UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

IMPLEMENTING MECHANISMS TO IMPROVE THE PERFORMANCE OF LTE-WLAN AGGREGATION

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Magí Coscollola Coletes

In partial fulfilment

of the requirements for the degree of

MASTER IN TELECOMMUNICATIONS ENGINEERING

Advisors: Klaus Moessner (University of Surrey) and Oriol Sallent Roig (UPC)

Barcelona, September 2017

Title of the thesis: Implementing mechanisms to improve the performance of LTE-WLAN Aggregation

Author: Magí Coscollola Coletes

Advisors: Klaus Moessner, Oriol Sallent Roig

Abstract:

As the release of 5G gets closer, different technologies appear aiming to make a difference in this next generation of mobile communications. A proposed way to achieve more capacity for end-users in 5G is aggregating Long Term Evolution (LTE) and Wireless Local Area Network (WLAN), and allowing users to connect to both at the same time. There are different technologies that aggregate LTE and WLAN, but in this project we focus only on one: LTE-WLAN Aggregation (LWA).

In this thesis we study how offloading WLAN beacons to LTE can improve its performance due to the higher efficiency that LTE brings. This offloading allows to remove overhead from WiFi and UEs using LWA can achieve higher capacities.

This investigation has been performed using OpenAirInterface, a software tool used to simulate and emulate LTE networks.

Acknowledgements

I would like to thank both my thesis supervisors, Klaus Moessner and Oriol Sallent, for their help and support during the development of the project.

I would also like to thank the people I met at University of Surrey, especially Marcin Filo, Atm Shafiul Alam and Faouzi Bouali for their advice and help during my stay at the 5GIC.

Revision history and approval record

Revision	Date	Purpose
0	15/05/2017	Document creation
1	05/07/2017	Document revision
2	14/08/2017	Document revision
3	21/08/2017	Document revision

Written by:		Reviewed and approved by:	
Date	01/09/2017	Date	04/09/2017
Name	Magí Coscollola Coletes	Name	Oriol Sallent Roig
Position	Project author	Position	Project supervisor

Contents

List of Figures

List of Tables

List of Code Listings

1. Introduction

Mobile communications have become the main technology that people use to communicate with each other. They started only with voice traffic and at this time our smartphones can send and receive any kind of media in seconds. Since the 1990s the world has seen a new generation of digital mobile communications every 10 years approximately. This means that when a new technology is deployed, the next one is about to be released.

Each generation increases the capacity and the features of the network, and this makes users to increase their traffic. This phenomenon ”makes the wheel spin”, so users generate more traffic because the network can support it, and the technology keeps evolving in order to give more capacity to the users because they keep increasing their traffic demand.

There are three ways of increasing the capacity of end-users: increasing the bandwidth, increasing the spectral efficiency and having less users per cell. The first one is achieved by allocating more spectrum for mobile communications. However, as spectrum is a scarce resource and there is a lot of regulatory bodies and regulations that allow (or deny) the use of it, increasing the spectrum allocation is a rather slow and difficult procedure that makes this option very complicated to achieve.

The second way, increasing the spectral efficiency, can be done by increasing the transmission power, using more efficient coding, Multiple In Multiple Out (MIMO) or using better signal processing in order to overcome the noise and increase the number of bps/Hz that the connection can hold.

The last solution, reducing the number of users per cell, can be done by reducing the size of the cells and hence increasing the number of them. In this way, cells are smaller, less users ”fit” into them, and less users have to share the capacity of the cell, leading to more capacity per user.

At this point, the research is focused on the fifth generation of mobile communications (5G), which is expected to be defined by 2020. This new technology will have to face the following challenges [8]:

- Providing 1000 times higher wireless area capacity and more varied service capabilities compared to previous generations.
- Saving up to 90% energy per service provided.
- Reducing the average service creation time cycle from 90 hours to 90 minutes.
- Creating a secure, reliable and dependable Internet with a ”zero perceived” downtime for services provision.
- Facilitating very dense deployments of wireless communication links to connect over 7 trillion wireless devices serving over 7 billion people.
- Enabling advanced user-controlled privacy.
- This new high-performance network will be operated via a scalable management framework enabling fast deployment of novel applications, including sensor-based ones, with reduction of the network management Operating Expenses (OPEX) by at least 20%.

As the release of 5G gets closer, different technologies appear aiming to make a difference in this next generation of mobile communications. In this project we studied a way to achieve more capacity for end-users in 5G by aggregating Long Term Evolution (LTE) and Wireless Local Area Network (WLAN), and allowing users to connect to both at the same time. In this way, dense WLAN scenarios can be deployed, so the number of cells is increased and, as a consequence, the number of users per cell decreases. Also, using WLAN allows us to use the unlicensed parts of the spectrum at 2.4 and 5 GHz, so we can see an increase of capacity due to the reduction of users per cell and the increase of used spectrum. There are different technologies that aggregate LTE and WLAN (e.g. LTE WLAN integration with IPsec tunnel (LWIP) and LTE-WLAN Aggregation (LWA)), in this project we are going to focus only on one: LWA, which has a very good performance and it is very simple to deploy and operate.

This project has been carried out at the 5G Innovation Centre (5GIC) of the University of Surrey in Guildford (UK) from February to July 2017 thanks to an Erasmus+ Traineeship agreement. It focuses on creating a new System Information Block (SIB) in LTE and then studying how the offloading of WLAN's beacon data to the new SIB can help to increase the system capacity.

Another important part of this thesis is working with OpenAirInterface (OAI). This software is a LTE network simulator and emulator and it is expected to become widely used by the research community over the next years. At the moment, though, it is still under development and its installation and operation is not straightforward. As you will see, we found several bugs and problems while working with it. However, it is a very complete and promising software. Alongside OAI, another system-level simulator was used in order to simulate Wi-Fi scenarios. This simulator is called ns-3 [18] and it provided interesting results when modifying beacons' configuration in IEEE 802.11.

1.1. Objectives

This project focuses on improving the performance of LWA, a technology that we may see when 5G gets deployed (or even before). LWA has potential to make a difference in the users' quality of experience in indoor and dense scenarios, so improving its performance would make it even better. The main goal of this project is proposing and testing an improvement for this kind of LTE-WLAN interworking technology.

Another main topic was working with OAI and getting familiar to it in order to work with and help other researchers in the centre work with it.

In order to become familiar with OAI, the first half of the stay in the 5GIC was dedicated to its installation, configuration and operation. After that, the next objective was adding a new SIB in OAI in order to perform tests with and without this message and see if it generated any problems to the normal operation of a standard LTE cell.

The last objective was proving that the use we thought for this new SIB, offloading beacon information in order to make WLANs more efficient, was actually worth it. In order to do so we would have to work with another simulator, ns-3, for simulating Wi-Fi environments (as OAI does not support WiFi), together with OAI for simulating LTE scenarios.

1.2. Thesis outline

This thesis is divided in the following parts:

- **State of the art**. This chapter explains the technologies involved in the development of this thesis, LTE and IEEE 802.11, focusing on the aspects that are the most important for understanding the project. It also includes a description of the used simulation software, OAI and ns-3.
- **Project development**. It shows the different parts of the development of the project, mainly divided in three parts: Installation of OAI, modifying OAI to implement a new SIB and running Wi-Fi simulations with ns-3.
- **Results**. This chapter exposes the results obtained from the simulations.
- **Budget**. A small budget counting the resources used in the development of the project.
- **Conclusions and future development**. Last chapter, which discusses about the results obtained and sets the next steps that should be performed to carry on with this project.

At the end there are also several appendices, which are documents that may be useful for the reader in order to have extra information about what is introduced in the thesis.

1.3. Work plan

Here we are going to present the work plan of this project. It started in February 2017 and ended in August of the same year. As we said before, the first half of the time was dedicated to installing and configuring OAI, and the second half to adding the new SIB and making simulations.

We can see the final work plan in Figure 1.1 and Figure 1.2. This plan, however, suffered modifications during the development of the project. The first and most important one was that, due to the lack of updated documentation, the installation of OAI, especially the Core Network, took much more time than expected. The first idea was that it would be just following a tutorial, but the reality was that it became a trial-and-error procedure where we had to start over several times. This delayed the implementation of the new SIB several weeks.

Another problem we found was that, due to the lack of reliability of OAI, when trying to simulate an LTE cell with OAI in order to test how much adding a new SIB would affect to the throughput, there was some problem with the software that did not let the simulations perform correctly. We tried to solve it, but we ran out of time because we detected the problem too late, in July. This did not allow us to simulate the performance of LTE before and after adding the new SIB.

Even though we had these shortcomings and some other minor ones, we could install and run OAI and help other researchers do the same. The objectives of learning how to use OAI and adding a new SIB to it were accomplished and also testing the impact of beacons in Wi-Fi.

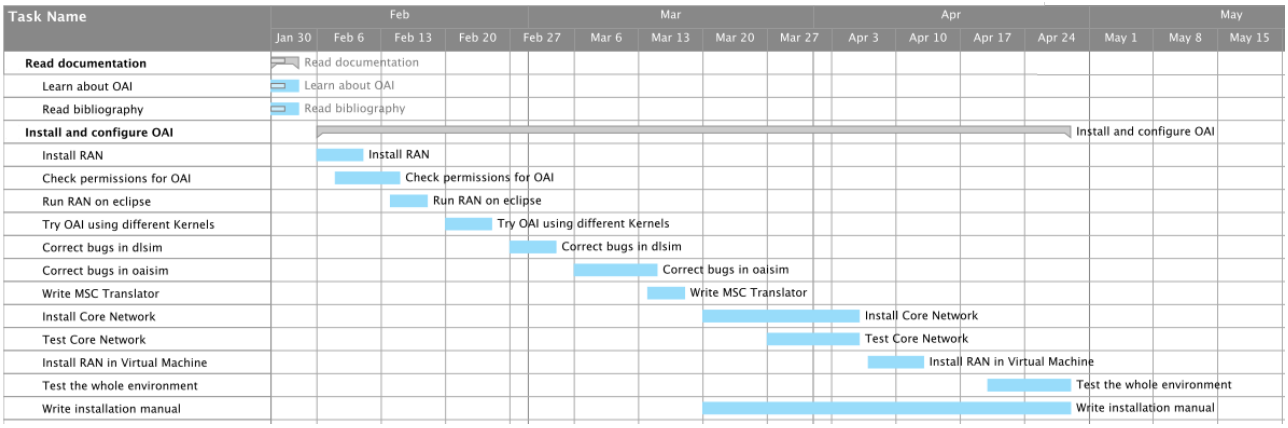


Figure 1.1: Gantt diagram of the work plan. (1)

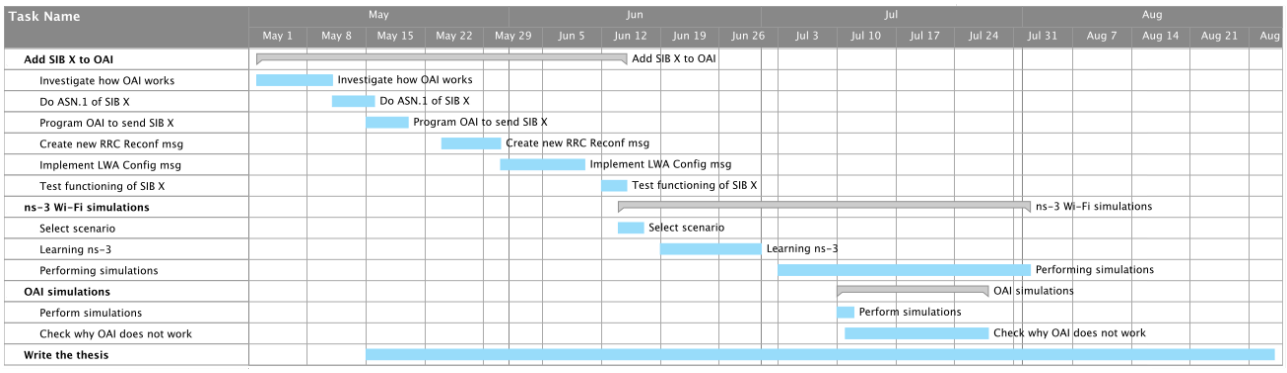


Figure 1.2: Gantt diagram of the work plan. (2)

2. State of the art

In this chapter we focus on the technologies involved in our project. As we are working with two different wireless technologies (LTE and IEEE 802.11), we are presenting them separately and then showing how the interworking between both will work. We also introduce the two simulators used for the development of the project.

2.1. LTE

LTE is the technology used in 4G, even though the first versions of it did not match the requirements that the International Telecommunication Union (ITU) outlined. In this way, LTE is considered *3.9G* and LTE Advanced (LTE-A) has been the version that matches the requirements for the fourth generation.

The main changes that this technology introduced compared to its predecessors, Universal Mobile Telecommunications System (UMTS) for voice and High-Speed Packet Access (HSPA) for data, was a simplification of the network architecture achieved thanks to moving the voice traffic to the packet domain with the introduction of Voice over ISM (VoIP). In addition to this, the new technology also introduced Orthogonal Frequency Division Multiple Access (OFDMA) and other physical layer improvements such as carrier aggregation or MIMO technology in order to obtain the best throughput.

Another important concept that gained momentum with the introduction of LTE was heterogeneous networks, which present scenarios where small cells are placed inside a bigger cell in order to work together to provide more capacity to the end-users. These small cells could implement LTE as well or they could implement some IEEE 802.11 standard. It is in this point where aggregation between LTE and Wi-Fi appeared, and we discuss this topic later in this chapter.

In addition to the ”normal” data and voice traffic, LTE also aims to provide support for other kinds of traffic that may be required in particular conditions or in the future, such as device-to-device communications, Multimedia Broadcast Multicast Service (MBMS) for broadcasting highly demanded content, Internet of Things, public safety communications, etc.

Even though LTE-A is the technology used for 4G, it seems that 5G will keep using it adding, of course, improvements in order to meet the requirements of IMT-2020, stated by the ITU and the minimum requirements for a technology to be considered of the fifth generation of mobile communications. One of the improvements that LTE may have in order to be also the standard of 5G may be LWA, as we said in the introduction.

The next subsections will present LTE features that need to be known in order to fully understand this thesis.

2.1.1. System Information Blocks in LTE

SIBs are implemented in mobile communications in order to broadcast information about the configuration of the network. System Information (SI) includes common information

(broadcasted to all User Equipments (UEs)) that comes from superior layers -Non Access Stratum (NAS). This information is applicable when the UE is on ”idle” mode.

SIBs are transmitted by means of the Radio Resource Control (RRC) protocol, which is the one in charge of controlling the behaviour of the UEs when it is on ”connected” mode and sends paging messages and broadcasts SIBs to the UEs that are on ”idle” mode [2].

This system is formed by the different kinds of SIB listed below [1] [2]:

- **Master Information Block (MIB).** Transmitted every 40 ms using the Physical Broadcast Channel (PBCH) and appears on the subframe number 0. It transmits fundamental parameters of the network to allow the initial access of the UEs, e.g. used channels, number of transmitting antennas on the Evolved Node B (eNB) side, the frame number, etc.
- **SIB Type 1.** Transmitted every 80 ms using the Physical Downlink Shared Channel (PDSCH) and appears in the subframe number 5. This SIB transmits information related to the access to the cell and resource assignment information, e.g. Public Land Mobile Network (PLMN) identity, location area, cell identity, scheduling of the other SIBs, etc.
- **SIB Type 2.** Broadcasted with a periodicity of 160 ms and used to inform the UEs about the configuration of the common and shared channels, random access parameters, bandwidth, uplink (UL) power control parameters, etc.
- **SIB Type 3.** Transmitted with a periodicity of 320 ms and used to inform the UEs about common parameters related to the cell reselection mechanisms.
- **SIB Type 4.** Transmitted every 320 ms and used to inform the UEs about parameters related to the configuration of the neighbouring cells that use the same sub-carriers (*intra-frequency cell reselection information*). This information is useful for future handover measurements.
- **SIB Type 5.** Has a periodicity of 640 ms. It is also used to inform the UEs about neighbouring cells, but in this case it informs about the neighbours that use different sub-carriers (*inter-frequency cell reselection information*). Again, this information is used so the UEs can perform measurements to support handover procedures.
- **SIB Type 6.** Transmitted every 640 ms and used to give support to the mechanisms that perform cell reselection to UMTS Terrestrial Radio Access Network (UTRAN) cells.
- **SIB Type 7.** Transmitted every 640 ms and used to give support to the mechanisms that perform cell reselection to GSM EDGE Radio Access Network (GERAN) cells.
- **SIB Type 8.** Broadcasted with a periodicity of 640 ms and used to give support to the mechanisms that perform cell reselection to CDMA2000 cells.
- **SIB Type 9.** Has a periodicity of 640 ms and contains the eNB identifier, which has a maximum size of 48 B. It allows the UEs to manually select the eNB they want to connect to.
- **SIB Type 10.** Contains Earthquake and Tsunami Warning System (ETWS) notifications.

- **SIB Type 11.** Contains complementary ETWS notifications.
- **SIB Type 12.** Contains Commercial Mobile Alert System (CMAS) notifications.
- **SIB Type 13.** Contains the information required to acquire the MBMS control information associated with one or more Multicast-Broadcast Single Frequency Network (MBSFN) areas.
- **SIB Type 14.** Contains the Extended Access Barring (EAB) parameters.
- **SIB Type 15.** Contains the MBMS Service Area Identifier (SAI) of the current and/or neighbouring carrier frequencies.
- **SIB Type 16.** Contains information related to GPS time and Coordinated Universal Time (UTC). The UE may use these parameters to obtain the UTC, the GPS and the local time.
- **SIB Type 17.** Contains information relevant for traffic steering between Evolved UTRAN (E-UTRAN) and WLAN.
- **SIB Type 18.** Indicates that E-UTRAN supports the sidelink UE information procedure and may contain resource configuration information related to sidelink communication.
- **SIB Type 19.** Indicates that E-UTRAN supports the sidelink UE information procedure and may contain resource configuration information related to sidelink discovery.
- **SIB Type 20.** Contains the information required to acquire the control information associated transmission of MBMS using Single Cell Point To Multipoint (SC-PTM).
- **SIB Type 21.** Contains Vehicle-To-Everything (V2X) sidelink communication configuration.

In LTE, only MIB, SIB1 and SIB2 provide the minimum required information in order to allow UEs to connect to the network, so it is mandatory that they are transmitted in any cell [19].

The different SIBs are mapped to the SI messages, which are transmitted using Physical Resource Blocks (PRBs) of the PDSCH. SIB1 is always mapped to SI1 and SIB2 is also always mapped to SI2. The remaining SIBs can be multiplexed between themselves if they have the same periodicity. As an example, SIB3, SIB4 and SIB5 could be multiplexed to only one SI with a periodicity of 320 ms.

2.1.1.1. Notification of changes in SIBs

Even though SIBs' information is quite static, it may suffer changes over time because the configuration of the eNB or the neighbouring eNBs has changed. For example, if a new SIB is started to broadcast the SIB mapping field in SIB1 will change, so it will be necessary to notify the existing UEs that are connected to the eNB that there has been a modification in a SIB. These changes need to be notified to the UEs of the network so they can update their knowledge about the cell. If the UEs are not notified about a change, they will automatically update their SIB information after 3h of obtaining it, as they will consider the old information invalid.

There are a couple of mechanisms in order to notify of this information update: the *systemInfoValueTag* in SIB1 and the *systemInfoModification* field in the paging messages.

systemInfoValueTag is a field in SIB1 that tells if changes have been done in other SIBs. Every time a change is made this tag is increased by one. UEs may use this tag to verify if the previously stored SI messages are still valid, e.g. when returning from out of coverage.

This tag, however, may not be always updated. E-UTRAN may not update it after changing some system information (e.g. ETWS information), regularly changing parameters like time information or EAB parameters [3].

Paging messages are used to inform UEs in *RRC_IDLE* and *RRC_CONNECTED* mode about a system information change. If the UE receives a paging message including the *systemInfoModification* it knows that the system information will change at the next modification period boundary. This message is used to tell UEs that there are going to be changes in the SI, but it does not provide further details, so the UE does not know which system information will change.

Modification periods are defined by the scheduling of system information, and SI may be transmitted several times with the same content within a modification period. The boundaries of modification periods are defined by System Frame Number (SFN) values for which SFN mod m = 0, where m is the number of radio frames comprising the modification period.

The modification period is transmitted by SIB2 using the fields *modificationPeriodCoeff* and *defaultPagingCycle*. The length of a modification period corresponds to *modificationPeriodCoeff* times *defaultPagingCycle* frames.

2.1.2. UE attachment and DRB assignment in LTE

It is necessary to explain this procedure because it was used as a trigger in OAI for sending a message that told the UEs to listen to our newly created SIB -SIBX- or not.

We are going to look into the case where a UE is turned on and attempts to attach to a network for the first time after subscribing to the LTE network, but, in order to simplify, we are only going to focus on the last part of this procedure (step 5 in Figure 2.1), where the Dedicated Radio Bearer (DRB) assignment is performed.

In the first step of the Initial Attach procedure the Mobility Management Entity (MME) obtains an International Mobile Subscriber Identity (IMSI) from the UE. The UE attempts to initially attach to the network by sending an Attach Request message containing its IMSI, so the MME obtains it from this message.

After collecting the UE's IMSI and security capability information, from the Attach Request message received from the UE, the MME performs the authentication and NAS security setup procedures (2 and 3 in Figure 2.1).

The authentication procedure consists of two steps: authentication vector acquisition, where the MME requires authentication vectors from the Home Subscriber Server (HSS) for the UE, and mutual authentication, during which the MME and the UE are mutually authenticated.

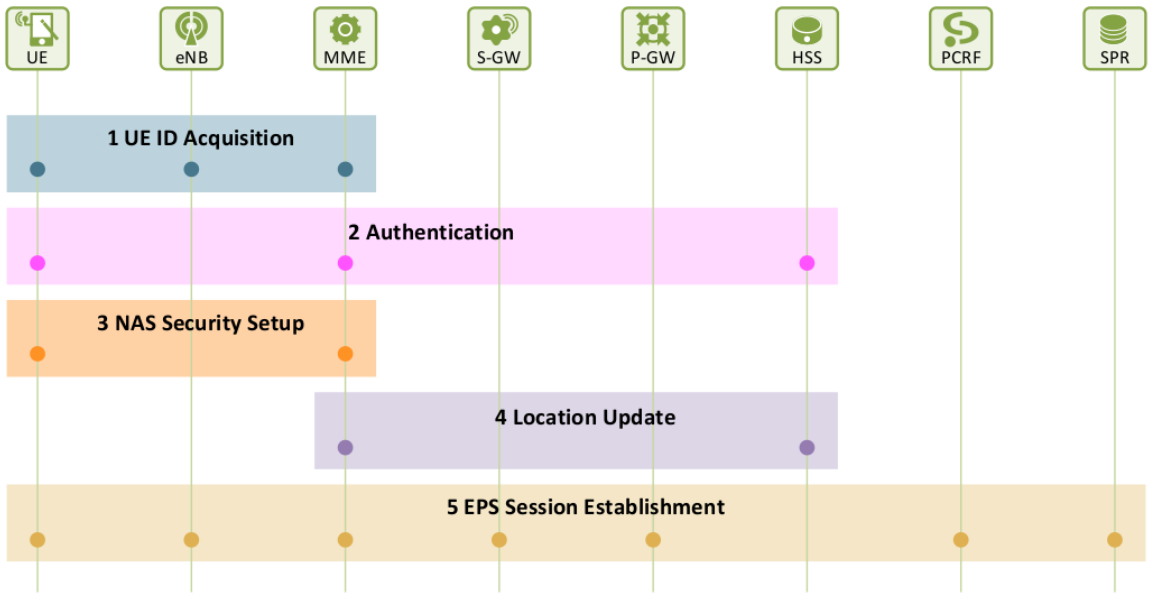


Figure 2.1: Summary of Initial Attach procedures [10].

