UiO : Department of Informatics
University of Oslo

Towards 5G Mobile Networks with OpenAirInterface5G Virtualization

Master's thesis

Bruno Dzogovic

01.05.2018

Oslo, 2018

Towards 5G mobile networks with OpenAirInterface5G virtualization

University of Oslo, Mathematics and Natural Sciences -
Department of Informatics
Programme: Network and System Administration

Bruno Dzogovic, M.Sc.
Head Cloud Network Engineer,
5G Networks Research Assistant
Oslo Metropolitan University
Oslo, Norway
+47 465 61 964
bruno.dzogovic@hioa.no
bruno.dzogovic@gmail.com

Mentor: Prof. Dr. Thanh van Do
Telenor Group, Telenor Research
Oslo Metropolitan University
Oslo, Norway
thanh-van.do@telenor.com

TABLE OF CONTENTS

1. INTRODUCTION

Mobile communications encompass most of modern-day life, including professional, personal and even enterprise applications. In order to enable adequate connectivity and optimized user experience, the telecom operators or communication service providers as they call themselves nowadays, are constantly introducing new and emerging technologies which fit to the different environments. Indeed, different technologies and different configuration are applied in order to achieve satisfactory level of quality of service.

1.1. Motivation

The main objectives of a sound mobile infrastructure are not only to provide appropriate quality of service for good user experience, and to offer high service availability to the user but also to ensure economic affordability. The latter objective is in direct conflict with the two first ones and quite often mobile operators would have to find a good balance between them. This is unfortunately not a trivial task since the number of users and the demand of bit rates are changing dynamically depending on the situation such as the required bit rate at a football station during a soccer game could be hundred times more than at regular daily situation. A static over dimensioning of resources in this case will lead to higher costs and consequently higher subscription for the users while a static configuration based on normal traffic will result to loss of service for a certain number of users in peak traffic situation. Consequently, a much more flexible network solution with dynamic resource allocation is urgently needed (UDDENFELDT, Jan, 2017).

Furthermore, with the advent of the Internet of Things the mission of the mobile infrastructure will be no longer to be confined to serving human-to-human communication but also to serving device-to-device or machine-to-machine communication. This constitutes a considerable challenge due the number of devices and also to their heterogeneous demands in terms of bit rate, latency, packet frequency, mobility, etc. which current 4G mobile technologies are not capable of dealing with. Again, the demand for new technologies supporting heterogeneous traffic demands is getting urgent.

To meet the urging needs mentioned above, activities on 5G specifications have been started and the concept of network slicing has been proposed.

According to 3GPP specification TS 23.501 V1.3.0 (3GPP, 2017)

"A Network Slice is defined as a logical network that provides specific network capabilities and network characteristics".

"Network slices may differ for supported features and network functions optimizations. The operator may deploy multiple Network Slice instances delivering exactly the same features but for different groups of UEs, e.g. as they deliver a different committed service and/or because they may be dedicated to a customer".

"A single UE can simultaneously be served by one or more Network Slice instances via a 5G-AN. A single UE may be served by at most eight Network Slices at a time. The AMF instance serving the UE logically belongs to each of the Network Slice instances serving the UE, i.e. this AMF instance is common to the Network Slice instances serving a UE".

The 5GPPP in the white paper "View on 5G Architecture" (5GPPP, 2016) has a more business oriented of 5G and network slicing as follows:

"In responding to the requirements of these services and application, the 5G system aims to provide a flexible platform to enable new business cases and models to integrate vertical industries, such as, automotive, manufacturing, and entertainment. On this basis, network slicing emerges as a promising future-proof framework to adhere by the technological and business needs of different industries".

“The vision of network slicing will therefore satisfy the demand of vertical sectors that request dedicated telecommunication services by providing “customer-facing” on-demand network slice requirement descriptions to operators”.

At first glance the two mentioned definitions seem to complement each other but a thorough review reveals conflicting requirements. Indeed, mobile operators are aiming at providing different network slices with different network functions optimizations and features which fit the demands of a vertical sectors such as automotive, manufacturing, and entertainment. However, the realization of this objective relies on the assumption that a vertical sector uses only one type of devices and a network slice with specific network functions and features can meet their requirements. This is unfortunately not always the case. For example, in health care, there is a need of all three types of devices as follows:

- eMBB (Enhanced Mobile Broadband) devices that have high requirements for bandwidth, such as high definition (HD) videos, virtual reality (VR), and augmented reality (AR).
- uRLLC (Ultra-reliable and Low-latency Communications) devices that requires high reliability and low latency.
- mMTC (Massive Machine Type) that have high requirements for connection density, such as smart city and smart agriculture.

It is hence uncertain whether three types of network slices are required for Health Care vertical or a unique network slice capable of accommodating all the three types of devices is the best solution. One major objective of this Master thesis work is to contribute to the clarification of the concept of network slice via introducing the concepts of softwareization and virtualization of the network function (vNF) of the mobile network infrastructure. With virtualization, the mobile setups should offer high potential for scalability, immutability, ease of operation and deployment simplification while paving the way towards the next generation 5G networks. In the end, the main principles upon which the future generation networks will be based on, are the software-defined networking (SDN) and virtualization of the network function (vNF) – key values for empowering the utilization of the network slicing concept.

