
            slap_name_2 = pkt.get_multiple_layers("slap")[i].
            nas_eps_nas_msg_emm_type.
            showname.split("NAS EPS Mobility Management Message Type: ")[1].
            split(" (")[0]

            enb_ue_id = pkt.get_multiple_layers("slap")[i].enb_ue_slap_id
            imsi = enb_ue_id_imsi_map[enb_ue_id]

            all_messages[imsi][slap_name_2] = pkt.sniff_timestamp
```

```
all_messages[slap_name] = pkt.sniff_timestamp
```

E.2 EXTRACTION GTPV2 CAPTURES

```
all_messages_gtpv2 = defaultdict(dict)
```

```
teid_gre_key_imsi_map = dict()
```

```
for pkt in capture_gtpv2:
```

```
    gtpv2_message = pkt.gtpv2.message_type.showname_value.split(" ")[0]
```

```
    if gtpv2_message == "Create Session Request":
```

```
        teidGre = pkt.gtpv2.f_teid_gre_key
```

```
        teid_gre_key_imsi_map[teidGre] = pkt.gtpv2.e212_imsi[:-1]
```

```
        all_messages_gtpv2[pkt.gtpv2.e212_imsi[:-1]][gtpv2_message]  
            = pkt.sniff_timestamp
```

```
    elif gtpv2_message == "Create Session Response":
```

```
        tunnel = pkt.gtpv2.teid
```

```
        teidGre = pkt.gtpv2.f_teid_gre_key
```

```
        imsi = teid_gre_key_imsi_map[tunnel]
```

```
        teid_gre_key_imsi_map[teidGre] = imsi
```

```
        all_messages_gtpv2[imsi][gtpv2_message] = pkt.sniff_timestamp
```

```
    elif gtpv2_message == "Modify Bearer Request":
```

```
        tunnel = pkt.gtpv2.teid
```

```
        imsi = teid_gre_key_imsi_map[tunnel]
```

```
        all_messages_gtpv2[imsi][gtpv2_message] = pkt.sniff_timestamp
```

```
    elif gtpv2_message == "Modify Bearer Response":
```

```
        tunnel = pkt.gtpv2.teid
```

```
        imsi = teid_gre_key_imsi_map[tunnel]
```

```
        all_messages_gtpv2[imsi][gtpv2_message] = pkt.sniff_timestamp
```

```
    elif gtpv2_message == "Release Access Bearers Request":
```

```
        tunnel = pkt.gtpv2.teid
```

```
        imsi = teid_gre_key_imsi_map[tunnel]
```

```
        all_messages_gtpv2[imsi][gtpv2_message] = pkt.sniff_timestamp
```

```
    elif gtpv2_message == "Release Access Bearers Response":
```

```
        tunnel = pkt.gtpv2.teid
```

```
        imsi = teid_gre_key_imsi_map[tunnel]
```

```
        all_messages_gtpv2[imsi][gtpv2_message] = pkt.sniff_timestamp
```

```
    elif gtpv2_message == "Delete Session Request":
```

```
        teidGre = pkt.gtpv2.f_teid_gre_key
```

```
        imsi = teid_gre_key_imsi_map[teidGre]
```



```

        all_messages_gtpv2[imsi][gtpv2_message] = pkt.sniff_timestamp
    elif gtpv2_message == "Delete Session Response":
        tunnel = pkt.gtpv2.teid
        imsi = teid_gre_key_imsi_map[tunnel]

        all_messages_gtpv2[imsi][gtpv2_message] = pkt.sniff_timestamp
    else:
        assert False

```

E.3 EXTRACTION DIAMETER CAPTURES

```

for pkt in capture_hss:
    hss_message = pkt.diameter.cmd_code.
        showname.split("Command Code: ")[-1].split(" ")[1]

    if hss_message == "Device-Watchdog" or hss_message == "Capabilities-Exchange":
        continue

    hss_flags = pkt.diameter.flags.showname.split("Flags: ")[1].split(", ")[1]

    if hss_message == "3GPP-Authentication-Information":
        if hss_flags == "Request":
            hbh_auth = pkt.diameter.hopbyhopid
            imsi_auth = pkt.diameter.e212_imsi

            hop_by_hop_imsi_map[hbh_auth] = imsi_auth
            all_messages_hss[imsi_auth]["HSS Authentication Request"] =
                pkt.sniff_timestamp

        elif hss_flags == "Proxyable":
            hbh_auth = pkt.diameter.hopbyhopid
            imsi_auth = hop_by_hop_imsi_map[hbh_auth]

            hop_by_hop_imsi_map.pop(hbh_auth)
            all_messages_hss[imsi_auth]["HSS Authentication Response"] =
                pkt.sniff_timestamp

        else:
            assert False

    elif hss_message == "3GPP-Update-Location":
        if hss_flags == "Request":
            hbh_upd = pkt.diameter.hopbyhopid
            imsi_upd = pkt.diameter.e212_imsi

            hop_by_hop_imsi_map[hbh_upd] = imsi_upd
            all_messages_hss[imsi_upd]["HSS Update Location Request"] =
                pkt.sniff_timestamp

        elif hss_flags == "Proxyable":

```

```
hbh_upd = pkt.diameter.hopbyhopid
imsi_upd = hop_by_hop_imsi_map[hbh_upd]

hop_by_hop_imsi_map.pop(hbh_upd)

all_messages_hss[imsi_upd]["HSS Update Location Response"] =
    pkt.sniff_timestamp
else:
    assert False
```