Once the UE and MME are mutually authenticated, the procedure for establishing NAS security begins. The MME initiates the NAS security setup procedure so that NAS messages can be securely exchanged between itself and the UE.

Once the procedures for authentication and NAS security setup are completed, the MME has to register the subscriber in the network and find out what service are available for them. Here, the MME notifies the HSS that the subscriber is registered in the network and located in its Tracking Areas (TAs) and then downloads information about the subscriber from the HSS. This is done through the location update procedure (Step 4 in Figure 2.1).

After that, the MME establishes an Evolved Packet System (EPS) session and a default EPS bearer for the user based on the subscription information. By doing so, the MME allocates the resources for providing each user with a Quality of Service (QoS) satisfying what they are subscribed to. We can see this detailed procedure in Figures 2.2 and 2.3.

As we can see in Figure 2.2, the MME first selects an EPS Bearer ID (a number between 5 and 15) in order to establish a default EPS bearer for the newly attached user. After that, the MME selects the PDN Gateway (P-GW) based on the subscription information received from the HSS.

Next, a ”create Evolved Packet Core (EPC) session request” is sent by the MME and it is established once the MME has received a ”create EPC session response”. With this response, the P-GW informs the MME that the resources had been approved and allocated to the user and also the QoS information applied to the established EPS sessions and default EPS bearer.

Once this procedure is finished with the creation of the UL S1 GTP-U tunnel, the MME returns an ”Attach Accept” message to the UE as a response to the ”Attach Request” it had sent before (seen in Figure 2.3), so at this point the UE is attached to the network.

At this point the MME creates the Access Stratum (AS) security base key, K_{eNB}, which the

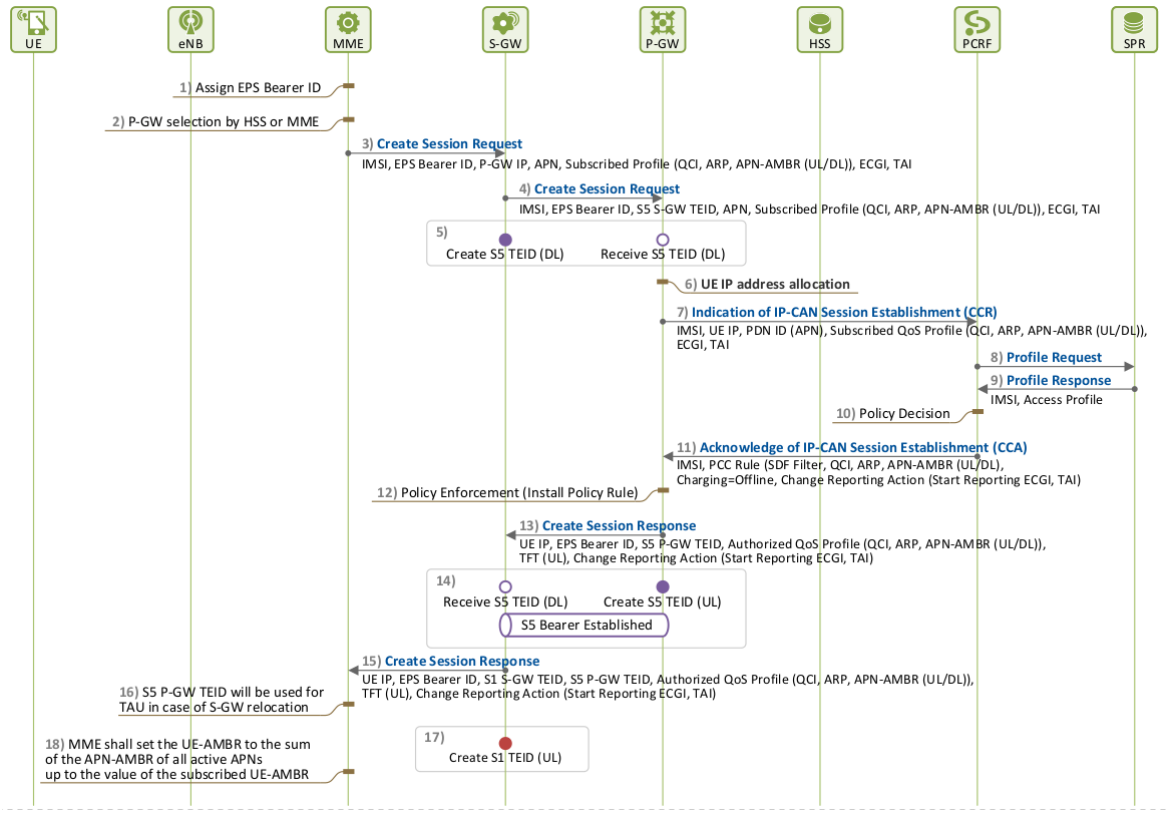


Figure 2.2: Procedure for EPS Session Establishment (1) [10].

AS security base key and is included in the ”Initial Context Setup Request” message that is sent afterwards to the eNB so that it can establish a S1 bearer with the Serving Gateway (S-GW) and a DRB with the UE.

Next, the AS security is set up. The eNB generates AS security keys from K_{eNB} received from the MME and selects the ciphering and integrity algorithms for RRC messages and user traffic between itself and the UE. The eNB also helps the UE to generate AS security keys by informing it about the chosen AS security algorithms through a ”Security Mode Command” message.

When the AS keys are completely generated and ready to work the UE indicates it to the eNB sending a ”Security Mode Complete” message. As the AS security setup is now completed, RRC messages over the radio link are now sent as encrypted and integrity-protected, and user traffic is delivered as encrypted. Now, the DRB establishment can begin.

The DRB establishment begins with a ”RRC Connection Reconfiguration” message from the eNB to the UE. The RRC connection was already established when the UE sent the ”Attach Request” message. However, it must be reconfigured now that the UE needs to configure parameters according to the resources allocated by the network as a result of permission to access it. The RRC layer of the UE allocates radio resources based on the configuration parameters gathered from the ”RRC Connection Reconfiguration” message and it replies with a ”RRC Connection Reconfiguration Complete” message.

Now the DRB is established and the UE and the eNB stay in EMM-Registered state and now

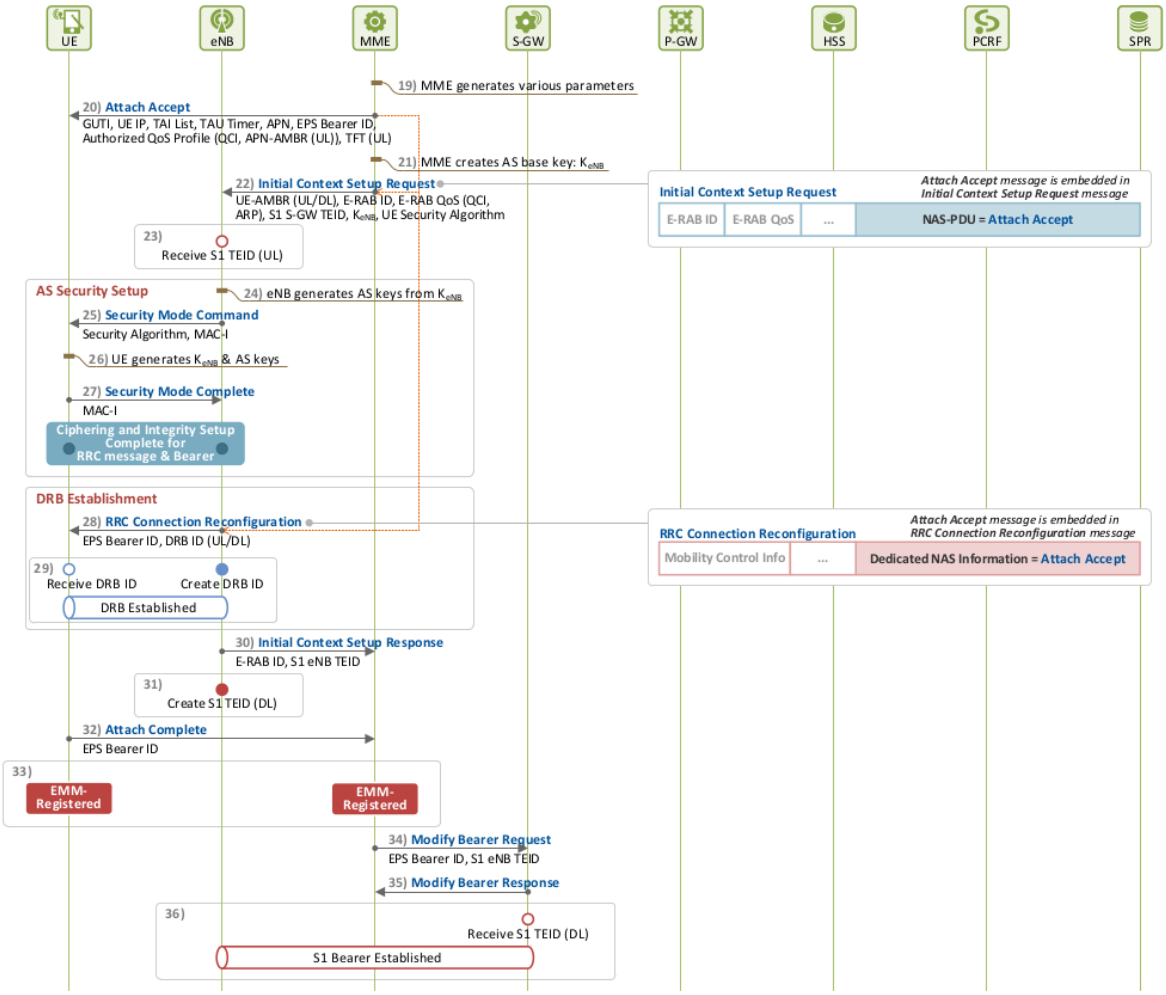


Figure 2.3: Procedure for EPS Session Establishment (2) [10].

the remaining step is establishing the S1 bearer between the eNB and the S-GW.

2.2. IEEE 802.11

IEEE 802.11 standard, mainly known as Wi-Fi, became the de facto standard for wireless connectivity for computers and other devices several years ago. WLANs were, at first, only used by computers and laptops, but now they are also massively used by smartphones and other ”smart” devices.

There are several standards under IEEE 802.11, each with a different letter suffix. These standards cover everything from the wireless standards themselves to standards for security aspects or QoS and some of them are listed below [16]:

- **802.11a** defines a wireless network bearer operating in the 5 GHz Industrial, Scientific and Medical (ISM) band with a data rate up to 54 Mbps.
- **802.11b** defines a wireless network bearer operating in the 2.4 GHz ISM band with data rates up to 11 Mbps.

- **802.11e** provides Media Access Control (MAC) enhancements for the prioritisation of traffic classes through the modification of MAC parameters [7].
- **802.11f** standardises the handover procedures.
- **802.11g** defines a wireless network bearer operating in the 2.4 GHz ISM band with data rates up to 54 Mbps.
- **802.11h** includes spectrum and transmit power management extensions to solve interference problems in the 5 GHz ISM band [7].
- **802.11i** contains security enhancements such as new encryption methods [7].
- **802.11j** defines interworking.
- **802.11k** allows Radio Resource Measurements (RRMs) of WLANs in order to facilitate their management and maintenance [7].
- **802.11n** defines a wireless network bearer operating in the 2.4 and 5 GHz ISM bands with data rates up to 600 Mbps.
- **802.11r** presents the definition of authentication and association messages in order to complete fast and secure handoffs between Basic Service Sets (BSSs) [7].
- **802.11s** provides mesh networking by describing the mechanisms to form self-configuring multi-hop networks for broadcast, multicast and unicast data delivery [7].
- **802.11ac** defines a wireless network bearer operating below 6 GHz to provide data rates of at least 1 Gbps for multi-station operation and 500 Mbps on a single link.
- **802.11ad** defines a wireless network bearer providing very high throughput at frequencies up to 60 GHz.
- **802.11af** standardises Wi-Fi in TV spectrum white spaces.
- **802.11ah** standardises Wi-Fi using unlicensed spectrum below 1 GHz to provide long range communications and support for the Internet of Everything.
- **802.11ax** has the aim of become a new amendment to the Wi-Fi standard which increases the transmission rates and improves efficiency of channel usage in case of dense deployment [14].

All the 802.11 Wi-Fi standards operate within the ISM frequency bands. The operation in these unlicensed bands helped this technology to become widespread, but it also presents drawbacks related to the need of being a ”good neighbour” with other devices and technologies that may use the same band at the same time.

IEEE 802.11 organises its communications in three frame types:

- Data frames carry data from station to station.
- Control frames are used alongside with data frames to deliver data in a reliable way.
- Management frames are used to join and leave wireless networks and move associations from access point to access point. Beacons are a type of management frames.

IEEE 802.11 can use two multiple access mechanisms: Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA), also called Distributed Coordination Function (DCF), and

a priority-based access, also called Point Coordinated Function (PCF). The first one makes sure that the channel is free before starting the transmission, while the second one implements a polling mechanism where the access point acts as the coordinator. In practice, CSMA/CA is the mostly used as it does not require centralised coordination.

2.2.1. CSMA/CA

This multiple access mechanism is based on listening to the channel and making sure it is free before starting a transmission. When someone wants to transmit, it first has to listen to the channel. If the channel is free during DCF Interframe Space (DIFS) time it transmits, if the channel is busy it waits until the transmission ends, waits DIFS time and after that it enters in random backoff.

If the medium keeps free during the backoff it transmits, if not, the timer is stopped until the channel has been free for a DIFS time. In this way stations that have been waiting for more time should finish the backoff before the ones that started waiting later. When the timer expires (again if the channel has been free during the wait), it transmits.

Even though this system tries to avoid collisions, they may still happen as the transmitter may not be able to detect another device that is transmitting to the same receiver (hidden terminal problem). In order to ensure that the transmission has been successful, the receiver needs to send an acknowledgement message. If the acknowledgement is not received, the collisioned frame will have to be retransmitted. Before retransmitting, the sender will enter in an exponential random backoff, which starts with a small window that keeps doubling its size when more collisions happen (from a size of 32 to a maximum size of 1024) [7].

2.2.1.1. RTS/CTS

Request To Send (RTS) and Clear To Send (CTS) messages may be optionally used in order to reduce the problem of the hidden terminal in a CSMA/CA system because the Access Point (AP) allows only one station (STA) to transmit at a time.

When a STA wants to transmit it sends a RTS with a time of channel reservation. When the AP receives it, it sends a CTS with the same period of reservation. This last message is received by all the stations that are connected to the AP, so all stations know that the channel will be busy during the solicited time, even if they do not detect activity in the channel.

In this way collisions can only happen in RTS packets, and as they are shorter than data packets they are also less likely to collide.

2.2.2. Beacon frames

Beacon frames are a type of management frame that are being periodically broadcasted by the AP in order to announce the presence of the WLAN and transmit useful information for enabling STAs to establish and maintain communication with them. A complete list with all the possible fields of a beacon frame can be found in Appendix C. Beacon size can vary a lot from one AP to another, but we can consider 300 B as a common size.

Another function of beacons is providing synchronisation to the network. Each station manages its own clock but it must be updated periodically in order to avoid synchronisation mismatches. This is done with the beacon, even though beacon intervals might be slightly irregular because the beacon also needs to follow CSMA/CA. Beacon messages have a timestamp field that STAs read in order to update their local clocks to this time, so all the STAs associated to an AP will have the same local clock value.

Beacons are also used to tell STAs in power save mode that they have packets to be received. These stations only wake up every time the beacon is being transmitted in order to check if there is traffic waiting for them.

APs normally broadcast beacons every 102.4 ms, which is 100 Time Units (TU) (1 TU is 1024 μs), and at the lowest data rate available. The beacon interval can be modified to a different number of TU in most of the commercial APs. Increasing it would lead to less overhead so STAs see more throughput. However, it delays association processes as stations scanning for networks could easily miss a beacon while they are scanning other channels. On the other hand, reducing the beacon interval increases the overhead, so STAs connected to the network would now see less throughput but it makes the process of association faster. Additionally, it also makes that STAs in power saving mode will have a higher consumption as they will have to wake up and listen to the beacon more frequently.

The beacon interval is typically not changed from the default value in the WLAN network installation phase. As a short beacon interval is good for the association but gives a lower throughput, and the opposite for a long beacon interval, this setting will be only good in certain moments and traffic conditions. That is why adaptive beacon interval algorithms have been invented [21]. These algorithms use short beacon intervals when the network has a low amount of connected STAs, so the new ones can detect it fast, and it increases the beacon interval as more users connect to it, so it can increase its throughput.

2.3. LTE-WLAN Aggregation

LWA is a feature of the Release-13 from the 3rd Generation Partnership Program (3GPP) which allows a mobile device to be configured by the network so that it utilises its LTE and Wi-Fi links simultaneously. Unlike other LTE-WLAN interworking methods, LWA has the capability of splitting a single IP flow at a sub-level granularity (i.e. Packet Data Convergence Protocol (PDCP)). This capability allows all applications to use both LTE and WLAN links simultaneously without any application-level enhancements, thus providing significant performance gains [20].

With LWA, the UEs will automatically use operator deployed WLAN without user intervention. If a user wishes to use a non-LWA enabled network, they may connect to the desired network even while a LWA session is ongoing if their UE is capable to connect simultaneously to two WLANs. For cases when the UE does not have this capability, it will disassociate from the LWA-enabled AP and connect to the user selected WLAN while any existing sessions continue, uninterrupted, over the LTE bearer.

The benefits of LWA include, thanks to the extra capacity provided by the WLAN, an increase in the whole system capacity, the ability to increase the number of users who can be served without making major upgrades to the LTE network, just deploying a LWA-enabled

WLAN where needed; and the simplification of WLAN operation within the cellular network.

Legacy LTE-WLAN interworking technologies implied that the operator had to deploy and maintain two separate networks, one for licensed spectrum and other for unlicensed spectrum. LWA, instead, allows the operator to integrate WLAN at the Radio Access Network (RAN) level so it can eliminate costly WLAN specific Core Network (CN) nodes. In this way, it can reduce CN signalling and making WLAN deployment easier and cheaper to maintain. It also allows the operator to deploy WLAN with a sufficient level of network control to ensure efficient network resources usage of both LTE and WLAN [20].

In order to quantise the performance gains, in [20] some simulations have been performed with a scenario using LWA with 20 MHz WLAN and 20 MHz LTE channels, using IEEE 802.11n capabilities and non full buffer File Transfer Protocol (FTP) traffic model 3. The result of these simulations was compared to a LWIP scenario, where there is integration but bearers are not split between LTE and WLAN [6]. The results of these simulations show that LWA performance gains for non-collocated deployments (see subsection 2.3.1) are between 20-30% in average user throughput and between 30-50% for collocated deployments.

As we can see, LWA offers an important increase of the throughput while keeping a simple architecture. It offers a good option for small cell deployment using unlicensed bands and WLAN equipment, which will result cheaper than using LTE base stations.

LWA can also be deployed in places where the Wi-Fi network already exists (i.e. shopping centres), so it can be controlled by eNBs and, in this way, manage the traffic over LTE, WLAN or both depending of the characteristics of each connection and improve the users' quality of experience.

2.3.1. LWA Architecture

LWA follows the Dual Connectivity architecture in LTE [12], which allows the UEs to connect to multiple base stations at the same time. Where, as stated previously, the APs are connected to a WLAN Termination (WT)/Access Controller (AC) device, which is connected to the eNBs via the *Xw* interface as we can see in Figure 2.4.

On the user plane, LTE and WLAN are aggregated at the PDCP level. In downlink (DL), the eNB may schedule PDCP Protocol Data Units (PDUs) of the same bearer to be delivered to the UE either via LTE or WLAN. The eNBs can receive radio information about both links that allow them to perform a more efficient scheduling. We can see a diagram of the splitting in Figure 2.5.

As WLAN MAC layer has to remain unchanged, 3GPP defined a new protocol: LWA Adaptation Protocol (LWAAP). This new protocol is transparent to WLAN. A new EtherType[1] had to also be created in order to allow UEs to differentiate between LWA and non-LWA packets.

When it receives the packets, the UE's PDCP layer performs re-ordering and in this way it allows packet-by-packet scheduling on both LTE and WLAN links.

[1] It is a field of 2 B in an Ethernet frame that is used to indicate which protocol is encapsulated in the payload of that frame.

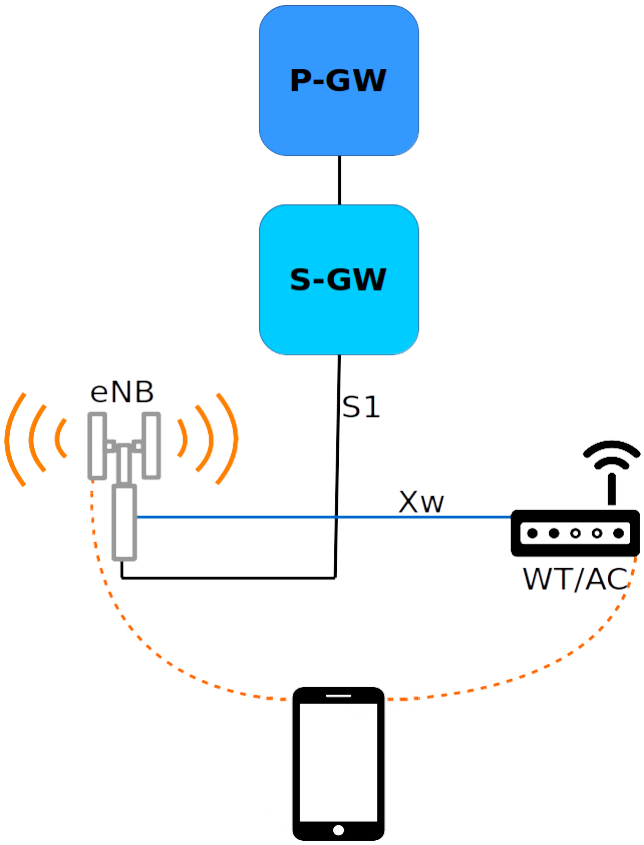


Figure 2.4: LWA network architecture.

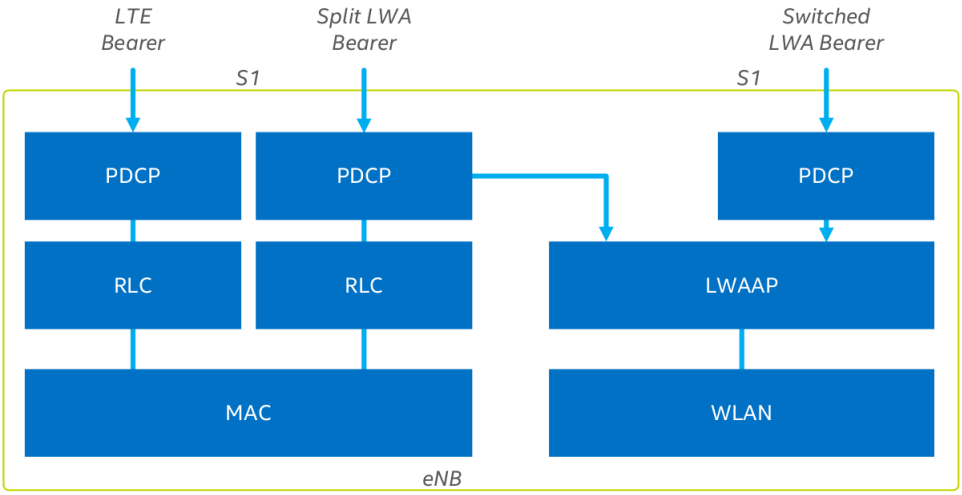


Figure 2.5: LWA user plane architecture. [20]

On the control plane, the eNB is in charge of LWA activation and the decision of which bearers are offloaded to the WLAN. WLAN measurements are used in order to perform these decisions.