1.1.Problem statement

To experiment, test and verify the network slice concept, it is necessary to have a 5G mobile network which is open for configuration and management such that network slices can be configured and instantiated dynamically. The most straightforward solution is to approach a commercial mobile equipment manufacturer such as Ericsson, Nokia, Huawei, etc. This option is challenging because it is not simple for a university to establish a deal with a commercial player, at the same time as a commercial solution may not be sufficiently open to carry our experiments. Another solution is therefore required.

It is hence decided to attempt building a distinct 5G mobile network. Indeed, if the objective is to test and verify the network slice concept and not the advanced radio access technologies, it is sufficient to build an early and primitive version of 5G mobile network consisting of only virtual Network Functions (vNF) connected together by SDN courses. Although quite exciting, this alternative solution is relatively precarious and may prove entirely unachievable.

The main problem addressed by the Master thesis is to demonstrate that it is possible to build an earlier version of 5G mobile network comprising of simply virtual Network Functions (vNF) connected together by SDN lanes by using an open source 4G/LTE mobile software.

To address the main problem, the following subproblems shall be considered:

Subproblem 1: Uncertainties about the quality and maturity of open source 4G/LTE software:

- There are currently a few open-source 4G/LTE software, namely OpenAirInterface5G and OpenLTE
- However, it is quite uncertain that they are able to function properly and form an operational 4G LTE network.
- It is hence necessary to verify that there exists a reliable and operational open source 4G/LTE software.

Subproblem 2: Difficulties in the virtualization of the open 4G/LTE software:

- Even if the open source 4G/LTE is functioning properly on commercial off-the-shelf (COTS) it may not work at all when being executed in virtual environments.
- It is hereafter necessary to verify that the open source 4G/LTE software can be virtualized properly.

Subproblem 3: Challenges in the cloudification aspect of the open 4G/LTE software:

- Even if the open source 4G/LTE could be virtualized, it is still unsure that its cloudification can function properly due to introduced overheads, propagation delays as well as unpredictable factors.
- It is henceforth necessary to experiment and verify that the open source 4G/LTE software can be cloudified.

1.1.Methodology

To solve the problem stated in the previous section the methodology adopted in this Master thesis work is a qualitative one, aiming to provide solutions to the subproblems consisting of the following research components:

- Verification of the quality and maturity of the open source 4G/LTE software.
- Verification of the virtualization of the open source 4G/LTE software.
- Verification of the cloudification of the open source 4G/LTE software.

For each research component an experimental research method is adopted and it consists of the following steps:

- Defining the objectives of the experiment
- Identifying the research Problem
- Conducting the Experiment
- Analysis and Conclusions

1.2.Organization of the thesis

The Master thesis is organized as follows:

- Chapter 1: Introduction gives an overview of the development in the mobile network technologies and an explanation of the motivation of the thesis work. The problem statement and the used methodology are also described thoroughly.
- Chapter 2: Background summarizes all the background knowledge and information that are necessary to read and understand this thesis.
- Chapter 3: Description of OpenAirInterface5G provides a thorough description of the OpenAirInterface5G, the open source 4G/LTE software used in this Master thesis work.
- Chapter 4: Description of the establishment of the mobile network
- Chapter 5: Security and authentication of the mobile network

- Chapter 6: Evaluation
- Chapter 7: Virtualization and Deployment in Cloud
- Chapter 8: Discussion
- Chapter 9: Conclusion

2. BACKGROUND

In this chapter, the crucial particularities that elucidate the essential traits of the next-generation mobile networks will be presented. In other words, the 4G LTE existing technology is the prime bridge to the evolution towards 5G networks. As it will later be explained in the further chapters, the softwareization and virtualization of the 4G LTE hardware is the main aspect, which will enable the next-generation networks to accommodate larger number of expected devices that indeed includes the IoT, sensor, M2M (Machine-to-Machine) communication devices as well as the existing and emerging mobile technologies. Through concepts of network slicing, containerization, service replication, network function virtualization and cloudification, the next-generation networks will provide extensive functionality and robustness of connectivity over longer period of time. However, in order to comprehend the implementation of the open-source solutions that underlie the development of 5G networks, the rudiments of the prevailing 4G LTE technology need to be explicated thus. Principally, the most essential elements to fathom are the LTE constituents, such as i.e. the access channels whose tedious tweaking is of utmost significance to render the production network operational and stable. The understanding of the LTE architecture is indispensable, and therefore, in the following chapter it is explained in detail, together with access techniques, antenna technologies, routing algorithms and security characteristics of the 4th Generation networks.

Furthermore, the chapter encompasses a description of all necessities required for achieving cloudification, virtualization and automation of the mobile network and its deployment. The fundamental open-source solutions are thus being introduced, that include cutting-edge technologies such as: Docker container technology, Kubernetes orchestration of containers, OpenStack cloud platform, as well as software-defined networking solutions as: Calico and Open vSwitch, which in conjunction with Docker and Linux networking shall provide Network Function Virtualization. The chapter is consequently closed with the description of the hardware used for the experiments, specifically the software-defined radio platform that defines the access stratum of the mobile network.