Regarding UE mobility between WLAN APs, the eNB configures the UE with a list of APs (called WLAN Mobility Set) within which the UE can move without notifying the network. This allows a fast and transparent selection of the WLAN selection providing the best throughput. The most common configuration would be that all APs in the WLAN Mobility Set are connected to the same eNB. When a UE moves outside this WLAN Mobility Set, the decision of which APs it is allowed to connect to is again made by the eNB.

From the network point of view, there are two options that need to be taken into account when deploying LWA:

- **Collocated**:
  - The eNB and the WLAN AP or AC are integrated in the same device.
  - It is more appropriate for new small cell deployments with integrated WLAN APs.
- **Non-collocated**:
  - The eNB and the WLAN AP or AC are connected via a standardised interface called Xw through a WT logical node.
  - This option is more appropriate for integrating existing WLAN deployments into LWA with minimum changes to the existing infrastructure.

On the UE side, the devices do not need new radio, antennas or RF components as they already support connection to WLANs. This means that it can be easily supported for new equipment and also for existing hardware with just software changes. Another good point of using WLAN for utilising the unlicensed spectrum is that the WLAN device already supports radio resource sharing, spectrum and/or time sharing with non-LWA WLAN and Bluetooth devices in an optimised way.

2.3.2. LWA Configuration message

The LWA Configuration message is a RRC message used to setup, modify and release LWA connections. It can be of two kinds:

- Release, which will make the UE release the LWA connection.
- Setup, which will setup the connection or modify its parameters.

A Release LWA Configuration message only contains the ”release” field, but a setup LWA Configuration message contains the parameters used for WLAN mobility (WLAN Mobility Set) and the parameters used for WLAN authentication.

The full contents of this message can be found in Appendix D.

2.3.3. Discussion

As we have seen, LWA joins the previously presented technologies (LTE and IEEE 802.11) in order to increase the end-user capacity.

Having LWA implemented we could use the aggregation between these two technologies in order to reduce signalling overhead in Wi-Fi, as its media access mechanism (CSMA/CA) works in a way that every message that is sent keeps the channel only for itself during the time of transmission. In a case like this, reducing the number of signalling messages or their length leaves more time for data messages to be sent, and this increases the throughput that users see.

IEEE 802.11 beacons suppose an important overhead to WLANs, while SIBs are an efficient way to broadcast information to the whole network. In this way, SIBs would be useful to reduce overhead of WLANs, and so improving the user experience, while not increasing a lot the amount of broadcast information sent by the eNB. We also have to take into account that LTE is more efficient in terms of broadcasting compared to WLAN, as it has a lot more control of the media access.

Following this, a new SIB could be added in order to broadcast information about the APs in a periodic way using the resources of the LTE network instead of using resources from Wi-Fi.

2.4. OpenAirInterface

OAI is a software platform created by the OpenAirInterface Software Alliance (OSA), which is a non-profit consortium that includes several academic and industrial partners, in order to provide open-source software for 3GPP cellular networks. OAI offers a software-based implementation of LTE covering the full protocol stack of this 3GPP standard both in E-UTRAN and EPC. It can be used to emulate eNBs, UEs and a core network on a PC [4].

It is written in C language and it can run on Linux machines and it is released as free software (anyone can download it from GitHub). However, it is still under development and, so far, it cannot be considered as a reliable tool as it still has missing parts, unsolved

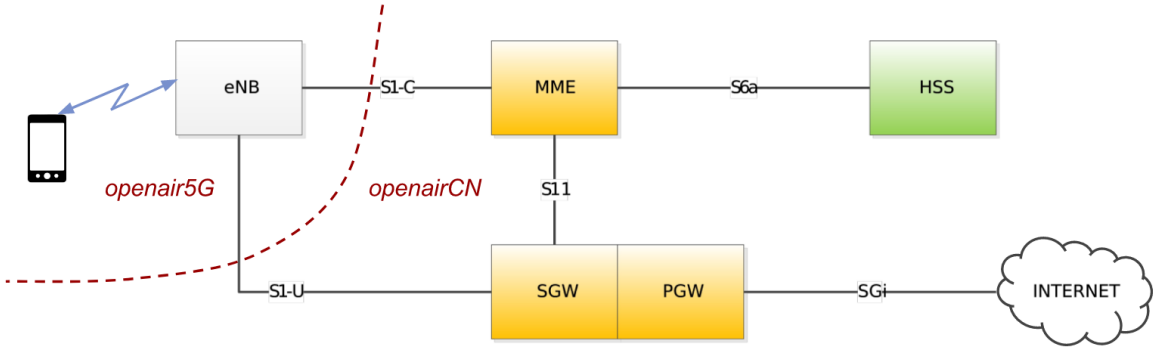


Figure 2.6: OpenAirInterface structure showing the interfaces that connect each module.

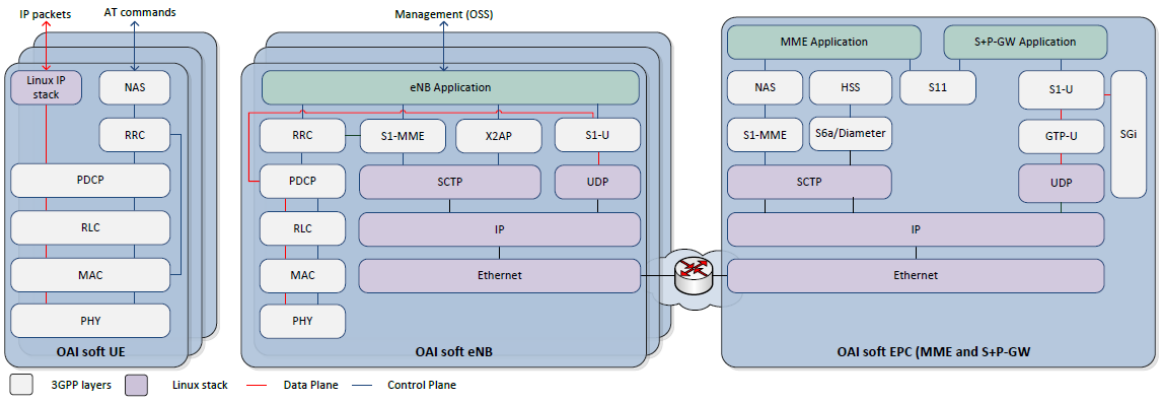


Figure 2.7: OpenAirInterface LTE software stack [4].

bugs and it does not provide enough stability. For example, only three SIB messages are implemented and paging messages are not implemented at all (its implementation was an objective for June 2017 [5]). Despite these problems, the program runs and we can use it, but, as we will see in section 4.2, it generated problems in the development of the project.

OAI is split in two parts: the access network, called *openair5G*; and the core network, named *openairCN*. These two can work together or independently.

OpenairCN is split in three modules that simulate the EPC, which are: MME, HSS and ”SPGW”. This last one is the combination of both S-GW and P-GW. Again, these modules could work together or independently, allowing a lot of flexibility for the deployment.

The E-UTRAN side of OAI, *openair5g*, is also formed by the emulators of the eNB and the UE, and also a simulation tool called *oaisim*, which simulates the whole access network. Following the philosophy of OAI, these parts can work together or independently, and *oaisim* can work connected to a core network or without this connection.

We can see the structure of OAI and where the division between *openair5g* (E-UTRAN) and *openairCN* (EPC) is in Figure 2.7.

OAI is designed to be agnostic to the hardware RF platforms and it can be interfaced with Software-defined Radio (SDR) RF platforms like USRP.

2.5. ns-3

ns-3 is a discrete-event network simulator mainly targeted for research and educational use. It is also a free software.

It supports research on both IP and non-IP networks. However, it is mainly used for simulating wireless networks such as Wi-Fi, WiMAX or LTE [18]. ns-3 has a large community behind itself, so there are a lot of documentation and examples online.

ns-3 is designed as a set of C++ libraries which provide a set of network simulation models implemented as C++ objects and wrapped through Python. Users normally write their applications using C++ or Python. These applications instantiate a set of simulation models to set up the scenario, start the simulation and end it when it is completed [17].

If we compare it with OAI, ns-3 is clearly easier to use and it gives a lot of freedom to the user in order to set the simulation scenarios and parameters. Their main difference, though, is that ns-3 is focused on simulations even though it can be used for emulation too, and OAI feels focused on emulation, even though simulations are possible. Another important difference is that OAI still needs a lot of development, while ns-3 looks more like a finished product even though it is being updated every few months.

Even though they have been used for the same purpose in this project, it is necessary to stress that the aim of OAI is providing software for real equipment in order to allow the use of open-source UEs, eNBs and core networks even though it provides simulation tools, while ns-3 aims to provide a good simulation environment for different communication technologies mainly for academic purposes.

2.6. Discussion

LWA has been defined by 3GPP and the way it will work is known. However, it is still not implemented in OAI or ns-3. This left us in a scenario where it is impossible to simulate a full LWA network and the only feasible solution would be simulating LTE and IEEE 802.11 separately.

For simulating LTE cells, OAI should be better than ns-3 as it is only focuses on LTE and implements its full stack. On the other hand, Wi-Fi simulations cannot be performed on OAI and they will have to be performed on ns-3. This is not a problem as ns-3 is a very good WLAN simulator.

As the purpose of this project was seeing the impact of offloading beacon information from Wi-Fi to LTE, it is possible to check how the reduction of beacon frames impacts on WLANs (using ns-3) and how adding a new SIB for sending this beacon data affects LTE (using OAI).

A priori, the overall capacity should increase as LTE broadcasts in a more efficient way than IEEE 802.11, so the throughput gain that we get in Wi-Fi should be smaller than the throughput loss that the addition of the new SIB causes to LTE.

3. Project development

In this chapter we are going to explain the whole development of the project from the beginning to the end. We started installing and configuring OAI, and also correcting some bugs. After that, a new SIB was added in the OAI environment. The last part of the development was testing the performance of IEEE 802.11 when reducing the beacon size and increasing the beacon interval.

3.1. Installation of OAI and bug corrections

Installing OAI is not straightforward as it is still under development. As the procedure was complicated and we had to restart the installation many times we prepared an installation manual that can be found in Appendix A.

After installing the access network, we discovered several bugs that we had to fix, e.g.:

- In *dlsim*:
  - Crash when the number of users was greater than one:
    Only user 0 was being initalised due to a typing error in the *for* loop that was supposed to initialise all users.
  - Problem with the number of Physical Downlink Control Channel (PDCCH) symbols when the number of users was greater than two.
    The sequence of *if* that was in charge to calculate the necessary number of PDCCH symbols returned 0 instead of 4 when having four users, and this made the program crash.
- In *oaisim*:
  - Incorrect filename when introducing a scenario template:
    The program did not load the template correctly because the generation of the path was done in a wrong way and did not work when running the program with the option `-c templateX.xml` (as said in *help*). We rewrote the part of code that was wrong and changed the option to only `-c X`, where *X* is the number of the template.
  - Compiling errors when trying to build *oaisim_noS1* with a Message Sequence Chart (MSC) output.
    Due to what it looked like a copy/paste error, a variable inside a function was never declared and it made the program crash.

We also found that what it was supposed to be a *Mscgen* compatible output was not understood by that software, so we had to write a translator. This is explained in the next subsection, 3.1.1.

3.1.1. MSC translator

As written before, *oaisim* has the option of creating an output that can be read by *Mscgen* and create a MSC. The problem in this case was that the file created by *oaisim* (see Figure

```
0 [PROTO] 0 IP_UE
1 [PROTO] 1 NAS_UE
2 [PROTO] 2 RRC_UE
3 [PROTO] 3 PDCP_UE
4 [PROTO] 4 RLC_UE
5 [PROTO] 5 MAC_UE
6 [PROTO] 6 PHY_UE
7 [PROTO] 7 PHY_ENB
8 [PROTO] 8 MAC_ENB
9 [PROTO] 9 RLC_ENB
10 [PROTO] 10 PDCP_ENB
11 [PROTO] 11 RRC_ENB
12 [PROTO] 13 S1AP_ENB
13 [PROTO] 14 GTPU_ENB
14 [PROTO] 15 GTPU_SGW
15 [PROTO] 16 S1AP_MME
16 [PROTO] 18 NAS_MME
17 [EVENT] 7 0 Add ue ffff88d8
18 [MESSAGE] 11 -> 2 0 00011:08 RRCConnectionSetup UE 88d8 size 25
19 [MESSAGE] 11 -> 10 0 00011:08 CONFIG_REQ UE 88d8 SRB
20 [MESSAGE] 9 <- 11 0 00011:08 [RNTI 88d8 SRB AM 00] CONFIG-REQ t_PollRetx 80 t_Reord 35 t_StatusPro 0
21 [EVENT] 3 0 Set security ciph 0 integ 0 UE 88d8
22 [MESSAGE] 4 <- 2 0 00011:00 [RNTI 88d8 SRB AM 00] CONFIG-REQ t_PollRetx 80 t_Reord 35 t_StatusPro 0
23 [MESSAGE] 2 -> 3 0 00011:00RRC_DCCH_DATA_REQ UE 88d8 MUI 0 size 59
24 [MESSAGE] 4 <- 3 0 00011:00 [RNTI 88d8 SRB AM 01] DATA-REQ size 64 mui 0
25 [MESSAGE] 4 -> 9 0 00013:00 [RNTI 88d8 SRB AM 01] DATA SN 0 size 66 RF 0 P 1 FI 0
26 [MESSAGE] 9 <- 4 0 00013:00 [RNTI 88d8 SRB AM 01] DATA SN 0 size 66 RF 0 P 1 FI 0
27 [MESSAGE] 9 -> 10 0 00013:00 [RNTI 88d8 SRB AM 01] DATA-IND size 64
28 [MESSAGE] 10 -> 11 0 [FRAME 00013][eNB][MOD 00][RNTI 88d8][SRB 01]  DATA-IND len 59
29 [MESSAGE] 11 <- 2 0 00013:06 RRCConnectionSetupComplete UE 88d8 size 59
30 [MESSAGE] 11 -> 2 0 00013:06 securityModeCommand UE 88d8 MUI 0 size 3
31 [MESSAGE] 11 -> 10 0 00013:06RRC_DCCH_DATA_REQ UE 88d8 MUI 0 size 3
32 [MESSAGE] 9 <- 10 0 00013:00 [RNTI 88d8 SRB AM 01] DATA-REQ size 8 mui 0
33 [MESSAGE] 9 -> 4 0 00013:00 [RNTI 88d8 SRB AM 01] STATUS ACK_SN 1
34 [MESSAGE] 9 -> 4 0 00013:00 [RNTI 88d8 SRB AM 01] DATA SN 0 size 9 RF 0 P 0 FI 1
35 [MESSAGE] 9 -> 4 0 00013:00 [RNTI 88d8 SRB AM 01] DATA SN 1 size 3 RF 0 P 1 FI 2
36 [MESSAGE] 4 <- 9 0 00013:00 [RNTI 88d8 SRB AM 01] STATUS size ACK_SN 1
37 [MESSAGE] 4 <- 9 0 00013:00 [RNTI 88d8 SRB AM 01] DATA SN 0 size 9 RF 0 P 0 FI 1
38 [MESSAGE] 4 <- 9 0 00013:00 [RNTI 88d8 SRB AM 01] DATA SN 1 size 3 RF 0 P 1 FI 2
39 [MESSAGE] 4 -> 3 0 00013:00 [RNTI 88d8 SRB AM 01] DATA-IND size 8
```

Figure 3.1: Example of *oaisim* MSC output.

3.1) was not in a format that could be read by *Mscgen* (see Figure 3.2), so we had to write a translator in Python in order to make it compatible (the code can be found in Appendix B.

The result after processing the translated input with *Mscgen* can be seen in Figure 3.3.

```
msc {
    0 [label="IP_UE"],1 [label="NAS_UE"],2 [label="RRC_UE"],3 [label="PDCP_UE"],4 [label="RLC_UE"],5 [label="MAC_UE"],6 [label="PHY_UE"],7 [label="PHY_ENB"],8
[label="MAC_ENB"],9 [label="RLC_ENB"],10 [label="PDCP_ENB"],11 [label="RRC_ENB"],13 [label="S1AP_ENB"],14 [label="GTPU_ENB"],15 [label="GTPU_SGW"],16
[label="S1AP_MME"],18 [label="NAS_MME"];

    --- [ label = "7 0 Add ue ffff88d8 " ];
    11->2 [ label = "00011:08 RRCConnectionSetup UE 88d8 size 25"];
    11->10 [ label = "00011:08 CONFIG_REQ UE 88d8 SRB"];
    9<-11 [ label = "00011:08 [RNTI 88d8 SRB AM 00] CONFIG-REQ t_PollRetx 80 t_Reord 35 t_StatusPro 0"];
    --- [ label = "3 0 Set security ciph 0 integ 0 UE 88d8 " ];
    4<-2 [ label = "00011:00 [RNTI 88d8 SRB AM 00] CONFIG-REQ t_PollRetx 80 t_Reord 35 t_StatusPro 0"];
    2->3 [ label = "00011:00RRC_DCCH_DATA_REQ UE 88d8 MUI 0 size 59"];
    4<-3 [ label = "00011:00 [RNTI 88d8 SRB AM 01] DATA-REQ size 64 mui 0"];
    4->9 [ label = "00013:00 [RNTI 88d8 SRB AM 01] DATA SN 0 size 66 RF 0 P 1 FI 0"];
    9<-4 [ label = "00013:00 [RNTI 88d8 SRB AM 01] DATA SN 0 size 66 RF 0 P 1 FI 0"];
    9->10 [ label = "00013:00 [RNTI 88d8 SRB AM 01] DATA-IND size 64"];
    10->11 [ label = "[FRAME 00013][eNB][MOD 00][RNTI 88d8][SRB 01]  DATA-IND len 59"];
    11<-2 [ label = "00013:06 RRCConnectionSetupComplete UE 88d8 size 59"];
    11->2 [ label = "00013:06 securityModeCommand UE 88d8 MUI 0 size 3"];
    11->10 [ label = "00013:06RRC_DCCH_DATA_REQ UE 88d8 MUI 0 size 3"];
    9<-10 [ label = "00013:00 [RNTI 88d8 SRB AM 01] DATA-REQ size 8 mui 0"];
    9->4 [ label = "00013:00 [RNTI 88d8 SRB AM 01] STATUS ACK_SN 1"];
    9->4 [ label = "00013:00 [RNTI 88d8 SRB AM 01] DATA SN 0 size 9 RF 0 P 0 FI 1"];
    9->4 [ label = "00013:00 [RNTI 88d8 SRB AM 01] DATA SN 1 size 3 RF 0 P 1 FI 2"];
    4<-9 [ label = "00013:00 [RNTI 88d8 SRB AM 01] STATUS size ACK_SN 1"];
    4<-9 [ label = "00013:00 [RNTI 88d8 SRB AM 01] DATA SN 0 size 9 RF 0 P 0 FI 1"];
    4<-9 [ label = "00013:00 [RNTI 88d8 SRB AM 01] DATA SN 1 size 3 RF 0 P 1 FI 2"];
    4->3 [ label = "00013:00 [RNTI 88d8 SRB AM 01] DATA-IND size 8"];

}
```

Figure 3.2: Example of translated MSC output (compatible with *Mscgen*).

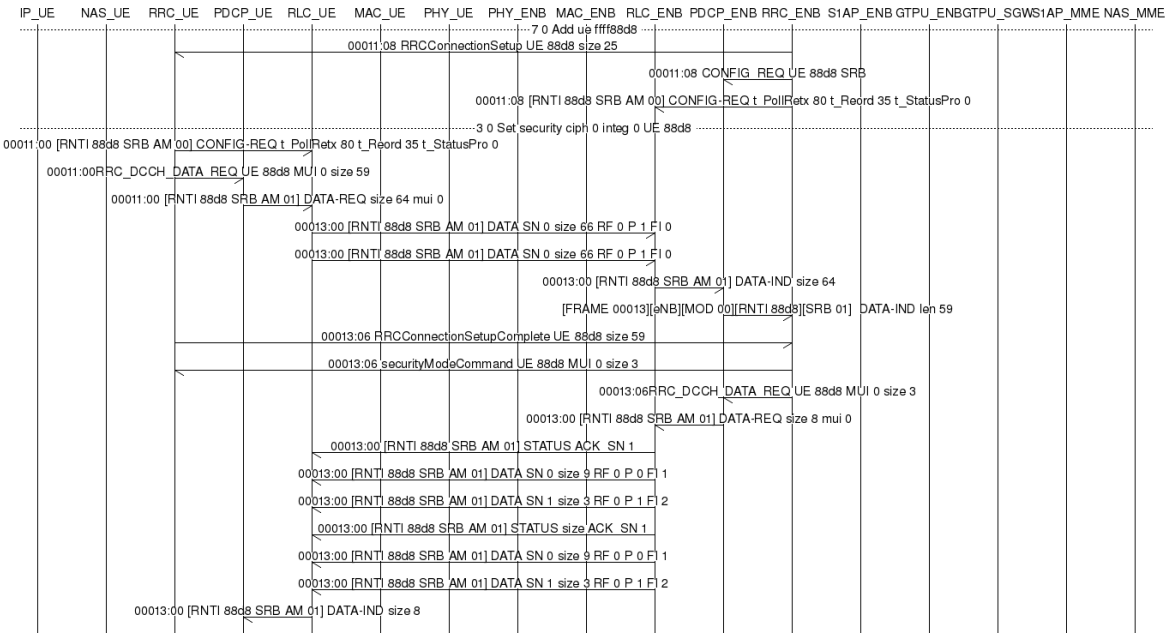


Figure 3.3: Example of MSC generated by *Mscgen*.

3.2. Adding SIB X

In this section we are going to present the changes that have been done in OAI's code in order to add this new SIB, send it and receive it properly. A large number of lines of code have been added, deleted or modified, but we are only going to show the main changes here.

3.2.1. SIB X contents

When creating SIB X we had to set some fields to it in order to send and receive some data. As it was only for testing, we set very simple fields that can be seen in table 3.1.

<table>
<tr><td rowspan="3">SystemInformationBlockTypeX
sibX</td><td rowspan="3">Group1
group1</td><td colspan="2">INTEGER (0..63)
test1</td></tr>
<tr><td rowspan="2">Pair
pair</td><td>BOOLEAN
booleanA</td></tr>
<tr><td>BOOLEAN
boolean B</td></tr>
</table>

Table 3.1: Contents of SIB X.
The italic words tell the type of every element and the bold ones its name.

3.2.2. Creating SIB X using ASN.1

Abstract Syntax Notation One (ASN.1) is an international standard for representing data types and structures. It was created to face one of the fundamental problems confronting users communicating with different systems, which is the efficient transfer of data in such a way that the received data is the same that has been transmitted [11].

ASN.1 is the method of specifying abstract objects in Open Systems Interconnection (OSI), which governs the interconnection of computers from the physical layer up to the application layer and it has to support many different implementations of the services at each layer.

In this way, ASN.1 is a notation that due to this abstract nature can be understood and implemented by the programming languages that are used to program the different functions in each layer [13].

OAI uses ASN.1 in order to automatically generate a big number of `.c` and `.h` files that are used by the simulator. These files include type declarations and the necessary functions to use them. Most of these automatically created files include types related to messages used by the system, for instance SIBs or RRC messages.

In order to create the new SIB with the contents of table 3.1, we had to add several pieces of code in the ASN.1 file that can be found in:
`openair2/RRC/LITE/MESSAGES/asn1c/ASN1_files/EUTRA-RRC-Definitions-a20.asn`.
This ASN.1 file is used by the compiler in order to generate all the necessary C files for the SIBs, among other messages.

The code after the changes is shown in code listing 3.1.

```
SystemInformation-r8-IEs ::= SEQUENCE {
  sib-TypeAndInfo   SEQUENCE (SIZE (1..maxSIB)) OF CHOICE {
    sib2          SystemInformationBlockType2,
    sib3          SystemInformationBlockType3,
    sib4          SystemInformationBlockType4,
    sib5          SystemInformationBlockType5,
    sib6          SystemInformationBlockType6,
    sib7          SystemInformationBlockType7,
    sib8          SystemInformationBlockType8,
    sib9          SystemInformationBlockType9,
    sib10         SystemInformationBlockType10,
    sib11         SystemInformationBlockType11,
    sibX          SystemInformationBlockTypeX,
    ...,
    sib12-v920        SystemInformationBlockType12-r9,
    sib13-v920        SystemInformationBlockType13-r9
  },
  nonCriticalExtension   SystemInformation-v8a0-IEs       OPTIONAL
}

...

SystemInformationBlockTypeX ::= SEQUENCE { -- MCC: Define what is SIBX
  group1          Group1
}

Group1 ::=        SEQUENCE { -- MCC: Define Group1 for SIBX
  test1         INTEGER(0..63),
  pair          Pair
}

Pair ::=          SEQUENCE { -- MCC: Define Pair for Group1
  booleanA        BOOLEAN,
  booleanB        BOOLEAN
}

...

SIB-Type ::=      ENUMERATED {
      sibType3, sibType4, sibType5, sibType6,
      sibType7, sibType8, sibType9, sibType10,
      sibType11, sibType12-v920, sibType13-v920, sibTypeX,
      spare4, spare3, spare2, spare1, ...}
                    -- MCC: changed "spare5" by "sibTypeX"
```

Code Listing 3.1: ASN.1 code for adding SIB X.

After the edit in code listing 3.1, when we built OAI we had a problem with the SHA-1 hash of `SystemInformation-r8-IEs.h` because it did not match the number that the building script was expecting. We had to modify the file called `fix_asn1` (located in `cmake_targets/tools`) and comment the line where it checked the SHA-1 checksum, in function *apply_patches()*.

After the first build we were able to uncomment the previously commented line and modify the array in the beginning of the same file that contained the value of the old hash of `SystemInformation-r8-IEs.h`. We changed it for the number we obtained calculating its new hash so now when checking the hash it will contemplate the last changes we made on the ASN.1 file that generates `SystemInformation-r8-IEs.h`.

3.2.3. Sending SIB X

It was necessary to implement mechanisms in the eNB side in order to send SIB X. We first had to create the function that initialises the values of the new SIB, *do_SIBX()*, in `asn1_msg.c` and `asn1_msg.h` (which can be found in `openair2/RRC/LITE/MESSAGES/`). We can see a summary of this function in code listing 3.2.

```
uint8_t do_SIBX(uint8_t Mod_id, // MCC: Added for initialising SIBX
                int CC_id,
                LTE_DL_FRAME_PARMS *frame_parms,
                uint8_t *buffer,
                BCCH_DL_SCH_Message_t *bcch_message,
                SystemInformationBlockTypeX_t **sibX

// ...

  //MCC: Memory allocation
  sibX_part = CALLOC(1,sizeof(struct
    SystemInformation_r8_IEs__sib_TypeAndInfo__Member));
  memset(sibX_part,0,sizeof(struct
    SystemInformation_r8_IEs__sib_TypeAndInfo__Member));

  //MCC: Setting message as SIB X
  sibX_part->present = SystemInformation_r8_IEs__sib_TypeAndInfo__Member_PR_sibX;

  *sibX = &sibX_part->choice.sibX;

  // MCC: Setting SIB X fields:
  (*sibX)->group1.test1 = 47;

  (*sibX)->group1.pair.booleanA = TRUE;
  (*sibX)->group1.pair.booleanB = FALSE;

// ...

  // MCC: Setting BCCH message as SIB
  bcch_message->message.present = BCCH_DL_SCH_MessageType_PR_c1;
  bcch_message->message.choice.c1.present =
    BCCH_DL_SCH_MessageType__c1_PR_systemInformation;

  bcch_message->message.choice.c1.choice.systemInformation.criticalExtensions.
    present = SystemInformation__criticalExtensions_PR_systemInformation_r8;

  bcch_message->message.choice.c1.choice.systemInformation.criticalExtensions.
    choice.systemInformation_r8.sib_TypeAndInfo.list.count=0;

// ...

  // MCC: Sending message to buffer
  enc_rval = uper_encode_to_buffer(&asn_DEF_BCCH_DL_SCH_Message,
                                   (void*)bcch_message,
                                   buffer,
                                   900);
  AssertFatal (enc_rval.encoded > 0, "ASN1 message encoding failed (%s, %lu)!\n",
               enc_rval.failed_type->name, enc_rval.encoded);

// ...

  return((enc_rval.encoded+7)/8);
```

```
}
```

Code Listing 3.2: Summarised code of function *do_SIBX()*.

After this we also needed to edit the function *init_SI()* in `openair2/RRC/LITE/rrc_enb.c` in order to do the initialisation of the new SIB. In code listing 3.3 we can see that we call *do_SIBX()* there.

```
static void init_SI(const protocol_ctxt_t* const ctxt_pP, const int CC_id) {
  uint8_t                                 SIwindowsize = 1;
  uint16_t                                SIperiod = 8;

  //...

  // MCC: Allocating memory for SIB X.
  eNB_rrc_inst[ctxt_pP->module_id].carrier[CC_id].SIBX = (uint8_t*) malloc16(64);

  if (eNB_rrc_inst[ctxt_pP->module_id].carrier[CC_id].SIBX) { // MCC: If SIBX
    exists (there is memory allocated for it)
      eNB_rrc_inst[ctxt_pP->module_id].carrier[CC_id].sizeof_SIBX = do_SIBX(
            ctxt_pP->module_id,
            CC_id,
            mac_xface->lte_frame_parms,
            eNB_rrc_inst[ctxt_pP->module_id].carrier[CC_id].SIBX,
            &eNB_rrc_inst[ctxt_pP->module_id].carrier[CC_id].systemInformation,
            &eNB_rrc_inst[ctxt_pP->module_id].carrier[CC_id].sibX
      //...
          );
    //  ...
      }
  }
  //...
}
```

Code Listing 3.3: Summarised code of function *init_SI()*.

We also had to edit the function *mac_rrc_data_req()* in `openair2/RRC/LITE/L2_interface.c`, which is the piece of code in charge of broadcasting the SIBs, in order to get SIB X sent periodically.

We can see in code listing 3.4 that we are sending SIB X every 16 frames starting in frame 3. This field can be changed depending on the periodicity we want.

```
//...
else if ((frameP%16) == 3) {
        //MCC: Here we send SIB X
      memcpy(&buffer_pP[0],
            eNB_rrc_inst[Mod_idP].carrier[CC_id].SIBX,
            eNB_rrc_inst[Mod_idP].carrier[CC_id].sizeof_SIBX);
//...
```

Code Listing 3.4: Piece of code extracted from *mac_rrc_data_req()*.

3.2.4. Receiving SIB X

After we are sending the new SIB successfully, we need to go to the UE side in order to make the UEs able to receive and understand it.

We had to create a function called *dump_SIBX()*, which is in charge of printing the contents of SIB X,

We had to edit the function *decode_SI()* in `openair2/RRC/LITE/rrc_UE.c` in order to add a case for the new SIB, as it can be seen in code listing 3.5.

```
static int decode_SI( const protocol_ctxt_t* const ctxt_pP, const uint8_t
    eNB_index )
{
    //...
    case SystemInformation_r8_IEs__sib_TypeAndInfo__Member_PR_sibX:
      if(UE_rrc_inst[ctxt_pP->module_id].listenToSIBXflag == 1){
          if((UE_rrc_inst[ctxt_pP->module_id].Info[eNB_index].SIStatus&2048) ==
    0){ //MCC: SISstatus is used as an array of 32 flags
            //UE_rrc_inst[ctxt_pP->module_id].Info[eNB_index].SIStatus|=2048;
            //new_sib=1;
            //MCC: These two lines are commented to keep listening to SIB X

            //MCC: We save the contents of SIB X
            memcpy( UE_rrc_inst[ctxt_pP->module_id].sibX[eNB_index], &typeandinfo
    ->choice.sibX, sizeof(SystemInformationBlockTypeX_t) );

            dump_sibX( UE_rrc_inst[ctxt_pP->module_id].sibX[eNB_index] );
          }
      }
      break;
    //...
}
```

Code Listing 3.5: Part of function *decode_SI()* where we handle the reception of SIB X.

As we can see in code listing 3.5, we commented two lines in order to keep receiving SIB X every time it is sent. The reason for keeping them is that all the other SIBs work in a slightly different way and they are ignored once they have been listened for the first time until the UEs have to listen to them again because they have been told to do so or it has been 3 hours since the last time they received it (see section 2.1.1.1).

3.3. Notifying the UEs about SIB X

OAI with the changes that we have done so far presents a scenario where eNBs are always transmitting SIB X and UEs are always listening to it. Taking into account that this SIB will not always be sent and that the network may decide to activate LWA only for certain users, we thought that we needed a way to notify these users about the existence of LWA and also inform them that SIB X would be available (because we could have LWA without implementing SIB X).

3.3.1. Deciding how to notify

SIB 1 has a field called *systemInfoValueTag* that tells if any SIB has changed. Our problem was that the UEs do not listen to SIB 1 with enough frequency for this case.

Another way to inform UEs about a system information change is to send a Paging message to them including the *systemInfoModification* field. In this way, UEs know that there are

going to be changes in the system information . We could not implement this method because paging messages were not implemented in OAI yet.

The first alternative regarding the way to notify UEs about SIB X was sending a RRC Connection Reconfiguration message to every UE to tell them to start listening to SIB X. In order to do so we had to create a new kind of RRC Connection Reconfiguration message. This can be found in section 3.3.2.

The second idea was implementing the LWA Configuration message from Release 14, and adding a field to it that would inform the UE about the availability of SIB X. The methodology we followed in this case is explained in section 3.4. This message can be implemented from scratch in OAI even though OAI does not support LWA. In this case we are going to send the message after a certain trigger event in the same way we send the new RRC Connection Reconfiguration message, so LWA will not be implemented in OAI but this message will be sent anyway.

3.3.2. Creating new RRC Connection Reconfiguration message

To create this new message we had to edit `EUTRA-RRC-Definitions-a20.asn` as when we created SIB X. This new message will send a boolean and an integer telling if SIB X is available and its period. We can see its contents in code listing 3.6.

```
DL-DCCH-MessageType ::= CHOICE {
  c1 CHOICE {
    csfbParametersResponseCDMA2000      CSFBParametersResponseCDMA2000,
    dlInformationTransfer               DLInformationTransfer,
    handoverFromEUTRAPreparationRequest     HandoverFromEUTRAPreparationRequest,
    mobilityFromEUTRACommand            MobilityFromEUTRACommand,
    rrcConnectionReconfiguration        RRCConnectionReconfiguration,
    listenRRCConnectionReconfiguration      RRCConnectionReconfiguration,
    -- MCC: Added to be able to receive it
    rrcConnectionRelease                RRCConnectionRelease,
    securityModeCommand                 SecurityModeCommand,
    ueCapabilityEnquiry                 UECapabilityEnquiry,
    counterCheck                        CounterCheck,
    ueInformationRequest-r9             UEInformationRequest-r9,
    loggedMeasurementConfiguration-r10      LoggedMeasurementConfiguration-r10,
    rnReconfiguration-r10               RNReconfiguration-r10,
    spare4 NULL,
    spare3 NULL, spare2 NULL, spare1 NULL
    },
  messageClassExtension SEQUENCE {}
}

...

RRCConnectionReconfiguration ::=  SEQUENCE {
  rrc-TransactionIdentifier RRC-TransactionIdentifier,
  criticalExtensions CHOICE{
    c1 CHOICE{
      rrcConnectionReconfiguration-r8 RRCConnectionReconfiguration-r8-IEs,
      listenToSIBX ListenToSIBX, -- MCC: Here we had "spare7 NULL" before
      spare6 NULL, spare5 NULL, spare4 NULL,
      spare3 NULL, spare2 NULL, spare1 NULL
    },
    criticalExtensionsFuture SEQUENCE {}
```

```
  }
}

ListenToSIBX ::= SEQUENCE { --MCC: Defined type for RRCConnectionReconfiguration
  listen BOOLEAN,
  period INTEGER(0..16) -- MCC: We can make it bigger if we need
}
```

Code Listing 3.6: ASN.1 code for adding the new RRC Connection Reconfiguration message.

As we can see in 3.6, we called this new message *listenRRCConnectionReconfiguration*. After creating it, we needed to find some event that triggered it. We realised that a good moment to send it would be after the DRB assignment seen in subsection 2.1.2.

To do so, we added the piece of code that can be seen in code listing 3.7 at the end of the function that processes *RRCConnectionReconfigurationComplete* messages in the eNB side. We did it this way because this message is sent for the first time after the eNB sends the *RRCConnectionReconfigurationRequest* message during the DRB establishment. In order to avoid sending this message every time we receive a *RRCConnectionReconfigurationComplete*, we set up a flag (per user) that gets set to 1 after the message is sent for the first time. This flag is called *listenToSIBXsent* and it can be seen in code listing 3.7.

```
void rrc_eNB_process_RRCConnectionReconfigurationComplete(
  const protocol_ctxt_t* const ctxt_pP,
  rrc_eNB_ue_context_t*        ue_context_pP,
  const uint8_t xid
)
{
// ...
  if(ue_context_pP->listenToSIBXsent == 0){ //MCC: Flag
    //MCC: We send the listenRRCConnectionReconf here.
    rrc_eNB_generate_listenRRCConnectionReconfiguration(ctxt_pP, ue_context_pP,
    1, 4);

    ue_context_pP->listenToSIBXsent = 1;
  }
}
```

Code Listing 3.7: Fragment where *listenRRCConnectionReconfiguration* message is sent.

As we had to do with SIB X, we also had to create new functions in order to set up the message in a proper way. These functions are:

- *rrc_eNB_generate_listenRRCConnectionReconfiguration()*, which can be seen in code listing 3.8, in `opeiair2/RRC/LITE/rrc_eNB.c`.
- *do_listenRRCConnectionReconfiguration()* in `opeiair2/RRC/LITE/MESSAGES/asn1_msg.c`.

```
void rrc_eNB_generate_listenRRCConnectionReconfiguration(
              const protocol_ctxt_t* const ctxt_pP,
                rrc_eNB_ue_context_t*          const ue_context_pP,
              int                listen,
              int                listenPeriod
                )
{
  uint8_t size;
  uint8_t buffer[100];

```

```
  DL_DCCH_Message_t dl_dcch_msg;
  RRCConnectionReconfiguration_t *rrcConnectionReconfiguration;

  memset(&dl_dcch_msg,0,sizeof(DL_DCCH_Message_t));

    // MCC: The initialisation of the message is done here.
    size = do_listenRRCConnectionReconfiguration(ctxt_pP, dl_dcch_msg, &
    rrcConnectionReconfiguration, listen, listenPeriod, &buffer);

// ...

    rrc_data_req( // MCC: The message is sent here.
            ctxt_pP,
            DCCH,
            rrc_eNB_mui++,
            SDU_CONFIRM_NO,
            size,
            buffer,
            PDCP_TRANSMISSION_MODE_CONTROL);

}
```

Code Listing 3.8: Summary of the function that generates the *listenRRCConnectionReconfiguration* message.

We also had to edit the UE side in order to understand the new message and act in consequence. We created a flag that the UE will set to 1 whenever it receives the message telling it to start listening to SIB X. In this way all UEs ignore SIB X until they are told to listen to it.

This flag was called *listenToSIBXflag* and we can see how we set it to the value that is carried by the *listenRRCConnectionReconfiguration* message in code listing 3.9. We can also see how it is taken into account at the moment of receiving SIB X in code listing 3.5.

```
void rrc_ue_decode_dcch(const protocol_ctxt_t* const ctxt_pP, const rb_id_t
    Srb_id, const uint8_t* const Buffer, const uint8_t eNB_indexP)
{
// ...
      case DL_DCCH_MessageType__c1_PR_listenRRCConnectionReconfiguration:

      //MCC: We set the "listen" flag here.
      UE_rrc_inst[ctxt_pP->module_id].listenToSIBXflag = dl_dcch_msg->message.
    choice.c1.choice.listenRRCConnectionReconfiguration.criticalExtensions.choice
    .c1.choice.listenToSIBX.listen;

      //MCC: Reply with a RRCConnectionReconfigurationComplete
      rrc_ue_generate_RRCConnectionReconfigurationComplete(
                    ctxt_pP,
                    target_eNB_index,
                    dl_dcch_msg->message.choice.c1.choice.
    rrcConnectionReconfiguration.rrc_TransactionIdentifier);
        break;
// ...
}
```

Code Listing 3.9: Part of function *rrc_ue_decode_dcch()* where we handle the reception of a *listenRRCConnectionReconfiguration*.

3.4. Implementing LWA Configuration message

After implementing *listenRRCConnectionReconfiguration* message as we have seen in subsection 3.3.2, we realised that a better way to do it may be using LWA specific messages and save RRC Connection Reconfiguration messages for other purposes, as RRC Connection Reconfiguration messages have spare space for implementing new features.

It was in this moment when we focused the project to LWA, while until this point it had been focused on implementing a new generic SIB for "future uses".

As we had to do before with the other two messages we implemented, we first had to implement the LWA Configuration message in ASN.1 as it can be seen in code listing 3.10.

```
LWA-Configuration-r13 ::=      CHOICE {
  release                        NULL,
  setup                        SEQUENCE {
    lwa-Config-r13                    LWA-Config-r13
  }
}

LWA-Config-r13 ::=   SEQUENCE {
  lwa-MobilityConfig-r13      WLAN-MobilityConfig-r13     OPTIONAL, -- Need ON
  lwa-WT-Counter-r13          INTEGER (0..65535)          OPTIONAL, -- Need ON
  listenToSIBX            BOOLEAN, -- MCC: Added
  listenPeriod            INTEGER (0..255), -- MCC: Added
  ...
}
```

Code Listing 3.10: ASN.1 code for LWA Configuration message.

As we can observe in 3.10, we added a couple of parameters in *LWA-Config-r13*: the boolean *listenToSIBX* and the integer *listenPeriod*. They are the fields of the message that we are going to use for enabling the listening of SIB X.

3.4.1. eNB side

As we saw with the previous messages as well, we had to create a couple of functions at the eNB side that are in charge of setting up the message. They are the ones listed below:

- *rrc_eNB_generate_LWAConfigurationSetup()* in `opeiair2/RRC/LITE/rrc_eNB.c`.
- *do_LWAConfigurationSetup()* in `opeiair2/RRC/LITE/MESSAGES/asn1_msg.c`.

In this case we only set up the "setup" message for LWA Configuration, which is the one that includes the variables for enabling the listening of SIB X. We did not create the functions for the "release" message because we would not use it in our simulations.

After that, we replaced the part of the code where we sent the *listenRRCConnectionReconfiguration* message to send the LWA Configuration Setup message instead. We can see this change in code listing 3.11.

```
void rrc_eNB_process_RRCConnectionReconfigurationComplete(
  const protocol_ctxt_t* const ctxt_pP,
  rrc_eNB_ue_context_t*        ue_context_pP,
```

```
  const uint8_t xid
)
{
// ...
  if(ue_context_pP->listenToSIBXsent == 0){ //MCC: Flag

    //MCC: Sending lwa-Config here.
    rrc_eNB_generate_LWAConfigurationSetup(ctxt_pP, ue_context_pP, 1, 4);

    ue_context_pP->listenToSIBXsent = 1;
  }
}
```

Code Listing 3.11: Fragment where LWA Configuration Setup message is sent.

3.4.2. UE side

In order to receive and understand this new message, we also had to edit the UE side of OAI. Similarly as we did with the previous created messages, we added a case in function *rrc_ue_decode_dcch()* in textttopenair2/RRC/LITE. We can see this edit in listing 3.12.

```
void rrc_ue_decode_dcch(const protocol_ctxt_t* const ctxt_pP, const rb_id_t
    Srb_id, const uint8_t* const Buffer, const uint8_t eNB_indexP)
{
// ...
      case DL_DCCH_MessageType__c1_PR_lwaConfiguration_r13:
                UE_rrc_inst[ctxt_pP->module_id].listenToSIBXflag = dl_dcch_msg->
    message.choice.c1.choice.lwaConfiguration_r13.choice.setup.lwa_Config_r13.
    listenToSIBX;

        break;
        // ...
}
```

Code Listing 3.12: Piece of code added to handle the reception of LWA Configuration Setup messages.

3.5. Selecting the scenario

In order to test the impact of changing the beacon size and interval we had to set up a set of relevant scenarios. The most straightforward case would be an indoors scenario with a high density of cells, so APs and STAs would receive the beacons from a large number of cells. This, together with CSMA/CA medium access procedures would result in a lot of time used for the transmission of beacons.

Our scenario was adapted from scenario 3, ”Indoor Small BSSs Scenario”, from [9], which has the objective to capture the issues and be representative of real-world deployments with high density of APs and STAs.

The infrastructure is planned. A hexagonal BSS layout is considered with a frequency reuse pattern, which can be 1, 3 or more. We used different frequency reuse values during the simulations. The most realistic frequency reuse value would be 3, even though frequency reuse 1 should also be considered because we could have limited available bandwidth or

multiple overlapped WLAN deployments from multiple network operators (with 3 operators using reuse 3 we would have a scenario that would be very similar to the one with one operator with reuse 1).

With this scenario we can test the impact of having multiple APs transmitting beacons and interfering each other. In this case, the neighbours of the AP that is broadcasting a beacon have to keep silent while the transmission is going on. This makes the impact of the size and the interval of the beacons even more important, because the time that is used for beacons becomes the time that one AP would take multiplied by the number of APs.

This scenario has the following features:

- **Environment description:**
  - BSSs are placed in a regular and symmetric grid as in Figure 3.4 for frequency reuse 1 and in Figure 3.5 for frequency reuse 3, which only shows the cells that are using the same frequency.
  - Each hexagon has the following configuration:
    - Radius (R): 10 m
    - Inter BSS distance: $2h$ m, where $h = \sqrt{R^2 - R^2/4}$.
- **APs location:**
  - APs are placed at the center of each hexagon, with 3m antenna height.
- **STAs location:**
  - STA antenna height of 1.5 m.
  - STAs are placed randomly within the cells area.
- **Number of STAs:**
  - The number of STAs varies depending on the experiment.
- **Channel model:**
  - Path loss model:
    - $PL(d) = 40.05 + 20 \cdot log_{10}(f_c/2.4) + 20 \cdot log_{10}(min(d, 10)) + (d > 10) \cdot 35 \cdot log_{10}(d/10)$
    - Where d is the distance in metres and f_c is the frequency in GHz.
- **AP Tx power:** 17 dBm

Figure 3.4: Example of a hexagonal BSS layout [14].