2.1. 4G LTE (Long-term evolution)

LTE, Long Term Evolution, the successor to UMTS and HSPA is the latest way of deployment of high speed cellular services. In its first forms it was a 3G or also referred as a 3.99G technology, but with supplementary accompaniments the technology satisfied the requirements for a 4G standard. In this form it was referred to as LTE-Advanced. There has been a rapid increase in the use of data carried by cellular services, and this increase will only become larger in what has been termed the "data explosion". To accommodate for this and the augmented demands for bigger data communication speeds and lesser latency, additional expansion of the cellular technology is essential. The UMTS cellular technology advancement has been labelled LTE - Long Term Evolution. The idea is that 4G LTE enables much higher speeds to be achieved along with much lower packet latency (a rising demand for many services nowadays), and that 3GPP LTE enables cellular communications services to move forward to meet the needs for cellular technology in the future. The use of LTE also provided the data capabilities that were required before the full launch of the 4G standard known as LTE-Advanced. To better understand the progression of the mobile technologies, the 3GPP (3rd Generation Partnership Project) introduces different releases. The releases start from Phase 1, which refers to the initial phase GSM deployment in 1987. The latest 3GPP Release 16, also known as “5G phase 2” (3GPP, 2017), is started on 22nd of March 2017 and is still in development in the time of writing of this thesis. Each release is a step further on in the evolution of the mobile technology. Purposefully, the Release 15 and 16 represent phase 1 and 2, consequently, where the initial proposals for the deployment of 5G infrastructure are discussed. The fifth generation of mobile networks, factually, represents an evolved LTE network. As with the preceding technologies, the 5G model exploits the existing traits of the LTE in a new manner, which should gracefully improve the performance and usability of the network. Analogously, there are minor and major changes on different architecture layers, especially the access stratum and the eNB (evolved NodeB), which are represented through adaptive measures and simplify the infrastructure further (3GPP, 2017).

Although there are major step changes between LTE and its 3G predecessors, it is nevertheless looked upon as an evolution of the UMTS / 3GPP 3G standards. Consequently, it uses a different form of radio interface, using OFDMA / SC-FDMA instead of CDMA access techniques, there are many similarities with the earlier forms of 3G architecture and there is scope for much re-use. In deciding what is LTE and how does it differ from other cellular systems, a preview at the specifications for the system can provide the desirable answers. LTE enables further evolution of functionality, increased speeds, and general improved performance, as observed in Table 1 (SAUTER, Martin, 2014).

Table 1. Comparison of LTE features with the earlier standards

	WCDMA (UMTS)	**HSPA/ HSDPA/ HSPUPA**	**HSPA+**	**LTE**
Max downlink speed (bps)	14 M	28 M	100M	14 M
Max uplink speed (bps)	128K	5.7 M	11 M	50 M
Latency round trip time (ms)	150 ms	100 ms	50ms (max)	~10 ms
3GPP releases	Rel 99 / 4	Rel 5 / 6	Rel 7	Rel 8
Year of initial roll out	2003 / 4	2005 / 6 HSDPA 2007 / 8 HSUPA	2008 / 9	2009 / 10
Access type	CDMA	CDMA	CDMA	OFDMA / SC - CDMA

Additionally, LTE is an all IP-based network, supporting both IPv4 and IPv6. Originally there was also no basic delivery for voice application. Although Voice over LTE (VoLTE) was complemented, GSMA is decided to be the standard for this purpose. Also, as a temporary solution, techniques including circuit switched fallback (CSFB) are used. LTE has introduced several new technologies in comparison to the aforementioned cellular systems. They allow LTE to function more cost-effectively relating to the spectrum utilization, and also to provide the much higher data rates that are being demanded (SAUTER, Martin, 2014):

- ***OFDM (Orthogonal Frequency Division Multiplex):*** OFDM technology has been introduced into LTE because it empowers high data rates to be conducted efficiently while still providing a high degree of pliability to reflections and interference. The access schemes differ between the uplink and downlink: OFDMA (Orthogonal Frequency Division Multiple Access is used in the downlink; while SC-FDMA (Single Carrier - Frequency Division Multiple Access) is used in the uplink. SC-FDMA is used in view of the fact that its peak to average power ratio is small and the more constant power enables high RF power amplifier efficiency in the mobile handsets - an important factor for battery power equipment.

- ***MIMO (Multiple Input Multiple Output):*** One of the main problems that previous telecommunications systems have encountered is that of multiple signals arising from the many reflections that are encountered. By using MIMO, these additional signal paths can be used to advantage and are able to be used to increase the throughput.

  When using MIMO, it is necessary to use multiple antennas to enable the different paths to be distinguished. Accordingly, schemes using 2 x 2, 4 x 2, or 4 x 4 antenna matrices can be used. While it is relatively easy to add further antennas to a base station, the same is not true of mobile handsets, where the dimensions of the user equipment limit the number of antennas which should be place at least a half wavelength apart. These properties are discussed in a greater detail in the further chapters.

- ***SAE (System Architecture Evolution):*** With the very high data rate and low latency requirements for 3G LTE, it is necessary to evolve the system architecture to enable the improved performance to be achieved. One change is that a number of the functions previously handled by the core network have been transferred out to the periphery. This provides a much "flatter" form of network architecture. In this way latency times can be reduced, and data can be routed more directly to its destination.

The speed in LTE is increased by the upsurge of narrowband carriers without changing the parameters of the actual narrowband channels. Few bandwidths are dedicated for the LTE standard: from 1.25 MHz up to 20 MHz. In order to accommodate the needs of the subscribers, the UE (User Equipment) vendors should produce devices that support those bandwidths. The usage of the particular bandwidth depends on the band utilized (for example band 3, from 1710-1785 MHz for uplink channel and 1805-1880 MHz for the downlink channel, according to the European standards). For example, with adequate signal conditions in a 20-MHz carrier, data speeds beyond 100 Mbit/s can be achieved. To separate the uplink and downlink channels, LTE uses FDD (Frequency Division Duplexing) in most European countries. Some countries have adopted the TDD (Time Division Duplexing), due to the conditions and therefore, the air interfaces of both versions differ significantly. Accordingly, the usage of some LTE devices can be restricted between these areas due to these differences. To address this drawback, the vendors are issuing devices with an air interface that can support the both operational modes, with exclusion of some UE that support either FDD or TDD-capable transmissions. However, the devices must be capable of backwards-compatibility, which means they have to be capable for supporting GSM, GPRS, EDGE and UMTS as well. In the core network of LTE, the interfaces and protocols are established to support sessions and routing of various traffic type and amalgamated movement between the technologies, when the user is roaming between areas served by different air interfaces. Since the LTE is completely IP-based, that trait can be regarded as a major change with regard the previous standards. The 3G UMTS network core is based on traditional circuit-switched packet core for voice, SMS and other services, inherited from GSM. Unlike that, the core network of LTE is completely IP-based, which significantly simplifies the design and reduces the costs for implementation. Analogously, that represents an easier way for management, maintenance and organization of the network infrastructure (SAUTER, Martin, 2014).

The Long-Term Evolution defines particular bands of operation on different continents, which is decided by the World Radio Conference (WRC). Table 2 represents the European bands on which LTE operates.

Table 2. European LTE frequencies (ETSI, 2017)

Band	**Duplex mode**	**f (MHz)**	**Uplink (MHz)**	**Downlink (MHZ)**	**Duplex spacing (MHz)**	**Channel bandwidths (MHz)**
1	FDD	2100	1920 – 1980	2110 – 2170	190	5, 10, 15, 20
3	FDD	1800	1710 – 1785	1805 – 1880	95	1.4, 3, 5, 10, 15, 20
7	FDD	2600	2500 – 2570	2620 – 2690	120	5, 10, 15, 20
8	FDD	900	880-915	2110-2170	400	5, 10, 15, 20
20	FDD	800	832 – 862	791 – 821	−41	5, 10, 15, 20
28	FDD	700	703-748	758-803	55	3, 5, 10, 15, 20
32	FDD	1500	N/A	1452-1496	N/A	5, 10, 15, 20
38	TDD	2600	2570 – 2620		N/A	5, 10, 15, 20

Another key concept and issue in LTE is the latency, which ranges from 50 – 100ms delay for the control-plane (the network core), and approximately 5ms delay for the user-plane. However, in practice even though LTE has low air interface delays, measurements reveal that core network delays compromise the overall round-trip time design requirement. LTE's break-before-make handover implementation causes a data interruption at each handover of 40ms at the median level (LAURIDSEN, Mads et al., 2017, pp.156 - 162). The overall delay in 4G LTE networks is the main entity that needs to be addressed in order to establish the evolution towards 5G. For that purpose, the 3GPP has introduced improvements at the physical and MAC layer in Release 14 and 15 (C. S. ARENAS, John et al., 2017).

2.1.1. Architecture and components of LTE

The LTE network architecture resembles the 3G UTRAN network, thereby the term E-UTRAN (Evolved Universal Terrestrial Radio Access Network). As portrayed in the Figure 1, the components of the E-UTRAN network are connected to the evolved packet core (EPC). The constituents of the EPC are routing the traffic from the physical E-UTRAN plane to the Internet, where each of them has a special dedicated role. Principally, the architecture of the 4G LTE Evolved Packet Core is very similar to the 3G UMTS and 2G GSM, with the difference that it is simplified and separated into radio network part and core network part (COX, C., 2014). The LTE network is divided in two layers of abstraction: *Access stratum (AS)* (3GPP, 2017) and *Non-Access stratum (NAS)* (3GPP, 2015). As the names indicate, the Access stratum enables the UEs to establish a successful connection through the radio equipment, which is also called radio access network. On the other side, the Non-Access stratum is the abstraction layer that defines the communication between the UE and the core network in a transparent manner. Examples of NAS messages are Update or Attach messages, Authentication messages, Service requests etc. (ALI-YAHIYA, Tara, 2011).