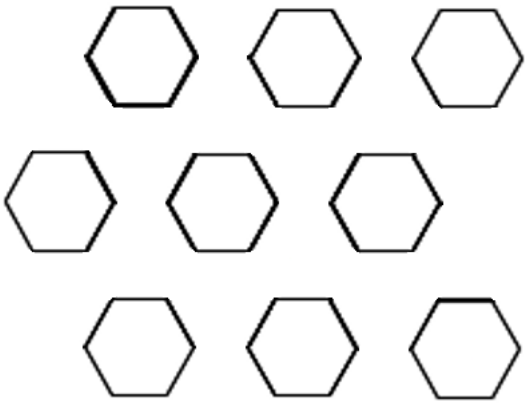


Figure 3.5: Example of a hexagonal BSS layout with reuse 3.

3.6. Theoretical calculations

Taking into account that beacons must be sent following CSMA/CA, it is not difficult to calculate how much time we can save when reducing beacon size or reducing beacon interval in a quiet environment.

As we can see in Figure 3.6, management frames in IEEE 802.11n have a header of 28 B plus 4 B of Frame Check Sequence (FCS) at the end in order to perform error detection. This means that the minimum size of a management frame without any information in the frame body would be of 32 B.

If we look at the beacon components (found in Appendix C), we can see that the mandatory fields are the following:

- Timestamp: 8 B.
- Beacon interval: 2 B.
- Capability information: 2 B.
- Service Set Identifier (SSID): max 2+32 B.
- Supported rates and BSS Membership selectors: max 2+8 B.

This makes a minimum beacon size of 18 B considering a SSID field of 2+1 B and a ”supported rates and BSS Membership selectors” field of 2+1 B, too. These 18 B should be added of the 32 B of overhead seen before, so the complete frame would have a size of 60 B.

This assumption is not real, as we will not find these two fields being that short in real environments, so we assumed that a reasonable minimum beacon size could be 38 B, which corresponds to having a SSID field of 2+16 B and a ”supported rates and BSS Membership selectors” field of 2+6 B. These 38 B of beacon frame make a total frame size of 70 B when the overhead is added.

At this point we can calculate the time gain we would get if we were sending these small beacons instead of ”standard” beacons with a total size of 300 B in a quiet environment.

In order to calculate this time gain we are going to compute the amount of time that the channel would be used for transmitting beacons in every case.

If we consider beacons with a size of 300 B every 100 TU and we are using IEEE 802.11n at 5 GHz:

- DIFS = 34 μs

Octets: 2	2	6	6	6	2	4	0–2312	4
Frame Control	Duration	Address 1 (DA)	SA	BSS ID	Sequence Control	HT Control	Frame Body	FCS

MAC header

Figure 3.6: IEEE 802.11n management frame structure [15].

- Slowest data rate = 6.5 Mbps
- Time needed for sending a beacon = $(300 \cdot 8)/(6.5 \cdot 10^6)$ = 369.2 µs
- We are using 369.2 µs every 100 TU. A TU equals to 1024 µs, so 100 TU are 102400 µs. This means that in this case beacons are taking 0.36 % of the time.

If we consider short beacons of 70 B and transmitted every 300 TU in the same scenario:

- DIFS = 34 µs
- Slowest data rate = 6.5 Mbps
- Time needed for sending a beacon = $(70 \cdot 8)/(6.5 \cdot 10^6)$ = 86.2 µs
- We are using 86.2 µs every 300 TU. 300 TU equal to 307200 µs. This means that in this case beacons are taking 0.028 % of the time.

We can see how reducing the beacon size and increasing the beacon interval leads us to use at least 10 times less time for beacons, allowing more time to be used for sending data.

If we increase the scale of the infrastructure deployment and now we consider that there are 20 APs that can listen to each other, we can calculate the time gain assuming that there are no collisions when the beacon is going to be sent, as we assumed before. This means that in this case, APs would have to be silent when any other AP sends the beacon, so the total percentage of time used for beacons would be:

- 7.2 % if we were using beacons of 300 B sent every 100 TU.
- 0.56 % if we were using beacons of 70 B sent every 300 TU.

We need to take into account that these results are only considering the use of the channel. If we looked from the AP side, the time that each AP uses for transmitting a beacon would be bigger as we should consider the DIFS waiting time for checking the channel is free and also the probable collisions. In case of collisions, CSMA/CA makes the AP that wants to transmit wait for a DIFS plus a random back-off time before trying to transmit again. This behaviour makes the beacon size less relevant when calculating the time used for sending the beacon, as the waiting time keeps increasing.

However, by reducing the beacon size and increasing its interval, we also reduce the chances of collision against beacons because we reduce the use of the media.

In order to take into account scenarios of larger scale and also see the difference on the throughput that users see, several simulations were made in order to see the benefits of using smaller and less frequent beacons. They are explained in section 3.7.

3.7. ns-3 simulations

Different simulations with ns-3 were performed in order to test the improvement in the throughput that STAs could see when reducing the size and the interval of beacons. The scenario is described in section 3.5.

The first simulation used a scenario with 7 cells and frequency reuse 1, this was mainly to practice with ns-3 as it was the first time we used it.

Following simulations increased the number of APs in the scenario up to 85, as, using a transmitted power of 17 dBm, hexagonal cells and a distance between APs of 10 m, was the maximum number of APs that the central AP would hear. In order to calculate that, we used the formula of the path loss model from 3.5.

To see it from a spacial point of view, 85 APs in this setup make 5 rings of hexagonal cells without the 6 ”corner cells” from the largest ring.

We simulated scenarios with 85 cells and 1 STA, so we could see the number of beacons that this station would see in each case. After, we increased the number of users up to 1 per cell in order to see the impact of the beacons in networks with higher density. As we were using full buffer transmissions, 1 user would use the AP in an equivalent way as several users connected to it but with less traffic.

In order to calculate the throughput, we transmitted this full buffer data in DL direction.

The first simulations were made using IEEE 802.11n standard at 5 GHz, but we changed it to 802.11ac with 160 MHz of bandwidth, so we could have a higher throughput. Another change that we introduced was in ”Remote Station Manager”, which controls the whole operation of APs including the Modulation and Coding Scheme (MCS) and so controlling the data rate. First, we used a constant rate Wi-Fi manager, but we changed it for the Minstrel High Throughput (HT) Wi-Fi manager, which implements the Minstrel HT Rate Control Algorithm [22], which adapts the MCS, channel width, number of streams, and short guard interval (enabled or disabled).

We collected the throughput from these simulations in order to compare the results when using different beacon sizes and intervals. Most of these simulations were performed by the servers in the 5GIC as they would require several hours to be completed even with a powerful machine.

All the simulation programs were written in C++ and the results can be seen in chapter 4.

4. Results

In this chapter we are going to expose the results obtained during the development of the project. As the project has two different parts, the one related to OAI and the one related to ns-3, we have divided this chapter in two sections. The first one will be ns-3 because even though we worked first with OAI, the simulations of LTE were done after the ns-3 simulations of Wi-Fi.

4.1. IEEE 802.11 ns-3 simulations

As we have seen in section 3.7, several simulations were performed with ns-3.

The first one implemented 7 APs with only one STA. This STA was connected to the central AP and full-buffer DL traffic was implemented. In this scenario, the default case (300 B[1] of beacon size and 100 TU of beacon interval) makes beacons use the channel a 2.56 % of the time.

If we reduced the beacon size to 100 B, this percentage was reduced to a 0.84 %, and, if we increased the beacon interval (keeping the size of 100 B) to 300 TU, it was reduced to a 0.30 % of the time. If we kept increasing the beacon interval to 500 TU, this would get to a 0.18 %.

In this case, the STA saw a very high throughput in both cases, around 45 Mbps. However, there was a gain of a 2.6 % between the best and the worst cases, 100 B of beacon size with a beacon interval of 500 TU and 300 B of beacon size with a beacon interval of 100 TU respectively.

The next case made our scenario much bigger by using the maximum number of cells, 85, calculated in section 3.7, but having reuse 3. In this case, we calculated graphically that the number of interfering cells would be 30 instead of the 84 we would have with reuse 1.

The results in this case went from a 8.48 % of the time when using 300 B beacons sent every 100 TU to 0.75 % when using 100 B beacons sent every 500 TU. The throughput gain, again we have only one STA receiving a full-buffer stream, would be of a 8.8 % when comparing the best case to the worst one. In this case, the gain would be between 3 and 4 Mbps.

The next scenario was the same but with reuse 1, meaning that there are 84 neighbours interfering our cell. In this case, the channel time used by beacons is a 15.61 % of the time when having a beacon size of 300 B and a beacon interval of 100 TU. When having a beacon size of 100 B and a beacon interval of 500 TU, the time used by beacons decreases to a 1.66 %. The throughput gain is a 19 % between the best and the worst case, gaining around 7 Mbps.

If we change to IEEE 802.11ac we can observe a very similar behaviour. In this case, using 802.11ac with 160 MHz of channel bandwidth, and keeping the same topology as before, we got that in the worst case (300 B of beacon size and 100 TU of beacon interval) the time the channel is used by beacons is a 17.6 %, while when using a beacon size[2] of 110 B sent

[1] The size takes into account the header and the FCS.
[2] 110 B was the smallest beacon size we could use.

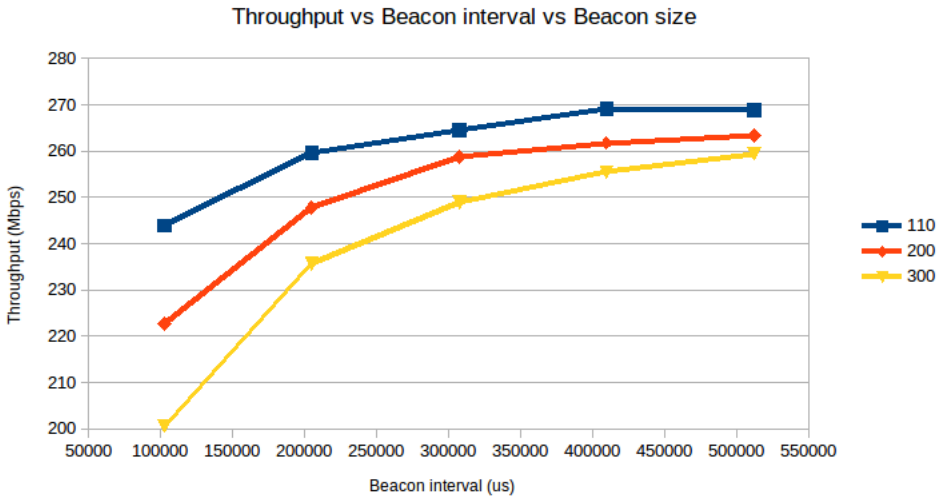


Figure 4.1: Results of the simulation with 85 APs and 1 STA.

every 500 TU we got a 3.0 %. From the throughput point of view, we get an improvement of a 34 % from the worst to the best case, in absolute numbers, this gain supposes almost 70 Mbps. We can see these results in Figure 4.1.

After that, we made much denser simulations where the number of beacons was very large and much more difficult to count, as it had to be done manually due to the characteristics of the simulations. In these cases, we compared the performance taking into account only the average throughput. In addition, we can assume that the number of beacons will not vary from what we got in the scenario with only one STA, as they have to be transmitted periodically in all cases.

In these new simulations we used the Minstrel HT Wi-Fi Manager in IEEE 802.11ac. We used 85 STAs, one in a random position within each cell. Each STA was receiving a full-buffer bit stream from its AP. This scenario with a lot of traffic, and thus collisions, made the throughput become very small for everyone.

In this case with 85 APs and 85 STAs, the worst overall average throughput, calculated averaging the sum of the throughput that each STA saw over the 100 simulations we did, was 50.7 Mbps for the scenario where the beacon size was 300 B and the beacon interval, 100 TU. The best case was when having a beacon size of 110 B and a beacon interval of 500 TU, resulting in an average total throughput of 64.0 Mbps. This supposes an increase of a 26.2 %, however, the results show a very low throughput caused by the huge amount of interference generated by the 85 DL connections between the APs and the STAs. We can see these results in figure 4.2.

We did a last simulation reducing the amount of STAs to 40, each one receiving a full-buffer data stream from its AP. In this case we went from 61.5 Mbps of average total throughput when having beacons of 300 B and an interval of 100 TU to 75.9 Mbps when having beacons of 110 B and an interval of 400 TU. This supposes a gain of a 23.4 %.

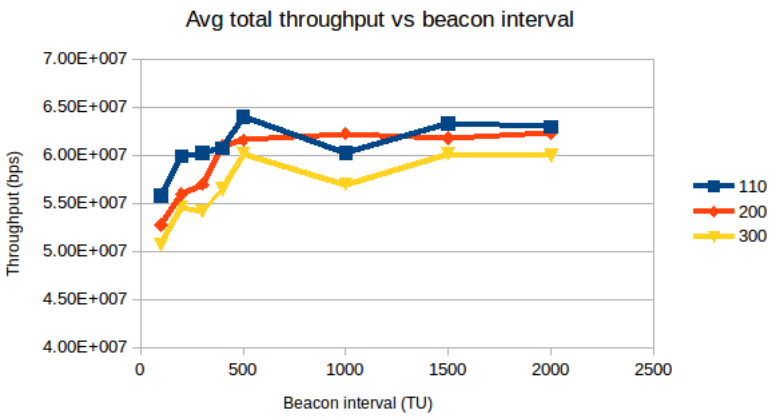


Figure 4.2: Results of the simulation with 85 APs and 85 STAs.

4.2. LTE with OpenAirInterface

After the simulations with ns-3, we tried to simulate a LTE cell in order to see the impact that adding a new SIB would have on the throughput that the cell would see.

As OAI is still under development and has some non-fixed issues, when simulating the cell we got the exact same throughput when the eNB transmitted a SIB X of a size of 200 B and when it did not transmit it. In addition, different random number seeds generated the same output as well.

The throughput that the cell saw was several hundreds of kbps. As this result is not normal at all, because it should be of around 75 Mbps, we investigated the cause of it.

This low throughput was caused by an authentication problem that did not let the UE connect to the eNB, and the number we got was probably based on the amount of control traffic that had been exchanged between the UE and the eNB.

This bug was impossible to solve because we ran out of time. However, we can assume that broadcasting a SIB over LTE is more efficient than broadcasting a beacon over Wi-Fi thanks to the much larger amount of control that LTE has over the channel compared to WLAN. This difference is mainly caused by the use of licensed spectrum by ”traditional” LTE, this makes that all those resources are going to be used only by this technology, so eNBs’ can schedule the use of the channel in order to use it in the most efficient way and they do not have to be aware of interferences and being ”good neighbours” as WLANs have.

These tests will be able to be completed once this authentication problem is solved. We were able to test that the new SIB was broadcasted and received correctly by the UEs. The configuration messages we also created in Chapter 3 were also sent correctly and understood by the receiver UEs. These facts mean that the only barrier that exists between us and a correct simulation when SIB X is deployed is the same that would exist when trying to simulate the system without any changes: this authentication problem when the UE tries to connect to the eNB. Hopefully, this issue will be solved soon by the developers of OAI.

5. Budget

During the development of the project, the only equipment that was needed was a computer complying with the minimum requirements of OAI. We can see the total cost of the hardware and software used in the table below:

Desktop computer	1200 €
OAI License	0 €
ns-3 License	0 €
Texmaker License	0 €
Total	**1200 €**

Table 5.1: Cost of software and hardware components.

As we can see, all the software used for this project is open source.

In addition to that, we need to take into account the amount of hours worked:

$$30 \text{ h/week} \cdot 24 \text{ weeks} = 864 \text{ h}$$

$$864 \text{ h} \cdot 15 \text{ €}/h = 12960 \text{ €}$$

This makes the total cost of the project of:

$$1200 \text{ €} + 12960 \text{ €} = \mathbf{14160} \text{ €}$$

6. Conclusions and future development

We have seen that beacons have a real impact on the throughput that WLANs can give, and it becomes more important when the density of APs increases. These results show that offloading beacon information from IEEE 802.11 to LTE would have a noticeable impact on WiFi performance by reducing the amount of time the channel is being occupied by beacon transmissions, and thus increasing the amount of time the channel is available for sending data. However, we have not been able to test the impact of the addition of another SIB on LTE.

Even though we can assume that what we gain by reducing the beacon size and increasing the beacon interval is more that what we lose adding a new SIB, it should be tested in each scenario to see if it is worth it.

As an example, we saw that when having only 7 APs it gave us a very small gain compared to when we had 85. So, in order to decide if this technique should be implemented, many different scenarios should be tested so we can see in which ones it provides a sensible gain and in which ones it does not.

Another issue that should be tested is the loss of synchronisation when increasing the beacon interval. As beacons are used for providing synchronisation to the STAs, we should take into account how long these devices can hold the synchronisation with the AP, because the more we increase the interval, the less time the channel is occupied by beacons. However, as we could see very clearly in Figure 4.1, the extra gain becomes less important the more we increase the beacon interval. Having seen that, a beacon interval between 300 and 500 TU would probably be the best option.

Another way to reduce the overhead generated by beacons that could be investigated would be sending a ”big beacon” with all the information we wanted to offload every several seconds alongside the ”small” beacons sent every hundreds of milliseconds. In this way we would not need any LTE network for offloading the beacon information and it could work for all WLANs with just an update on their software.

Opposite to this approach, we would find another interesting one which would send very small beacons, only containing timing information and the identifier of the AP in order to keep synchronisation, and sending all the other data over LTE. This would make the impact of the beacons much smaller on Wi-Fi, but it would also mean that legacy devices would not be able to connect to that WLAN.

Talking about legacy devices, an issue that should be taken into account by operators when deploying LWA is whether they allow everyone to connect to their WLANs or only their clients. In the first case they should provide backwards compatibility so tablets and laptops without LTE can connect to the network. On the other hand, in the second case this compatibility with non-LTE devices would not be necessary and then there would be no problem in reducing the amount of information carried by the beacons, as all the devices connected to the WLAN would have been first connected to the LTE network and they can receive beacon information from there.

Regarding the future development of this topic, and alongside what we have written before in this chapter, OAI should be improved and corrected in order to make it able to test the impact of introducing a new SIB. OAI is presented as a very powerful tool, but at the moment

it is not suitable for investigation as it is incomplete. It still needs a lot of work and so far it is not a good option because users will need a lot of time for configuring it, finding what features work and which ones do not work, and, if it supports their needs, modifying it in order to perform simulations. The feeling that OAI gave when using it was that it only worked in several particular cases that had been tested by the developers.

OAI, however, will be a very powerful tool when it is completed, and it will be very useful in research, but so far it is better for investigators that want to perform simulations to use ns-3, as it is more complete than OAI, with less errors and with a large community behind and a large amount of documentation available online. This makes ns-3 a good candidate for testing the impact of a new SIB in LTE. This would be a long task as the people doing it will have first to understand how ns-3 works internally and how to add this new message in a reliable way, as we did with OAI, but in this case it would bring back some results and the impact of adding a new SIB would get measured. In the case of using ns-3, a whole LWA system simulating together LTE and WLAN could be simulated, but it would require several weeks (or even months) of work.

To conclude, LWA is a very powerful technology that will enable more capacity to the UEs, which will be needed in the next years. LWA works perfectly with the idea of having less users per base station so their capacity is larger, and it does it in a very easy way that does not require a lot of changes in the hardware, so it is easy to introduce. 5G will support LWA and it will also introduce other mechanisms and technologies to reach higher throughputs in order to handle the users' demand and also make them increase their consumption. Operators, however, will develop only the features that are profitable. So, even though LTE-WLAN aggregation seems a very good option, at this moment it is difficult to say whether users will be able to experience it or not.

Bibliography

[1] 3GPP TS 36.331. *Evolved Universal Terrestrial Radio Access (E-UTRA): Radio Resource Control (RRC) Protocol Specification (Release 14)*. Technical Specification. Apr. 2017.

[2] R. Agustí et al. *LTE: Nuevas Tendencias en Comunicaciones Móviles*. Ed. by Fundación Vodafone España. 2010.

[3] S. Ahmadi. *LTE-Advanced. A Practical Systems Approach to Understanding the 3GPP LTE Releases 10 and 11 Radio Access Technologies*. Elsevier, 2014.

[4] OpenAirInterface Software Alliance. *OpenAirInterface (OAI): Towards Open Cellular Ecosystem*. English. Website. URL: http://www.openairinterface.org/?page_id=864 (visited on 08/16/2017).

[5] OpenAirInterface Software Alliance. *Third OAI General Workshop. Status and Objectives for 2017-2018*. Presentation. BUPT, Beijing, China. Apr. 2017.

[6] R. Burbidge. *LTE-WLAN Aggregation (LWA) and LTE WLAN Radio Level Integration with IPsec Tunnel (LWIP)*. Presentation at IEEE meeting. Macao. Mar. 2016.

[7] J. Casademont. *Communication Networks. Wireless Local Area Networks*. Course slides. 2014.

[8] R. Ferrús. *Advanced Mobile Communications. Chapter 1 - Introduction*. Course slides. 2015.

[9] IEEE 802.11ax Task Group. *TGax Simulation Scenarios*. doc.: IEEE 802.11-14/0980r16. Nov. 2015.

[10] NMC Consulting Group. *EMM Procedure 1. Initial Attach. Part 2. Call flow of Initial Attach*. Netmanias Technical Document. Jan. 2014.

[11] Objective Systems Inc. *ASN.1. ASN.1 Tutorial*. English. Website. URL: https://www.obj-sys.com/asn1tutorial/asn1only.html (visited on 06/27/2017).

[12] S. C. Jha et al. "Dual Connectivity in LTE Small Cell Networks". In: *Globecom 2014 Workshop - Heterogeneous and Small Cell Networks* (2014).

[13] B. Kaliski. *A Layman's Guide to a Subset of ASN.1, BER, and DER*. English. Technical Note. RSA Laboratories, June 1991. URL: http://luca.ntop.org/Teaching/Appunti/asn1.html (visited on 06/27/2017).

[14] E. Khorov, A. Kiryanov, and A. Lyakhov. "IEEE 802.11ax: How to Build High Efficiecny WLANs". In: *2015 International Conference on Engineering and Telecommunication* (2015).

[15] R. Nayanajith. *CWAP – 802.11 Mgmt Frame Types*. English. Website. Sept. 2014. URL: https://mrncciew.com/2014/09/29/cwap-802-11-mgmt-frame-types/ (visited on 08/22/2017).

[16] I. Poole. *IEEE 802.11 Wi-Fi Standards*. English. Website. URL: http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11-standards-tutorial.php (visited on 08/7/2017).

[17] ns-3 project. *ns-3 Tutorial. Release ns-3.26*. Mar. 2017.

[18] ns-3 project. *What is ns-3*. English. Website. URL: https://www.nsnam.org/overview/what-is-ns-3/ (visited on 08/16/2017).