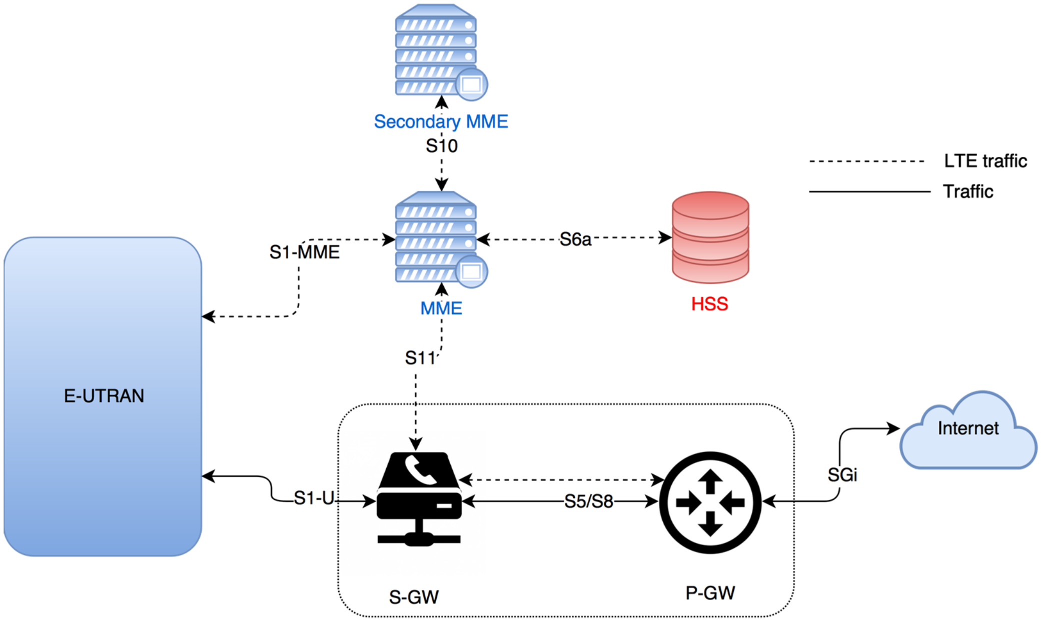


Figure 1. Components of the Evolved Packet Core

The first component is the *Home Subscriber Server (HSS)*, which is in fact a MySQL database containing the users. Accessing the database is regulated with the DIAMETER protocol (FAJARDO, V. et al., 2012), which provides Authentication, Authorization and Accounting (AAA). The S6a is the DIAMETER IP interface through which the MME communicates with the HSS database. The HSS has all the user parameters required for successful authentication of the UE (User Equipment), i.e. mobile phone, with the EPC. The most important parameters are (COX, C., 2014):

- The International Mobile Subscriber Identity (IMSI), which is a unique identifier of a subscriber. The IMSI has the Mobile Country Code (MCC) and Mobile Network Code (MNC), which identifies the user when roaming abroad in order to locate the home network and contact the HSS. The IMSI code is embedded into the SIM card.
- Authentication information for generating encryption keys each session
- Circuit-switched service features as the Mobile Subscriber Integrated Service Digital Network (MSISDN or known as a telephone number). This service allows the subscriber to utilize the GSM and UMTS networks for voice calls, instead of using IP-based LTE.
- Packet-switched service features as the Access Point Names (APNs). This refers to the PDN (Packet Data Network) that the subscribers are going to use in order to access the IP network through the Packet Gateway (P-GW).
- IMS-specific information
- The ID of the particular MSC (Mobile Switching Center, that is a protocol of GSM and UMTS) for correct routing of circuit-switched calls and SMS messages
- The ID of the SGSN (Serving GPRS Support Node) or MME (Mobility Management Entity). This is used in case there are changes in the user's profile, so the updates can be pushed to the other network elements.

The next element of the EPC is the *Packet Data Network Gateway (P-GW).* This gateway enables the EPC to communicate to the outside world through SGi interface. The SGi interface is utilized by the P-GW for communication with external devices or other packet data networks, operator's servers, the Internet or some IP multimedia subsystem. As previously stated, the packet data network is identified by APN (Access Point Name), found in the HSS database. An operator can define few APN names for different purposes, for example: one access point name for accessing the Internet and another one for accessing IP multimedia subsystem. Those APNs are saved as entries in the mobile device, which should automatically connect to the default packet data network, such as the Internet (COX, C., 2014).

The *Serving Gateway (S-GW)* is another type of router that forwards data between the eNB base station and the P-GW. One network usually contains multiple S-GWs, which have the role of tracking the mobile devices in certain geographical region. Every device that is attached to a base station is assigned to a certain S-GW but can also change the router if it roams to another geographical region with different eNB and dedicated S-GW (COX, C., 2014). In the radio network plane, the S-GW terminates the S1-UP GTP (GPRS Tunneling Protocol) tunnels, and on the core network plane, it terminates the S5-UP GTP tunnels to the gateway to the outside world. The S1 and S5 tunnels are independent and are interchanged by requirement. For example, when there is a handover to an eNB under the control of the same MME and S-GW, only the S1 tunnel needs to be modified to redirect the user's stream to and from the new base station. On the other hand, if the connection is handed over to an eNB that is under the control of another MME and S-GW, the S5 tunnel has to be modified as well (SAUTER, Martin, 2014). The tunnel generation and modification are controlled by the MME, which informs the S-GW via the S11 interface (Figure 1). In fact, the S11 interface utilizes the same GTP-C control protocol from GSM and UMTS by presenting new messages. UDP protocol is utilized as a transport protocol instead of SCTP, and the IP protocol is used in the network layer (SAUTER, Martin, 2014).

The *Mobility Management Entity (MME)* is the most complex component of the EPC. It controls the high-level operation of the mobile devices. Namely, the MME handles the users and the eNBs at the core network. Bigger networks utilize multiple MMEs to handle the bigger load and to enable redundancy and fault-tolerance. Since the MME is not responsible for the air interface operations, the signaling it exchanges with the radio network is referred to as Non-Access Stratum (NAS) signaling. The following tasks are the main obligations of MME (SAUTER, Martin, 2014):

- The MME handles the user authentication with the core network. Since it communicates directly with the HSS via the S6a interface, the user authentication requests are forwarded from the eNB through the S1 interface to MME and then the MME proceeds with the DIAMETER protocol. If successful, the MME forwards encryption keys to the eNB so that further signaling exchange encryption proceeds over the radio network.
- Another task that MME has is the establishment of bearers. Since it is not directly implicated in the exchange of user data packets between mobile devices and the Internet, the MME establishes IP tunnels between the eNB and other EPC components as the P-GW. This includes selection of a gateway router to the Internet if there is more than one gateway available.
- Non-Access Stratum mobility management. A mobile device that can find itself idle for some time (usually 10-30 seconds) is released from the radio network. The device can roam between different eNBs in a same Tracking Area (TA), without notifying the network in order to save battery capacity and signaling overhead. In case when new data packets from the Internet arrive at the device while in this state, the MME sends paging messages to all base stations that are part of the current Tracking Area of the mobile device. Once the device responds to the paging, the bearers are reinitialized.
- If there is no support for X2 interface, the MME aids the forwarding of handover messages between the two involved base stations. The MME is also responsible for establishing and modification of the user data IP tunnel after a handover, in case different core network routers are selected.
- The MME dictates interworking with other radio networks. This refers to devices that reach the limit of the LTE coverage area and roam into areas that are covered by GSM or UMTS. In this case, the eNB decides to hand over the device to the GSM or UMTS networks or instructs it to perform a cell change to suitable cell. During this process, the MME communicates with the GSM or UMTS network to manage the transfer of the device successfully.
- SMS and voice support are managed by MME in LTE. Since LTE is IP-based network, still the SMS and voice services are in high demand. The MME maps these services to the UMTS and GSM circuit-switched core networks. To perform this, the MME initializes a number of different interfaces (S5, S6a, S11 and SGs).

When compared to GPRS and UMTS, the tasks of MMEs are the same as those of the SGSN. The major difference between the two entities is that while the SGSN is also responsible for forwarding the user data between the core network and the radio network, the MME deals only with the signaling tasks described above and leaves the user data to the Serving Gateway (S-GW), which is described in the next section (SAUTER, Martin, 2014).

A. Protocol architecture in LTE

Generally, in an LTE network, the protocols can be divided into two groups: *Control-plane protocols* and *User-plane protocols*. The control-plane protocols are handling Access Stratum (AS) radio-specific functionalities, whereas the user-plane protocols define three main tasks: handling IP packets, radio link control and MAC-layer particularities (ALI-YAHIYA, Tara, 2011).

Control plane protocols

As represented in Figure 2, the greyed part of the stack represents the Access Stratum protocols. The AS interacts with the Non-Access Stratum (NAS), also referred to as “upper layers”. Among other functions, the NAS control protocol handle Public Land Mobile Network (PLMN) selection, tracking area update, paging, authentication and Evolved Packet System (EPS) bearer establishment, modification and release. The applicable AS-related procedures largely depend on the Radio Resource Control (RRC) state of the User Equipment (UE), which can be either RRC_IDLE or RRC_CONNECTED. A UE in RRC_IDLE performs cell selection and reselection – in other words, it decides on which cell to camp. The cell (re)selection process takes into account the priority of each applicable frequency of each applicable Radio Access Technology (RAT), the radio link quality and the cell status (i.e. whether a cell is barred or reserved). An RRC_IDLE UE monitors a paging channel to detect incoming calls, and also acquires system information. The System Information (SI) mainly consists of parameters by which the network (E-UTRAN) can control the cell (re)selection process. In RRC_CONNECTED, the E-UTRAN allocates radio resources to the UE in order to facilitate the transfer of unicast data via shared data channels. To support this operation, the UE monitors an associated control channel used to indicate the dynamic allocation of the shared transmission resource in time and frequency. The UE provides the network with reports of its buffer status and of the downlink channel quality, as well as neighborhood cell measurement information to enable E-UTRAN to select the most appropriate cell for the UE. These measurement reports include cells using other frequencies or RATs. The UE also receives SI, consisting mainly of information required to use the transmission channels. To extend its battery lifetime, a UE in RRC_CONNECTED may be configured with a Discontinuous Reception (DRX) cycle. RRC, as specified in the figure, is the protocol by which the E-UTRAN controls the UE behavior in RRC_CONNECTED. RRC also includes the control signaling applicable for a UE in RRC_IDLE, namely paging and SI, which altogether defines the connection control in LTE (VELDE, Himke van der, 2011, pp.57-86).