[19] ShareTechnote. *LTE Basic Procedure. SIB Scheduling*. English. Website. URL: http://www.sharetechnote.com/html/BasicProcedure_LTE_SIB_Scheduling.html (visited on 08/5/2017).

[20] S Sirotkin. *LTE-WLAN Aggregation (LWA): Benefits and Deployment Considerations*. White Paper. Intel Corporation. 2016.

[21] A. Väisänen, P. Orava, and H. Haverinen. *Adaptive Beacon Interval in WLAN*. US Patent 7333460 B2. Feb. 2008.

[22] Linux Wireless. *en:developers:documentation:mac80211:ratecontrol:minstrel*. English. Website. Jan. 2016. URL: `https://wireless.wiki.kernel.org/en/developers/documentation/mac80211/ratecontrol/minstrel` (visited on 08/30/2017).

Appendices

A. Installation of OAI

A.1. System requirements

- **Processor:** Generation 3/4/5/6 Intel Core i5,i7; Generation 2/3/4 Intel Xeon; Intel Atom Rangely, E38xx, x5-z8300
- **OS:** Ubuntu 14.04 64-bit
- **Kernel:** Linux kernel version ¿= v.4.7 for Core Network (OpenAirCN) and v.3.19 lowlatency for Access Network (OpenAir5G).
- CPU frequency scaling disabled for emulation.
- If we want to simulate the whole system in one machinne:
  - 16 GB RAM
  - 8x Processor
  - VMware Player installed

To see the full requirements visit:
https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/OpenAirSystemRequirements
and https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/OpenAirKernelMainSetup

IMPORTANT: In order to run eNB and EPC+HSS in the same machine, one of the two parts (RAN or EPC) needs to be running on a virtual machine (we used VMware) while the other one is running on the physical machine.

This manual will be for configuring the whole system in the same physical machine, with the RAN running on the virtual machine.

A.1.1. Installation of a Linux Image in VMWare

First, we need to download the 14.04 image from: http://releases.ubuntu.com/14.04/, clicking on “64-bit PC (AMD64) desktop image”.

Second, we need to install VMware Player. We can download it from the following website: https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0.

The downloaded file will have a similar name as the following one (or even the same): “Vmware-Player-12.5.5-5234757.x86_64.bundle”. Now, we need to run the following command on a terminal:

```
sudo sh /Download_location/Vmware-Player-12.5.5-5234757.x86_64.bundle
```

We need to follow the installer’s prompts until the installation ends successfully.

To create a new virtual machine, we need to start VMware Player and click on “Create a New Virtual Machine”, then select “Use ISO image” and choose the ISO image we downloaded before.

After that, it will ask us to set a username and a password, a virtual machine name and its location in our real machine.

Next, it will ask if we want the virtual disk as a single file or splitted into multiple files. And after this part we need to set the resources we are going to give to the virtual machine. We gave it half of the processors and half of the RAM memory our computer had, so it was the following:

- Memory: 4 GB
- Processors: 4

After that we can click on “Finish” and it will start installing Ubuntu.

A.2. Download of the files

Before downloading anything we need to install Git:

```
sudo apt-get install git
```

We need to configure Git with our name and email:

```
git config --global user.name "Your Name"
git config --global user.email "Your email address"
```

To download the original software from Eurecom we need to run the following commands:

```
sudo echo -n | openssl s_client -showcerts -connect gitlab.eurecom.fr:443
2>/dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p'
>> /etc/ssl/certs/ca-certificates.crt

git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git

git clone https://gitlab.eurecom.fr/oai/openair-cn.git
```

If you want to download the files from my own GitLab from University of Surrey you first need to ask to be added into the project and then run the following:

```
git clone https://gitlab.eps.surrey.ac.uk/m09021/5g-oai
git clone https://gitlab.eps.surrey.ac.uk/m09021/Openair_CN
```

It will ask you for your username and password for https://gitlab.eps.surrey.ac.uk and then it will download it.

It should create two folders in /home/”username”/ with the two software packages.

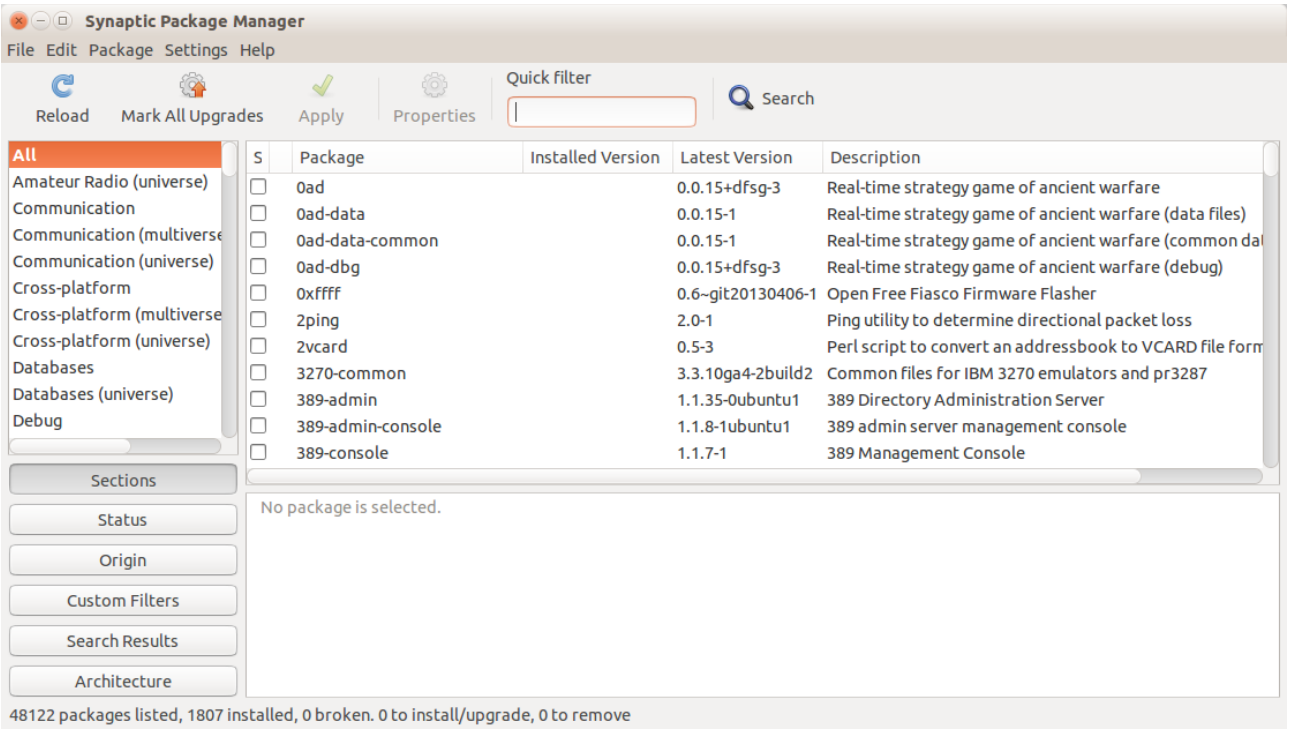


Figure A.1: Synaptic Package Manager screenshot.

A.3. Installation of the required Linux kernels

A.3.1. Linux Kernel 3.19 Low Latency

We need this Kernel for building and running the Access Network software (OpenAir5G). To download and install it we can use “Synaptic Package Manager”.

```
sudo apt-get install synaptic
```

After installing we run it and we will see a screen like in Figure A.1.

Here we can use the search tool to find “linux-image-3.19.0-80-lowlatency”, then we need to right-click on it and select “Mark for Installation”. After that, we need to do the same with “linux-headers-3.19.0-80” and “linux-headers-3.19.0-80-lowlatency”. When we have selected the three packages we can click on “Apply” and it will install them.

After this we can restart the machine and load the new Kernel in the GRUB menu. The new Kernel may not appear in the main menu. In this case, it will be in the “Advanced options for Ubuntu” submenu. To customise GRUB we can use “Grub Customizer”, we can install it running the following on a terminal:

```
 sudo add-apt-repository ppa:danielrichter2007/grub-customizer
sudo apt-get update
sudo apt-get install grub-customizer
```

After this we can run the program and customise the GRUB menu.

After this, restart the computer and select the new Kernel when asked. The following command is to check the Kernel version that a Linux machine is running:

```
uname -r
```

A.3.2. Linux Kernel 4.7

To download and install Linux Kernel 4.7 and select GTP to be compiled as a module run the commands below from the console:

```
cd /usr/src sudo wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.7.10.tar.gz

sudo tar xf linux-4.7.10.tar.gz

sudo rm linux-4.7.10.tar.gz

cd linux-4.7.10/

sudo patch -p1 < /home/magi/Openair_CN/BUILD/TOOLS/
kernel4.7-GTPv1U-LTE-dedicated-bearer.v0.patch

sudo make menuconfig #To select GTP to be compiled as a module

sudo make oldconfig #IF "MENUCONFIG" DOES NOT WORK
```

If we ran “menuconfig”, we need to go in “Device Drivers” -¿ “Network Device Support”. Then, under “Network Core Driver Support” we will find “¡ ¿ GPRS Tunneling Protocol datapath”. We need to select that option and press “M”. Afterwards, we need to save and exit.

If we ran “oldconfig”, a long-press Enter will set all the options as default. Then, we need to edit the “.config” file.

After, we need to check if the option has been activated:

```
sudo gedit .config
```

In the file we need to find “CONFIG_GTP=m”. If it is not set like this (if we used “oldconfig” we may find this option commented), we need to set it manually. And then carry on with the commands:

```
 sudo make -j‘nproc‘
sudo make modules_install
sudo make install
```

Note, if there is an error related to OpenSSL, which is probably because of the unavailability of developer OpenSSL in the system. In this case, run the following command:

```
sudo apt-get install libssl-dev
```

After this, restart the computer and select the new Kernel when asked. The following command is to check the Kernel version that a Linux machine is running:

```
uname -r
```

A.4. Installation of Openair5G

OpenAir5G will provide us with the software for simulating and emulating the devices in the E-UTRAN (Access Network).

This installation only works with 3.19 lowlatency kernel.

First, we need to install pip (a package manager for python programs). If it is missing we will have problems during the installation. We can download "get-pip.py" it from https://pip.pypa.io/en/stable/installing, and then run:

```
sudo python get-pip.py
```

After that we can go to the folder where OpenAir5G was downloaded to and start installing it:

```
cd "openair5G_dir"/

source oaienv # This command is IMPORTANT

cd cmake_targets/

./build_oai --install-external-packages --install-system-files
--install-optional-packages
```

After installing all the needed packages and files we can build the simulators:

```
./build_oai --oaisim -x # For building oaisim with S1 interface
./build_oai --oaisim --noS1 -x # For building oaisim without S1 interface
./build_oai --phy_simulators -x # For building the physical layer simulators
```

We can check if it has been built correctly by running the following commands (in cmake_targets/):

```
cd tools/
./run_enb_ue_virt_noS1 -n 10
```

This command should run until the end of the simulation, when we see "OAIEMU Ending".

A.5. Installation of OpenairCN

To install OpenAirCN we need to install three blocks: HSS, MME and SPGW. We also need to set up a database for the HSS.

We need to do it using the 4.7 Linux Kernel. If we want to run OpenAir5G and OpenAirCN in the same machine we need to install the latter in a VMWare Virtual Machine.

The system is organised as we can see in Figure A.2.

First we will need to set up the database, then install HSS, MME and SPGW. The configuration of these three modules will depend on the organisation of our system: distributed between machines or all in one machine. After, we will have to set up the connectivity with the Access Network (E-UTRAN).

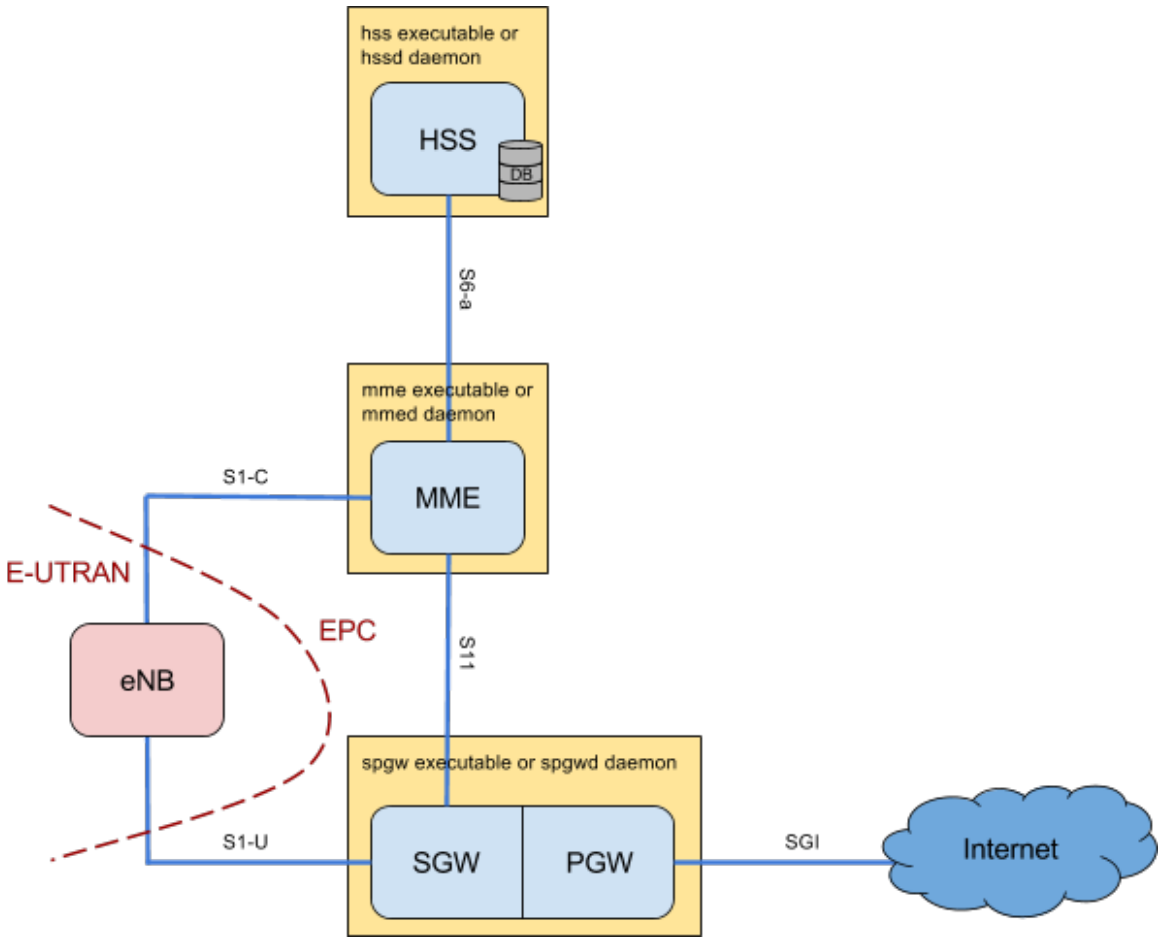


Figure A.2: Organisation diagram of OpenairCN

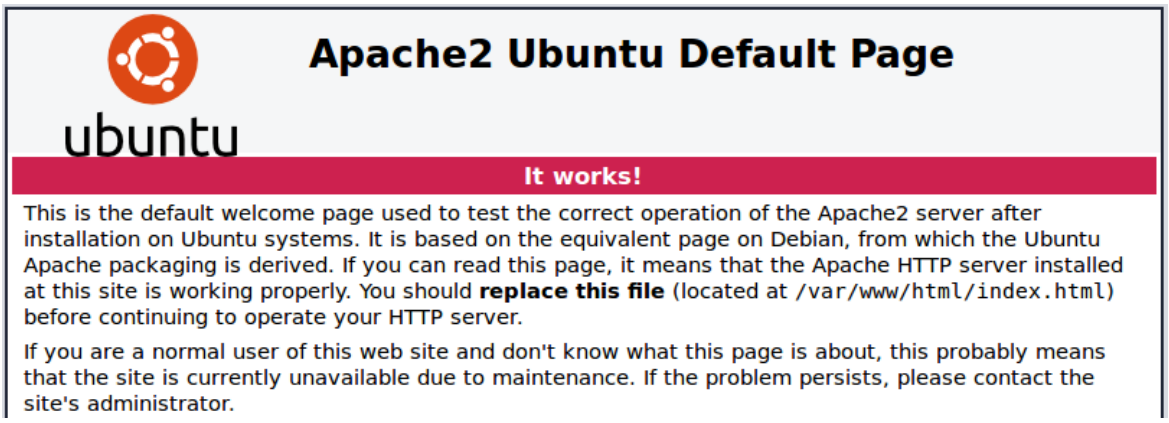


Figure A.3: Apache2 default page

A.5.1. Installation of LAMP (Linux, Apache, MySQL and PHP

To install and set the database up we need to install what is called "LAMP", so we need to install Apache, MySQL and PHP in our Linux machine. We will also install phpMyAdmin, which will help us to administrate the database.

A.5.1.1. Installation of Apache

Type the following commands:

```
sudo apt-get update
sudo apt-get install apache2
```

After this, our web server is installed. To check it we can visit our public IP address on our Internet browser (http://your_server_IP_address), in our case: http://127.0.0.1. If it works, we will se a web page starting like in Figure A.3.

A.5.1.2. Installation of MySQL

Run the following commands to install the needed software:

```
sudo apt-get install mysql-server php5-mysql
```

During the installation, it will ask for a password for the "root" MySQL user.

After the installation, we need to tell MySQL to create its database directory structure by typing:

```
sudo mysql_install_db
```

Afterwards, we should run a security script that will remove some dangerous defaults by running:

```
sudo mysql_secure_installation
```

After this, the database system is set up.

A.5.1.3. Installation of PHP

We can install PHP and some helper packages by running:

```
sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
```

This should install PHP without any problems.

After, we need to change the configuration of Apache to modify the way it serves files when a directory is requested. To do this we need to open the "dir.conf" file with root privileges:

```
sudo nano /etc/apache2/mods-enabled/dir.conf
```

It will look like this:

```
<IfModule mod_dir.c>
        DirectoryIndex index.html index.cgi index.pl index.php index.xhtml index.htm
</IfModule>
```

Code Listing A.1: Original *dir.conf* file.

And we need to modify it by moving "index.php" to the first position like the following:

```
<IfModule mod_dir.c>
        DirectoryIndex index.php index.html index.cgi index.pl index.xhtml index.htm
</IfModule>
```

Code Listing A.2: Modified *dir.conf* file.

After this, we need to restart the Apache web server in order to apply our changes:

```
sudo service apache2 restart
```

In order to test that our system is configured properly for PHP, we can create a very basic PHP script. We will call it "info.php" and it must be saved at */var/www/html/*. We can create the file at the location by typing:

```
sudo nano /var/www/html/info.php
```

This will open an empty file. We will fill and save the file with the following text, which is valid PHP code:

```
<?php
phpinfo();
?>
```

Code Listing A.3: Code that we need to write in *info.php*.

Now we can test if our web server can correctly display content generated by a PHP script. To do it, we just have to visit the following webpage: http://your_server_IP_address/info.php (in our case http://127.0.0.1/info.php).

If we can see something similar to Figure A.4 it means that our PHP is working as expected.

PHP Version 5.5.9-1ubuntu4.21

System	Linux magi 4.9.0-040900-generic #201612111631 SMP Sun Dec 11 21:33:00 UTC 2016 x86_64
Build Date	Feb 9 2017 20:54:17
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini

Figure A.4: Example of *info.php*.

After this test we should remove the file because it could give information about our server to unauthorised users. We just need to remove the "info.php" file that we created before:

```
sudo rm /var/www/html/info.php
```

A.5.1.4. Installation of phpMyAdmin

It is a tool that will allow us to manage our database through a web interface.

To install the files in our system we need to run the following:

```
sudo apt-get install phpmyadmin
```

It will ask us a few questions during the installation:

- For the server selection, choose *apache2*. To select it press **SPACE** when the cursor is on it and **ENTER** afterwards.
- Select "yes" when asked if you want to use *dbconfig-common* to set up the database.
- It will ask for the database administrator's password.
- It will ask for a password for the phpMyAdmin application itself.

After this, the only thing we need to do is explicitly enable the *php5-mcrypt* extension and restart the server by typing: `sudo php5enmod mcrypt`
`sudo service apache2 restart`

We can now access the web interface by visiting the next web page:
http://your_server_IP_address/phpmyadmin (in our case http://127.0.0.1/phpmyadmin).

It should look like Figure A.5.

We can now log into the interface using "root" username and the MySQL admin password that we set up during the installation.

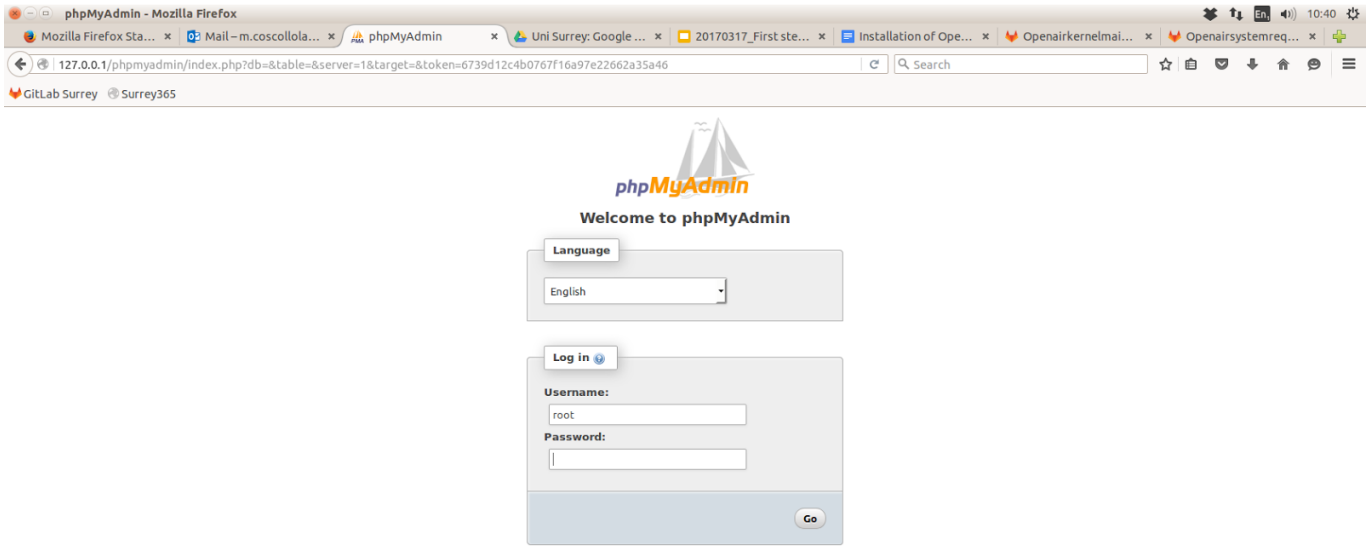


Figure A.5: Example of phpMyAdmin log in page.

A.5.2. Installation of HSS

First we need to run the following command in order to install the packages required for a proper functioning of HSS:

```
cd "openairCN_dir"/
source oaienv # This command is IMPORTANT
cd SCRIPTS/
./build_hss -i
```

When asking to install freeDiameter we should say “yes” as it is not installed.