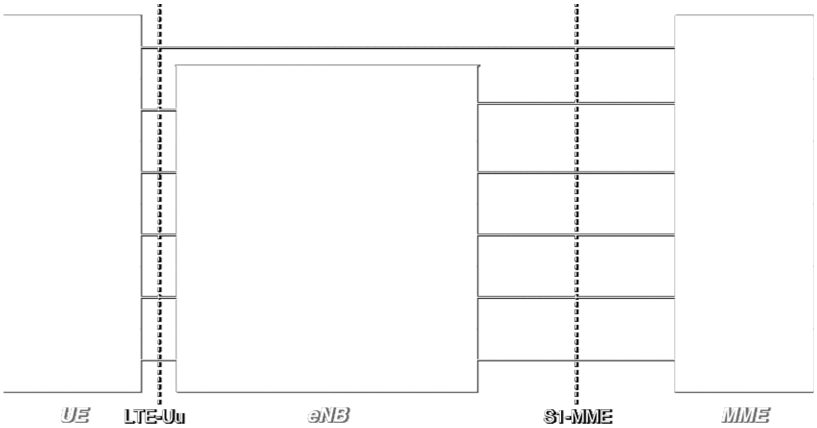


Figure 2. LTE Control plane protocol stack

The more important entities for establishment and connection detachment in LTE are the constituents that carry system information, namely the carried System Information Blocks (SIBs) (VELDE, Himke van der, 2011, pp.57-86). They constitute functionality-related parameters required for a successful communication between the UE and the NAS:

- **Master Information Block (MIB):** Includes limited number of the most frequently transmitted parameters, which are essential for a UE’s initial access to the network
- **System Information Block Type 1 (SIB1):** Contains parameters needed to determine if a cell is suitable for cell selection, as well as information about the time-domain scheduling of other SIBs.
- **System Information Block Type 2 (SIB2):** Includes common and shared channel information.
- **SIB3-SIB8:** Include parameters used to control intra-frequency, inter-frequency and inter-RAT cell reselection.
- **SIB9:** Used to signal the name of a Home eNodeB (HeNB).

- **SIB10-SIB12:** Include the Earthquake and Tsunami Warning Service (ETWS) notifications and Commercial Mobile Alert System (CMAS) warning messages
- **SIB13:** Includes MBMS (Multimedia Broadcast Multicast Service) related control information.

The way connections are established in LTE are described in Figure 3, where the RRC connection involves establishment of SRB1 and the transfer of the initial uplink NAS message (the SRB0-2 are the signaling radio bearers, which are used for the transfer of RRC and NAS signaling messages and elucidated in the succeeding subsection. The same signaling radio bearers carry also information about the previously-explained channels assignment). The NAS message triggers the establishment of the S1 connection, which initiates a subsequent step during which E-UTRAN activates Access Stratum security and starts the following SRB2 (VELDE, Himke van der, 2011, pp.57-86).

Figure 3. RACH procedure connection establishment in LTE

User plane protocols

The user-plane Layer-2 architecture incorporates three sublayers as shown in the encapsulation in Figure 4:

- **Packet Data Convergence Protocol (PDCP):** This protocol resides on the physical Layer 1 and processes RRC messages in the control plane and IP packets in the user plane. In accordance to the radio bearer, the main functions of the PDCP layer are header compression, security (integrity protection and ciphering), also support for encoding and retransmission during handovers. For radio bearers which are configured to use PDCP layer, there is one PDCP entity per radio bearer. The PDCP layer manages data streams in the user plane as well as in the control plane, only for the radio bearers using either a Dedicated Control Channel (DCCH) or a Dedicated Transport Channel (DTCH). The architecture of the PDCP layer differs for user plane data and control plane data. Two different types of PDCP PDU are defined in LTE: PDCP Data PDUs and PDCP Control PDUs. The PDCP Data PDUs are used for both control and user plane data, whereas PDCP Control PDUs are only used to transport the feedback information for header compression and for PDCP status reports, which are used in case of handover. (FISCHER, P. et al., 2011, pp.87-120).