We also need to set a FQDN (Fully Qualified Domain Name) for the HSS. An easy way to do that is to fill the */etc/hosts* file:

```
sudo nano /etc/hosts
```

And then add the HSS in this way:

```
127.0.0.1      localhost
127.0.1.1      ubuntu.openair4G.eur ubuntu
127.0.33.1     hss.openair4G.eur    hss
```

Code Listing A.4: Edited */etc/hosts*.

After this we need to configure the HSS. First we need to copy the HSS configuration file template to /usr/local/etc/oai/ using the following commands:

```
sudo mkdir -p /usr/local/etc/oai/freeDiameter
sudo cp ../ETC/hss.conf /usr/local/etc/oai
```

Now we can customise the copied hss.conf file (in */usr/local/etc/oai/*), which will look like this:

```
HSS :
{
## MySQL mandatory options
MYSQL_server = "127.0.0.1";  # HSS S6a bind address
MYSQL_user   = "root";       # Database server login
MYSQL_pass   = "1234admin";  # Database server password set during the
    installation
MYSQL_db     = "oai_db";     # Your database name
## HSS options
OPERATOR_key = "1006020f0a478bf6b699f15c062e42b3"; # OP key matching your
    database
RANDOM = "true";             # True random or only pseudo random (for subscriber
    vector generation)
## Freediameter options
FD_conf = "/usr/local/etc/oai/freeDiameter/hss_fd.conf";
};
```

Code Listing A.5: Modified *hss.conf*.

For the MYSQL_server field, we wrote our own local address as the database is located in the same host.

After this, we need to copy two more files:

```
sudo cp "openairCN_dir"/ETC/acl.conf "openairCN_dir"/ETC/hss_fd.conf
/usr/local/etc/oai/freeDiameter
```

It should not be necessary to customise these files, but you need to make sure that we can find the following two lines in hss_fd.conf:

```
Identity = "hss.openair4G.eur"
Realm = "openair4G.eur"
```

Code Listing A.6: Lines that need to be found in *hss_fd.conf*.

After this we need to generate the certificates for the HSS:

```
cd "openairCN_dir"/SCRIPTS/
./check_hss_s6a_certificate /usr/local/etc/oai/freeDiameter hss.openair4G.eur
```

After this we can build HSS in two different ways: an executable that can run in a terminal or a daemon that would run in the background.

```
./build_hss --clean --debug # For the executable
./build_hss --clean --debug --daemon # For the daemon
```

A.5.3. Installation of MME

To install the MME we need to run the following command (we are still in *"openairCN_dir"/SCRIPTS/*):

```
./build_mme -i # It will install the required software in our host
```

When asking whether to install *asn1c rev 1516* and *liblfds7.1.0* it is necessary to say yes in order to be able to build MME afterwards.

After that, we need to copy the configuration files as we did when installing the HSS:

```
sudo mkdir -p /usr/local/etc/oai/freeDiameter # If we haven't done it before
sudo cp "openairCN_dir"/ETC/mme.conf /usr/local/etc/oai
```

Now we need to customise the copied configuration file. We only modified the following part:

```
NETWORK_INTERFACES :
    {
        # MME binded interface for S1-C or S1-MME communication (S1AP), can be
    ethernet interface, virtual ethernet interface, we don't advise wireless
    interfaces
        MME_INTERFACE_NAME_FOR_S1_MME      = "vmnet8"; # YOUR NETWORK CONFIG HERE
       MME_IPV4_ADDRESS_FOR_S1_MME        = "172.16.53.1/24";
       MME_INTERFACE_NAME_FOR_S1_MME      = "vmnet8"; # YOUR NETWORK CONFIG HERE
       MME_IPV4_ADDRESS_FOR_S1_MME        = "172.16.53.1/24";
       MME_INTERFACE_NAME_FOR_S1_MME      = "vmnet8"; # YOUR NETWORK CONFIG HERE
       MME_IPV4_ADDRESS_FOR_S1_MME        = "172.16.53.1/24";
       MME_INTERFACE_NAME_FOR_S1_MME      = "vmnet8"; # YOUR NETWORK CONFIG HERE
       MME_IPV4_ADDRESS_FOR_S1_MME        = "172.16.53.1/24";
        # MME binded interface for S11 communication (GTPV2-C)
        MME_INTERFACE_NAME_FOR_S11_MME     = "lo"; # YOUR NETWORK CONFIG HERE
        MME_IPV4_ADDRESS_FOR_S11_MME       = "127.0.11.1/8";
        MME_PORT_FOR_S11_MME               = 2123;
    };
```

```
    ...
    S-GW\_LIST\_SELECTION = (
        {ID="tac-lb01.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ;
        SGW_IPV4_ADDRESS_FOR_S11="127.0.11.2/8";},
        ... # ALL THE IPS ARE SET TO THE SAME VALUE
        {ID="tac-lb0f.tac-hb00.tac.epc.mnc093.mcc208.3gppnetwork.org" ;
        SGW_IPV4_ADDRESS_FOR_S11="127.0.11.2/8";}
    );"
```

Code Listing A.7: Example of modified *mme.conf*.

In the NETWORK_INTERFACES field, we need to set the IP that the MME will have for the S1-C and S11 interfaces. In the case of the S1-C, we put the IP address of the machine when using vmnet8 interface (the one connecting the physical machine with the virtual one); for S11, we put an address inside the range of addresses for the lo interface (local loop). In the S-GW_LIST_SELECTION we set all the addresses to 127.0.11.2, another address in the range of addresses for the lo interface, which will be the address that the S-GW will have (so we will have to write it again when configuring spgw.conf).

After, we need to copy another file:

```
sudo cp "openairCN_dir"/ETC/mme_fd.conf /usr/local/etc/oai/freeDiameter
```

Then we need to customise the copied “mme_fd.conf”. We need to make sure that these three lines are correct:

```
Identity = "ubuntu.openair4G.eur";
Realm = "openair4G.eur";
ConnectPeer= "hss.openair4G.eur" { ConnectTo = "127.0.33.1"; No_SCTP ; No_IPv6;
    Prefer_TCP; No_TLS; port = 3868;  realm = "openair4G.eur";};
```

Code Listing A.8: Lines that need to be checked in *mme_fd.conf*.

We need to put the hostname and address that we set for the HSS host in /etc/hosts.

After that, we can build the MME as an executable or as a daemon following the next commands:

```
./build_mme --clean # For the executable
./build_mme --clean --daemon # For the daemon
```

This command will compile *oai_mme* (for the executable) or *oai_mmed* (for the daemon). After, we need to generate the certificate for the MME:

```
./check_mme_s6a_certificate /usr/local/etc/oai/freeDiameter/ ubuntu.openair4G.eur
```

A.5.4. Installation of SPGW

To install the required software we need to run the following command (in “openairCN_dir”/SCRIPTS/):

```
./build_spgw -i
```

When asking to ask *libgtpnl* say “yes”.

After this, we need to copy the configuration files:

```
sudo mkdir -p /usr/local/etc/oai # If we haven't done it before
sudo cp "openairCN_dir"/ETC/spgw.conf /usr/local/etc/oai
```

Then, we need to customise the copied configuration files like the following:

```
S-GW :
{
   NETWORK_INTERFACES :
   {
      # S-GW binded interface for S11 communication (GTPV2-C), if none selected
    the ITTI message interface is used
      SGW_INTERFACE_NAME_FOR_S11 = "lo"; # YOUR NETWORK CONFIG HERE
      SGW_IPV4_ADDRESS_FOR_S11 = "127.0.11.2/8"; # YOUR NETWORK CONFIG HERE

      # S-GW binded interface for S1-U communication (GTPV1-U) can be ethernet
    interface, virtual ethernet interface, we don't advise wireless interfaces
      SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP = "vmnet8"; # YOUR NETWORK CONFIG HERE
    , USE "lo" if S-GW run on eNB host
      SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP = "172.16.53.1/24";# YOUR NETWORK CONFIG
     HERE
      SGW_IPV4_PORT_FOR_S1U_S12_S4_UP = 2152; # PREFER NOT CHANGE UNLESS YOU KNOW
     WHAT YOU ARE DOING
      # S-GW binded interface for S5 or S8 communication, not implemented, so
    leave it to none
      SGW_INTERFACE_NAME_FOR_S5_S8_UP = "none"; # DO NOT CHANGE (NOT IMPLEMENTED)
      SGW_IPV4_ADDRESS_FOR_S5_S8_UP ="0.0.0.0/24"; # DO NOT CHANGE (NOT
    IMPLEMENTED)
   };
   ...
}
P-GW =
{
   NETWORK_INTERFACES :
   {
      # P-GW binded interface for S5 or S8 communication, not implemented, so
    leave it to none
      PGW_INTERFACE_NAME_FOR_S5_S8 = "none"; # DO NOT CHANGE (NOT IMPLEMENTED YET
    )
            # P-GW binded interface for SGI (egress/ingress internet traffic)
            PGW_INTERFACE_NAME_FOR_SGI = "eth0"; # STRING, YOUR NETWORK CONFIG
    HERE
            PGW_MASQUERADE_SGI = "yes"; # STRING, {"yes", "no"}. YOUR NETWORK
    CONFIG HERE, will do NAT for you if you put "yes".
            UE_TCP_MSS_CLAMPING = "no"; # STRING, {"yes", "no"}.
   };
   ...
   # DNS address communicated to UEs
   DEFAULT_DNS_IPV4_ADDRESS = "131.227.100.5"; # YOUR NETWORK CONFIG HERE
   DEFAULT_DNS_SEC_IPV4_ADDRESS = "131.227.130.5"; # YOUR NETWORK CONFIG HERE
   ...
}
```

Code Listing A.9: Modified *spgw.conf*.

For the S11 interface, we set that the address of the S-GW will be the same we told the MME before (127.0.11.2) and lo as the interface. Then, for the S1-U interface we set again *vmnet8* and the IP address of the machine for this interface, as we did for S1-C in MME.

As our machine gets the connection to Internet through *eh0*, we set that for the SGI interface

name. We got an error when running if *PGW_MASQUERADE_SGI* was set to “no”, so we set it to “yes”.

We can check the default DNS address with the following command:

```
nm-tool | grep DNS
```

After that, we can build the SPGW as an executable or as a daemon following the next commands:

```
./build_spgw --clean # For the executable
./build_spgw --clean --daemon # For the daemon
```

A.6. First run of OpenairCN

A.6.0.1. Running HSS

Always run HSS first.

The first time we run it we need to generate the database using the following command:

```
./run_hss -i "openairCN_dir"/SRC/OAI_HSS/db/oai_db.sql
```

After generating the database, we need to add the host where we have the MME in *mmeidentity* table (in our case, “ubuntu.openair4G.eur”). We can see this table in Figure A.6.

We can know the hostname of the machine where MME is installed on with the following command:

```
hostname -f
```

For the following runs, we only need to use:

```
./run_hss
```

After running HSS, we should see the terminal like in Figure A.7.

After that, we can run MME.

+ Options

←T→	idmmeidentity	mmehost	mmerealm	UE-Reachability Indicates whether the MME supports UE Reachability Notifcation
Edit Copy Delete	2	mme2.openair4G.eur	openair4G.eur	0
Edit Copy Delete	1	nano.openair4G.eur	openair4G.eur	0
Edit Copy Delete	5	abeille.openair4G.eur	openair4G.eur	0
Edit Copy Delete	4	ubuntu.openair4G.eur	openair4G.eur	0
Edit Copy Delete	3	mme3.openair4G.eur	openair4G.eur	0
Edit Copy Delete	6	calisson.openair4G.eur	openair4G.eur	0

Figure A.6: *mmeidentity* table.

```
magi@ubuntu: ~/Openair_CN/SCRIPTS 100x29
04/05/17,11:54:52.858590  NOTI     Local applications ..... : (none)
04/05/17,11:54:52.858591  NOTI     Flags : - IP ........... : Enabled
04/05/17,11:54:52.858608  NOTI             - IPv6 ......... : DISABLED
04/05/17,11:54:52.858609  NOTI             - Relay app .... : DISABLED
04/05/17,11:54:52.858631  NOTI             - TCP .......... : Enabled
04/05/17,11:54:52.858633  NOTI             - SCTP ......... : DISABLED
04/05/17,11:54:52.858634  NOTI             - Pref. proto .. : TCP
04/05/17,11:54:52.858643  NOTI             - TLS method ... : Separate port
04/05/17,11:54:52.858649  NOTI     TLS :   - Certificate .. : /usr/local/etc/oai/freeDiameter/hss.ce
rt.pem
04/05/17,11:54:52.858652  NOTI             - Private key .. : /usr/local/etc/oai/freeDiameter/hss.ke
y.pem
04/05/17,11:54:52.858653  NOTI             - CA (trust) ... : /usr/local/etc/oai/freeDiameter/hss.ca
cert.pem (1 certs)
04/05/17,11:54:52.858655  NOTI             - CRL .......... : (none)
04/05/17,11:54:52.858657  NOTI             - Priority ..... : (default: 'NORMAL')
04/05/17,11:54:52.858659  NOTI             - DH bits ...... : 1024
04/05/17,11:54:52.858661  NOTI     Origin-State-Id ........ : 1491389692
04/05/17,11:54:52.858672  NOTI   Loaded extensions: '/usr/lib/freeDiameter/acl_wl.fdx'[/usr/local/et
c/oai/freeDiameter/acl.conf], loaded
04/05/17,11:54:52.858674  NOTI   Loaded extensions: '/usr/lib/freeDiameter/dict_nas_mipv6.fdx'[(no c
onfig file)], loaded
04/05/17,11:54:52.858683  NOTI   Loaded extensions: '/usr/lib/freeDiameter/dict_s6a.fdx'[(no config
file)], loaded
04/05/17,11:54:52.858702   DBG   Core state: 1 -> 2
04/05/17,11:54:52.860392  NOTI   Local server address(es): 172.16.53.128{---L-}
04/05/17,11:54:52.860443   DBG   Core state: 2 -> 3
Initializing s6a layer: DONE
```

Figure A.7: Example of terminal when running HSS.

A.6.1. Running MME

To run MME we need to use the following command:

```
./run_mme
```

If the connection between the MME and HSS is successful we will see ***STATE_OPEN*** in the HSS terminal.

In case of getting *STATE_ZOMBIE* or some other state that is not *STATE_OPEN* we need to check if our host is registered to the database and if the addresses and interfaces in the configuration files match each other.

A.6.2. Running SPGW

We can run SPGW using the next command:

```
./run_spgw
```

When SPGW is running, so the whole Core Network is running, we can connect eNBs to it using the Openair5G software.

A.7. First run of Openair5G

A.7.0.1. Editing configuration file

Before running *oaisim* we need to edit the configuration file in order to make it connect to the Evolved Packet Core. The file can be modified in the following way:

```
gedit "openair5G_dir"/targets/PROJECTS/GENERIC-LTE-EPC/CONF/
enb.band7.generic.oaisim.local_mme.conf
```

And then it should look like the following:

```
// Tracking area code, 0x0000 and 0xfffe are reserved values
tracking_area_code  =  "1";
mobile_country_code =  "208";
mobile_network_code =  "93";
...
////////// MME parameters:
mme_ip_address  = ( { ipv4           = "172.16.53.1";
                      ipv6       = "192:168:30::17";
                      Active     = "yes";
                      preference = "ipv4";
                    }
                    );
NETWORK_INTERFACES :
{
    ENB_INTERFACE_NAME_FOR_S1_MME    = "eth0";
    ENB_IPV4_ADDRESS_FOR_S1_MME      = "172.16.53.129/24";
    ENB_INTERFACE_NAME_FOR_S1U       = "eth0";
    ENB_IPV4_ADDRESS_FOR_S1U         = "172.16.53.129/24";
    ENB_PORT_FOR_S1U                 = 2153; # Spec 2152
};
```

Code Listing A.10: Example of *enb.band7.generic.oaisim.local_mme.conf*.

It is very important that the *tracking_area_code*, *mobile_country_code* and *mobile_network_code* match the ones in the EPC configuration. If not, the system will not work.

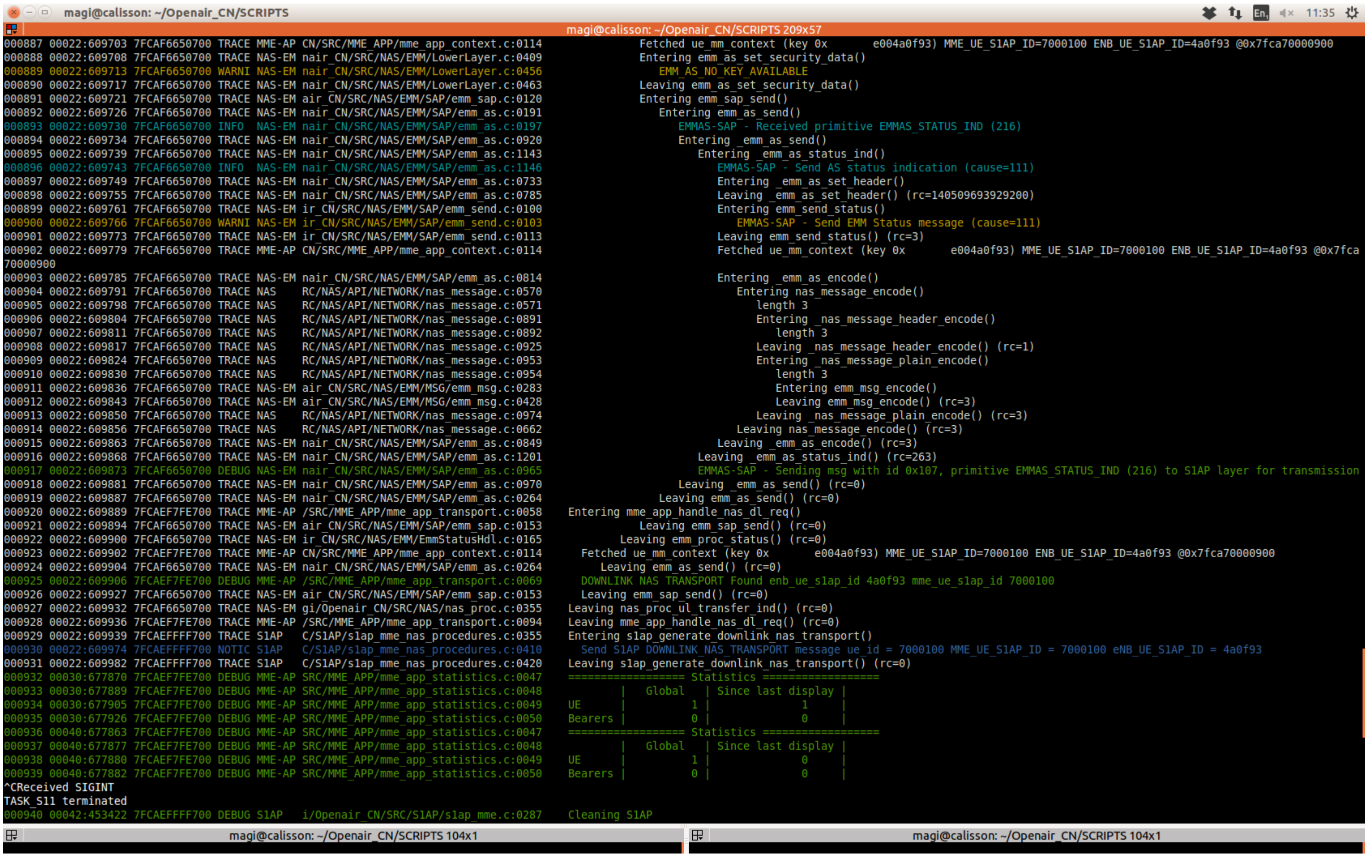
Afterwards, we need to set the IP address of the MME in the *mme_ip_address* section and set the interfaces and the IP address that oaisim will have in order to use S1-C and S1-U.

A.7.1. Running oaisim

When the configuration file is properly set, we can run oaisim in the following way:

```
./run_enb_ue_virt_s1 -c "openair5G_dir"/targets/PROJECTS/GENERIC-LTE
-EPC/CONF/enb.band7.generic.oaisim.local_mme.conf
```

We can check if the eNB connects to the MME if we can see the following outputs on the terminal where MME is running.



```
000887 00022:609703 7FCAF6650700 TRACE MME-AP CN/SRC/MME_APP/mme_app_context.c:0114               Fetched ue_mm_context (key 0x       e004a0f93) MME_UE_S1AP_ID=7000100 ENB_UE_S1AP_ID=4a0f93 @0x7fca70000900
000888 00022:609708 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/LowerLayer.c:0409               Entering emm_as_set_security_data()
000889 00022:609713 7FCAF6650700 WARNI NAS-EM nair_CN/SRC/NAS/EMM/LowerLayer.c:0456                  EMM_AS_NO_KEY_AVAILABLE
000890 00022:609717 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/LowerLayer.c:0463               Leaving emm_as_set_security_data()
000891 00022:609721 7FCAF6650700 TRACE NAS-EM air_CN/SRC/NAS/EMM/SAP/emm_sap.c:0120               Entering emm_sap_send()
000892 00022:609726 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0191                  Entering emm_as_send()
000893 00022:609730 7FCAF6650700 INFO  NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0197                     EMMAS-SAP - Received primitive EMMAS_STATUS_IND (216)
000894 00022:609734 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0920                     Entering _emm_as_send()
000895 00022:609739 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:1143                        Entering _emm_as_status_ind()
000896 00022:609743 7FCAF6650700 INFO  NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:1146                           EMMAS-SAP - Send AS status indication (cause=111)
000897 00022:609749 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0733                           Entering _emm_as_set_header()
000898 00022:609755 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0785                           Leaving _emm_as_set_header() (rc=140509693929200)
000899 00022:609761 7FCAF6650700 TRACE NAS-EM ir_CN/SRC/NAS/EMM/SAP/emm_send.c:0100                           Entering emm_send_status()
000900 00022:609766 7FCAF6650700 WARNI NAS-EM ir_CN/SRC/NAS/EMM/SAP/emm_send.c:0103                              EMMAS-SAP - Send EMM Status message (cause=111)
000901 00022:609773 7FCAF6650700 TRACE NAS-EM ir_CN/SRC/NAS/EMM/SAP/emm_send.c:0113                           Leaving emm_send_status() (rc=3)
000902 00022:609779 7FCAF6650700 TRACE MME-AP CN/SRC/MME_APP/mme_app_context.c:0114                           Fetched ue_mm_context (key 0x       e004a0f93) MME_UE_S1AP_ID=7000100 ENB_UE_S1AP_ID=4a0f93 @0x7fca
70000900
000903 00022:609785 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0814                           Entering _emm_as_encode()
000904 00022:609791 7FCAF6650700 TRACE NAS    RC/NAS/API/NETWORK/nas_message.c:0570                              Entering nas_message_encode()
000905 00022:609798 7FCAF6650700 TRACE NAS    RC/NAS/API/NETWORK/nas_message.c:0571                                 length 3
000906 00022:609804 7FCAF6650700 TRACE NAS    RC/NAS/API/NETWORK/nas_message.c:0891                                 Entering _nas_message_header_encode()
000907 00022:609811 7FCAF6650700 TRACE NAS    RC/NAS/API/NETWORK/nas_message.c:0892                                    length 3
000908 00022:609817 7FCAF6650700 TRACE NAS    RC/NAS/API/NETWORK/nas_message.c:0925                                 Leaving _nas_message_header_encode() (rc=1)
000909 00022:609824 7FCAF6650700 TRACE NAS    RC/NAS/API/NETWORK/nas_message.c:0953                                 Entering _nas_message_plain_encode()
000910 00022:609830 7FCAF6650700 TRACE NAS    RC/NAS/API/NETWORK/nas_message.c:0954                                    length 3
000911 00022:609836 7FCAF6650700 TRACE NAS-EM air_CN/SRC/NAS/EMM/MSG/emm_msg.c:0283                                    Entering emm_msg_encode()
000912 00022:609843 7FCAF6650700 TRACE NAS-EM air_CN/SRC/NAS/EMM/MSG/emm_msg.c:0428                                    Leaving emm_msg_encode() (rc=3)
000913 00022:609850 7FCAF6650700 TRACE NAS    RC/NAS/API/NETWORK/nas_message.c:0974                                 Leaving _nas_message_plain_encode() (rc=3)
000914 00022:609856 7FCAF6650700 TRACE NAS    RC/NAS/API/NETWORK/nas_message.c:0662                              Leaving nas_message_encode() (rc=3)
000915 00022:609863 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0849                           Leaving _emm_as_encode() (rc=3)
000916 00022:609868 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:1201                        Leaving _emm_as_status_ind() (rc=263)
000917 00022:609873 7FCAF6650700 DEBUG NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0965                        EMMAS-SAP - Sending msg with id 0x107, primitive EMMAS_STATUS_IND (216) to S1AP layer for transmission
000918 00022:609881 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0970                     Leaving _emm_as_send() (rc=0)
000919 00022:609887 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0264                  Leaving emm_as_send() (rc=0)
000920 00022:609889 7FCAEF7FE700 TRACE MME-AP /SRC/MME_APP/mme_app_transport.c:0058    Entering mme_app_handle_nas_dl_req()
000921 00022:609894 7FCAF6650700 TRACE NAS-EM air_CN/SRC/NAS/EMM/SAP/emm_sap.c:0153               Leaving emm_sap_send() (rc=0)
000922 00022:609900 7FCAF6650700 TRACE NAS-EM ir_CN/SRC/NAS/EMM/EmmStatusHdl.c:0165            Leaving emm_proc_status() (rc=0)
000923 00022:609902 7FCAEF7FE700 TRACE MME-AP CN/SRC/MME_APP/mme_app_context.c:0114      Fetched ue_mm_context (key 0x       e004a0f93) MME_UE_S1AP_ID=7000100 ENB_UE_S1AP_ID=4a0f93 @0x7fca70000900
000924 00022:609904 7FCAF6650700 TRACE NAS-EM nair_CN/SRC/NAS/EMM/SAP/emm_as.c:0264         Leaving emm_as_send() (rc=0)
000925 00022:609906 7FCAEF7FE700 DEBUG MME-AP /SRC/MME_APP/mme_app_transport.c:0069      DOWNLINK NAS TRANSPORT Found enb_ue_s1ap_id 4a0f93 mme_ue_s1ap_id 7000100
000926 00022:609927 7FCAF6650700 TRACE NAS-EM air_CN/SRC/NAS/EMM/SAP/emm_sap.c:0153      Leaving emm_sap_send() (rc=0)
000927 00022:609932 7FCAF6650700 TRACE NAS-EM gi/Openair_CN/SRC/NAS/nas_proc.c:0355    Leaving nas_proc_ul_transfer_ind() (rc=0)
000928 00022:609936 7FCAEF7FE700 TRACE MME-AP /SRC/MME_APP/mme_app_transport.c:0094    Leaving mme_app_handle_nas_dl_req() (rc=0)
000929 00022:609939 7FCAEFFFF700 TRACE S1AP   C/S1AP/s1ap_mme_nas_procedures.c:0355    Entering s1ap_generate_downlink_nas_transport()
000930 00022:609974 7FCAEFFFF700 NOTIC S1AP   C/S1AP/s1ap_mme_nas_procedures.c:0410      Send S1AP DOWNLINK_NAS_TRANSPORT message ue_id = 7000100 MME_UE_S1AP_ID = 7000100 eNB_UE_S1AP_ID = 4a0f93
000931 00022:609982 7FCAEFFFF700 TRACE S1AP   C/S1AP/s1ap_mme_nas_procedures.c:0420    Leaving s1ap_generate_downlink_nas_transport() (rc=0)
000932 00030:677870 7FCAEF7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0047    ================== Statistics ==================
000933 00030:677889 7FCAEF7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0048            |   Global   | Since last display |
000934 00030:677905 7FCAEF7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0049    UE      |          1 |             1     |
000935 00030:677926 7FCAEF7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0050    Bearers |          0 |             0     |
000936 00040:677863 7FCAEF7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0047    ================== Statistics ==================
000937 00040:677877 7FCAEF7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0048            |   Global   | Since last display |
000938 00040:677880 7FCAEF7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0049    UE      |          1 |             0     |
000939 00040:677882 7FCAEF7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0050    Bearers |          0 |             0     |
^CReceived SIGINT
TASK_S11 terminated
000940 00042:453422 7FCAEFFFF700 DEBUG S1AP   i/Openair_CN/SRC/S1AP/s1ap_mme.c:0287    Cleaning S1AP
```

Figure A.8: Example MME terminal output when the Access Network is running.

B. MSC Translator

Here is the code for MSC_translator.py.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys

def handle_proto(outfile, line):
  text = line[line.find("[PROTO] ")+8:]
  for s in text.split():
    if s.isdigit():
      outfile.write(s)
    else:
      outfile.write(" [label=\""+s+"\"]")

def handle_event(outfile, line):
  text = line[line.find("[EVENT] ")+8:]
  outfile.write("\t—— [ label = \""+text+"\" ];\n")

def handle_message(outfile, line):
  text = line[line.find("[MESSAGE] ")+10:]
  splitted = text.split(" ", 4)
  outfile.write("\t")#tab
  for i in range(0,3):
    outfile.write(splitted[i])
  outfile.write(" [ label = \""+splitted[4]+"\"];\n")

############# MAIN STARTS HERE #############
num_args = len(sys.argv)
protoflag = 0 #flag to know if proto has been the last line, for deciding comma
    or semicolon
if num_args != 2:  # the program name and the arguments
  # stop the program and print an error message
  sys.exit("Must provide one file to translate")

for i in range(1,num_args):
  print sys.argv[i] #sys.argv[i] is the arg value

#Create new file and start writing
outfile = open("MSCoutput.txt", "w")
outfile.write("# MSC for " + sys.argv[1]+"\n")
outfile.write("msc {\n")

#Open input file
infile = open(sys.argv[i], "r")

#Reading lines of original file and doing the proper actions
for line in infile:
  line = line.replace("\n","")
  if "PROTO" in line:
    if protoflag == 1:
      outfile.write(",")
    else:
      outfile.write("\t") #we add tab if it's the first proto
    handle_proto(outfile, line)
    protoflag = 1
  elif "EVENT" in line:
```

```
    if protoflag == 1:
      outfile.write(";\n\n")
      protoflag = 0
    handle_event(outfile, line)
  elif "MESSAGE" in line:
    if protoflag == 1:
      outfile.write(";\n\n")
      protoflag = 0
    handle_message(outfile, line)

#Close created file
outfile.write("\n}\n")
outfile.close()
```

Code Listing B.1: msc_translator.py file.

C. Beacon contents

Order	Information	Notes
1	Timestamp	
2	Beacon interval	
3	Capability Information	
4	Service Set Identifier (SSID)	If dot11MeshActivated is true, the SSID element is the wildcard value as described in 9.4.2.2.
5	Supported Rates and BSS Membership Selectors	
6	DSSS Parameter Set	The element is optionally present. The DSSS Parameter Set element is present within Beacon frames generated by STAs using Clause 15, Clause 16, and Clause 18 PHYs. The element is present within Beacon frames generated by STAs using a Clause 19 PHY in the 2.4 GHz band.
7	CF Parameter Set	The CF Parameter Set element is present only within Beacon frames generated by APs supporting a PCF. This element is not present if dot11HighThroughputOption-Implemented is true and the Dual CTS Protection field of the HT Operation element is 1.
8	IBSS Parameter Set	The IBSS Parameter Set element is present only within Beacon frames generated by IBSS STAs.
9	Traffic indication map (TIM)	The TIM element is present only within Beacon frames generated by APs or mesh STAs.

Order	Information	Notes
10	Country	The Country element is present if dot11MultiDomainCapabilityActivated is true or dot11SpectrumManagementRequired is true or dot11RadioMeasurementActivated is true.
11	Power Constraint	The Power Constraint element is present if dot11SpectrumManagementRequired is true and is optionally present if dot11RadioMeasurementActivated is true.
12	Channel Switch Announcement	Channel Switch Announcement element is optionally present if dot11SpectrumManagementRequired is true.
13	Quiet	The Quiet element is optionally present if dot11SpectrumManagementRequired is true or dot11RadioMeasurementActivated is true.
14	IBSS DFS	IBSS DFS element is present if dot11SpectrumManagementRequired is true in an IBSS.
15	TPC Report	The TPC Report element is present if dot11SpectrumManagementRequired is true or dot11RadioMeasurementActivated is true.
16	ERP	The ERP element is present within Beacon frames generated by STAs using extended rate PHYs (ERPs) defined in Clause 18 and is optionally present in other cases.
17	Extended Supported Rates and BSS Membership Selectors	The Extended Supported Rates and BSS Membership Selectors element is present if there are more than eight supported rates, and it is optional otherwise.
18	RSN	The RSNE is present within Beacon frames generated by STAs that have dot11RSNAActivated equal to true.
19	BSS Load	The BSS Load element is present if dot11QosOption-Implemented and dot11QBSSLoadImplemented are both true.
20	EDCA Parameter Set	The EDCA Parameter Set element is present if dot11QosOptionImplemented is true, and dot11MeshActivated is false, and the QoS Capability element is not present.
21	QoS Capability	The QoS Capability element is present if dot11QosOption-Implemented is true, and dot11MeshActivated is false, and EDCA Parameter Set element is not present.
22	AP Channel Report	If dot11RMAPChannelReportActivated is true, one AP Channel Report element is present for each operating class that has at least 1 channel to report.
23	BSS Average Access Delay	The BSS Average Access Delay element is present if dot11RMBSSAverageAccessDelayActivated is true and the value of the AP Average Access Delay field is not equal to 255 (measurement not available); otherwise, the BSS Average Access Delay element is optionally present if dot11RMBSSAverageAccessDelayActivated is true.
24	Antenna	The Antenna element is present if dot11RMAntennaInformationActivated is true and the value of the Antenna ID field is not equal to 0 (unknown antenna); otherwise, the Antenna element is optionally present if dot11RMAntennaInformationActivated is true.

Order	Information	Notes
25	BSS Available Admission Capacity	The BSS Available Admission Capacity element is present if dot11RMBSSAvailableAdmissionCapacityActivated is true with the following exceptions: 1) when Available Admission Capacity Bitmask equals 0 (Available Admission Capacity List contains no entries), or 2) when the BSS Load element is present and the Available Admission Capacity Bitmask states that only AC_VO is present in the Available Admission Capacity List field.
26	BSS AC Access Delay	The BSS AC Access Delay element is present if dot11RMBSSAverageAccessDelayActivated is true and at least one field of the element is not equal to 255 (measurement not available); otherwise, the BSS AC Access Delay element is optionally present if dot11RMBSSAverageAccessDelayActivated is true.
27	Measurement Pilot Transmission	The Measurement Pilot Transmission element is present if dot11RMMeasurementPilotActivated is a value between 2 and 7.
28	Multiple BSSID	One or more Multiple BSSID elements are present if dot11RMMeasurementPilotActivated is a value between 2 and 7 and the AP is a member of a multiple BSSID set (see 11.11.14) with two or more members, or if dot11MultiBSSIDActivated is true, or if dot11InterworkingServiceActivated is true and the AP is a member of a multiple BSSID set with two or more members and at least one dot11GASAdvertisementID exists.
29	RM Enabled Capabilities	RM Enabled Capabilities element is present if dot11RadioMeasurementActivated is true.
30	Mobility Domain	The Mobility Domain element (MDE) is present if dot11FastBSSTransitionActivated is true.
31	DSE registered location	The DSE Registered Location element is present if dot11LCIDSERequired is true.
32	Extended Channel Switch Announcement	The Extended Channel Switch Announcement element is optionally present if dot11ExtendedChannelSwitchActivated is true.
33	Supported Operating Classes	The Supported Operating Classes element is present if dot11ExtendedChannelSwitchActivated is true. The Supported Operating Classes element is optionally present if dot11TVHTOptionImplemented is true.
34	HT Capabilities	The HT Capabilities element is present when dot11HighThroughputOptionImplemented is true.
35	HT Operation	The HT Operation element is included by an AP and a mesh STA when dot11HighThroughputOptionImplemented is true.
36	20/40 BSS Coexistence	The 20/40 BSS Coexistence element is optionally present when the dot112040BSSCoexistenceManagementSupport is true.
37	Overlapping BSS Scan Parameters	The Overlapping BSS Scan Parameters element is optionally present if dot11FortyMHzOptionImplemented is true.
38	Extended Capabilities	The Extended Capabilities element is present if any of the fields in this element are nonzero.
39	FMS Descriptor	The FMS Descriptor element is present if dot11FMSActivated is true.
40	QoS Traffic Capability	The QoS Traffic Capability element is optionally present if dot11ACStationCountActivated is true.

Order	Information	Notes
41	Time Advertisement	The Time Advertisement element is present every dot11TimeAdvertisementDTIMInterval if dot11UTCTSFOffsetActivated is true.
42	Interworking	The Interworking element is present if dot11InterworkingServiceActivated is true.
43	Advertisement Protocol	Advertisement Protocol element is present if dot11InterworkingServiceActivated is true and at least one dot11GASAdvertisementID MIB attribute exists.
44	Roaming Consortium	The Roaming Consortium element is present if dot11InterworkingServiceActivated is true and the dot11RoamingConsortiumTable has at least one entry.
45	Emergency Alert Identifier	One or more Emergency Alert Identifier elements are present if dot11EASActivated is true and there are one or more EAS message(s) active in the network.
46	Mesh ID	The Mesh ID element is present if dot11MeshActivated is true.
47	Mesh Configuration	The Mesh Configuration element is present if dot11MeshActivated is true.
48	Mesh Awake Window	The Mesh Awake Window element is optionally present if dot11MeshActivated is true.
49	Beacon Timing	The Beacon Timing element is optionally present if both dot11MeshActivated and dot11MBCAActivated are true.
50	MCCAOP Advertisement Overview	The MCCAOP Advertisement Overview element is optionally present if both dot11MeshActivated and dot11MCCAActivated are true.
51	MCCAOP Advertisement	One or more MCCAOP Advertisement elements are optionally present if both dot11MeshActivated and dot11MCCAActivated are true.
52	Mesh Channel Switch Parameters	The Mesh Channel Switch Parameters element is present when dot11MeshActivated is true and either Channel Switch Announcement element or Extended Channel Switch Announcement element is present.
53	QMF Policy	Indicates the QMF policy parameters of the transmitting STA. The QMF Policy element is present when dot11QMFActivated is true and the transmitting STA is an AP or a mesh STA. This element is not present otherwise.
54	QLoad Report	The QLoad Report element is present every dot11QLoadReportIntervalDTIM DTIMs if dot11QLoadReportActivated is true.
55	HCCA TXOP Update Count	The HCCA TXOP Update Count element is present if both dot11PublicHCCATXOPNegotiationActivated is true and an HC is collocated with the AP.
56	Multi-band	The Multi-band element is optionally present if dot11MultibandImplemented is true.
57	VHT Capabilities	The VHT Capabilities element is present when the dot11VHTOptionImplemented is true.
58	VHT Operation	The VHT Operation element is present when the dot11VHTOptionImplemented is true; otherwise, it is not present.

Order	Information	Notes
59	Transmit Power Envelope element	One Transmit Power Envelope element is present for each distinct value of the Local Maximum Transmit Power Unit Interpretation subfield that is supported for the BSS if both of the following conditions are met: — dot11VHTOptionImplemented or dot11ExtendedSpectrumManagementImplemented is true; — Either dot11SpectrumManagementRequired is true or dot11RadioMeasurementActivated is true. Otherwise, this parameter is not present.
60	Channel Switch Wrapper element	The Channel Switch Wrapper element is optionally present if dot11VHTOptionImplemented or dot11ExtendedSpectrumManagementImplemented is true and at least one of a Channel Switch Announcement element or an Extended Channel Switch Announcement element is also present in the Beacon frame and the Channel Switch Wrapper element contains at least one subelement.
61	Extended BSS Load element	The Extended BSS Load element is optionally present if dot11QosOptionImplemented, dot11QBSSLoadImplemented, and dot11VHTOptionImplemented are true.
62	Quiet Channel	Either one Quiet Channel element containing an AP Quiet Mode field equal to 0 or, in an infrastructure BSS, one or more Quiet Channel elements each containing an AP Quiet Mode field equal to 1 are optionally present if dot11VHTOptionImplemented is true, and either dot11SpectrumManagementRequired or dot11RadioMeasurementActivated is true.
63	Operating Mode Notification	The Operating Mode Notification element is optionally present if dot11OperatingModeNotificationImplemented is true.
64	Reduced Neighbor Report	The Reduced Neighbor Report element is optionally present if dot11TVHTOptionImplemented is true.
65	TVHT Operation	The TVHT Operation element is present for a TVHT STA when the dot11TVHTOptionImplemented is true; otherwise it is not present.
66	Estimated Service Parameters	The Estimated Service Parameters element is present if dot11EstimatedServiceParametersOptionImplemented is true.
67	Future Channel Guidance	The Future Channel Guidance element is optionally present if dot11FutureChannelGuidanceActivated is true.
Last	Vendor Specific	One or more vendor-specific elements are optionally present. These elements follow all other elements.

D. LWA Configuration message contents

```
-- ASN1START

LWA-Configurarion-r13   ::=     CHOICE {
    release             NULL,
    setup               SEQUENCE {
        lwa-Config-r13          LWA-Config-r13
    }
}

LWA-Confgi-r13   ::=     SEQUENCE {
    lwa-MobilityConfig-r13          WLAN-MobilityConfig-r13     OPTIONAL, -- Need ON
    lwa-WT-Counter-r13              INTEGER (0..65535)          OPTIONAL, -- Need ON
}
```

Code Listing D.1: LWA-Configuration message contents.

LWA-Configuration field descriptions:

- **lwa-MobilityConfig**: Indicates the parameters used by WLAN mobility.
- **lwa-WT-Counter**: Indicates the parameter used by UE for WLAN authentication.

Glossary

3GPP 3rd Generation Partnership Program. 22, 23, 26, 28

5GIC 5G Innovation Centre. 10, 46

AC Access Controller. 23, 25

AP Access Point. 21–23, 25, 26, 41, 42, 45–48, 51

AS Access Stratum. 17, 18

ASN.1 Abstract Syntax Notation One. 32, 33, 40

BSS Basic Service Set. 20, 41, 42, 44

CDMA Code Division Multiple Access. 14

CMAS Commercial Mobile Alert System. 15

CN Core Network. 23

CSMA/CA Carrier Sense Multiple Access / Collision Avoidance. 20–22, 26, 41, 44, 45

CTS Clear To Send. 21

DCF Distributed Coordination Function. 20, 21, 81

DIFS DCF Interframe Space. 21, 44, 45

DL downlink. 23, 46–48

DRB Dedicated Radio Bearer. 16, 18, 38

E-UTRAN Evolved UTRAN. 15, 16, 26, 27

EAB Extended Access Barring. 15, 16

EDGE Enhanced Data rates for GSM Evolution. 14, 81

EMM EPS Mobility Management. 18

eNB Evolved Node B. 14, 15, 18, 19, 23, 25–28, 34, 36, 38, 40, 49

EPC Evolved Packet Core. 17, 26, 27

EPS Evolved Packet System. 17, 81

ETWS Earthquake and Tsunami Warning System. 14–16

FCS Frame Check Sequence. 44, 47

FTP File Transfer Protocol. 23

GERAN GSM EDGE Radio Access Network. 14

GSM Universal Mobile Telecommunications System. 14, 81

HSPA High-Speed Packet Access. 13

HSS Home Subscriber Server. 16, 17, 27

HT High Throughput. 46, 48

IEEE Institute of Electrical and Electronics Engineers. 10, 11, 13, 19, 20, 23, 26, 28, 29, 44, 46–48, 51

IMSI International Mobile Subscriber Identity. 16

IP Internet Protocol. 13, 22, 28, 82, 84

IPsec IP security. 10, 82

ISM Industrial, Scientific and Medical. 19, 20

ITU International Telecommunication Union. 13

LTE Long Term Evolution. 10, 11, 13, 15, 16, 22, 23, 26, 28, 47, 49, 51, 52, 82

LTE-A LTE Advanced. 13

LWA LTE-WLAN Aggregation. 10, 13, 22, 23, 25, 26, 28, 36, 37, 40, 51, 52, 82

LWAAP LWA Adaptation Protocol. 23

LWIP LTE WLAN integration with IPsec tunnel. 10, 23

MAC Media Access Control. 20, 23

MBMS Multimedia Broadcast Multicast Service. 13, 15

MBSFN Multicast-Broadcast Single Frequency Network. 15

MCS Modulation and Coding Scheme. 46

MIB Master Information Block. 14, 15

MIMO Multiple In Multiple Out. 9, 13

MME Mobility Management Entity. 16–18, 27

MSC Message Sequence Chart. 29, 73

NAS Non Access Stratum. 14, 16, 17

OAI OpenAirInterface. 10, 11, 16, 26–29, 32, 33, 36, 37, 41, 47, 49–52

OFDMA Orthogonal Frequency Division Multiple Access. 13

OPEX Operating Expenses. 9

OSA OpenAirInterface Software Alliance. 26

OSI Open Systems Interconnection. 32

P-GW PDN Gateway. 17, 27

PBCH Physical Broadcast Channel. 14

PCF Point Coordinated Function. 21

PDCCH Physical Downlink Control Channel. 29

PDCP Packet Data Convergence Protocol. 22, 23

PDN Public Data Network. 17, 83

PDSCH Physical Downlink Shared Channel. 14, 15

PDU Protocol Data Unit. 23

PLMN Public Land Mobile Network. 14

PRB Physical Resource Block. 15

QoS Quality of Service. 17, 19

RAN Radio Access Network. 23

RRC Radio Resource Control. 14, 18, 26, 32, 37, 40

RRM Radio Resource Measurement. 20

RTS Request To Send. 21

S-GW Serving Gateway. 18, 19, 27

SAI Service Area Identifier. 15

SC-PTM Single Cell Point To Multipoint. 15

SDR Software-defined Radio. 27

SFN System Frame Number. 16

SI System Information. 13, 15, 16

SIB System Information Block. 10, 11, 13–16, 26–29, 32, 34–40, 49, 51, 52

SSID Service Set Identifier. 44

STA station. 21, 22, 41, 42, 45–48, 51

TA Tracking Area. 17

TU Time Units. 22, 44, 45, 47, 48, 51

UE User Equipment. 14–18, 22, 23, 25–28, 35–37, 39, 41, 49, 52, 80

UL uplink. 14, 17

UMTS Universal Mobile Telecommunications System. 13, 14, 84

UTC Coordinated Universal Time. 15

UTRAN UMTS Terrestrial Radio Access Network. 14, 15, 81

V2X Vehicle-To-Everything. 15

VoIP Voice over IP. 13

WiMAX Worldwide Interoperability for Microwave Access. 28

WLAN Wireless Local Area Network. 10, 15, 19–23, 25, 26, 28, 42, 49, 51, 52, 80, 84

WT WLAN Termination. 23, 25