- **Radio Link Control (RLC):** This protocol is situated on Layer-2 and the main functions it performs are segmentation and reassembly of upper layer packets in order to adapt them to the size which can actually be transmitted over the radio interface. For radio bearers which need error-free transmission, the RLC layer also performs retransmission to recover out-of-order reception due to Hybrid Automatic Repeat Request (HARQ) operation in the lower layer. One RLC entity exists per radio bearer. The RLC layer is located between the PDCP layer and the MAC layer (Figure 4). It communicates with the PDCP layer through a Service Access Point (SAP), and with the MAC layer via logical channels. The RLC layer reformats PDCP PDUs in order to fit them into the size indicated by the MAC layer; that is, the RLC transmitter segments and/or concatenates the PDCP PDUs, and the RLC receiver then reassembles the RLC PDUs to reconstruct the PDCP PDUs. Additionally, the RLC reorders the RLC PDUs if they are received out of sequence due to the HARQ operation performed in the MAC layer. This is the key difference from UMTS, where the HARQ reordering is performed in the MAC layer. The advantage of HARQ reordering in RLC is that no additional SN and reception buffer are required for the HARQ reordering and RLC-level ARQ related operations. The functions of the RLC layer are performed by RLC entities. An RLC entity is configured in one of three data transmission modes: Transparent Mode (TM), Unacknowledged Mode (UM) and Acknowledged Mode (AM). In AM, special functions are defined to support retransmission. When UM or AM is used, the choice between the two modes is made by the eNB during the RRC radio bearer setup procedure, based on the QoS requirements of the EPS bearer (FISCHER, P. et al., 2011, pp.87-120).
- **Media Access Control (MAC**): The Layer-2/3 MAC is analogous to the TCP/IP MAC layer, which in LTE actually performs multiplexing of data from different radio bearers. Therefore, there is only one MAC entity per UE. By deciding the amount of data that can be transmitted from each radio bearer and instructing the RLC layer as to the size of packets to provide, the MAC layer aims to achieve the negotiated Quality of Service (QoS) for each radio bearer. For the uplink (UL), this process includes reporting to the eNB the amount of buffered data for transmission. Specifically, the MAC layer consists of a HARQ entity, a multiplexing/demultiplexing entity, a logical channel prioritization entity, a random access control entity and a controller which performs various control functions. The MAC layer conducts multiplexing and demultiplexing between logical and transport channels as well as transport channels by constructing MAC PDUs, known as Transport Blocks (TBs), from MAC SDUs received through the aforementioned logical channels. Afterwards, the MAC layer in the receiving side recovers MAC SDUs from MAC PDUs received through transport channels. To elucidate the HARQ entity, its responsibility for the transmission and receiving of HARQ operations is explicated; indicating, that the transmit HARQ operation includes (re)transmission of TBs and reception and processing of ACK/NACK signaling. The receive HARQ operation includes reception of TBs, combining of the received data and generation of ACK/NACK signaling. In order to enable continuous transmission while previous TBs are being decoded, up to eight HARQ processes in parallel are used to support multiprocess ‘Stop-And-Wait’ (SAW) HARQ operation. SAW operation means that upon transmission of a TB, a transmitter stops further transmission and waits for feedback from the receiver. When a NACK is received, or when a certain time elapses without receiving any feedback, the transmitter retransmits the TB. Such a simple SAW HARQ operation cannot on its own utilize the transmission resources during the period between the first transmission and the retransmission. Therefore, multiprocess

HARQ interlaces several independent SAW processes in time so that all the transmission resources can be used. Each HARQ process is responsible for a separate SAW operation and manages a separate buffer (FISCHER, P. et al., 2011, pp.87-120). The MAC layer is comprised of various logical, transport and physical channels which are detailed in the incoming section.

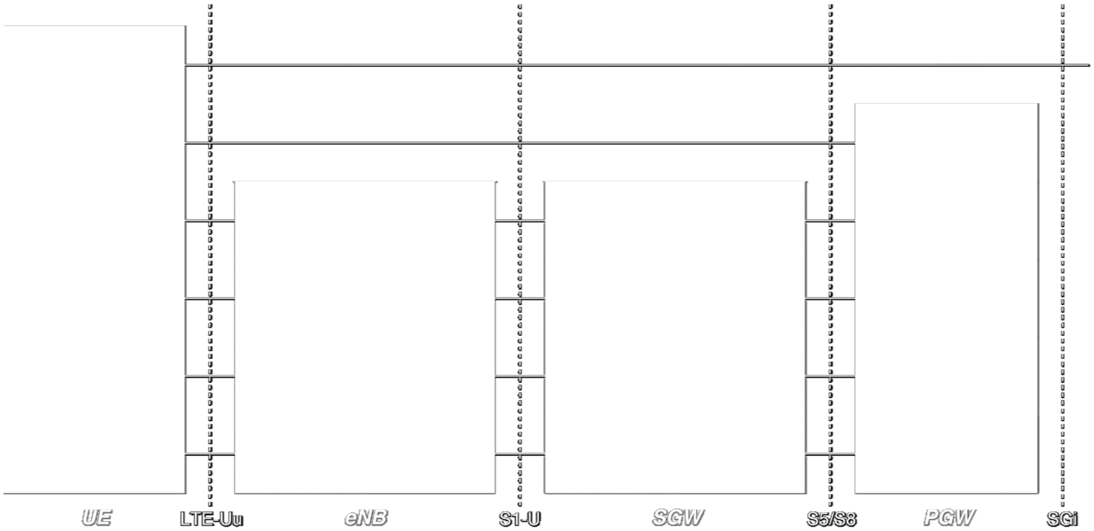


Figure 4. LTE User plane protocol stack

B. Interfaces

LTE defines various interfaces for communication between the different constituents, as shown in Figure 1. Namely, there are several significant interfaces that are residing in the EPC among which the most important is the S1 which defines the communication between the eNB and the EPC through the MME, as well as communication between base-stations:

- **S1 interface** – The S1 interface is split into two interfaces, one for the control plane and the other for the user plane. At the *control plane* (see Figure 2)*,* the S1 is based on the SCTP (Stream Control Transmission Protocol) protocol (PALAT, S. and Godin, P., 2011, pp.25-55).
  SCTP is constructed to carry Public Switched Telephone Network (PSTN) signaling messages over IP networks, but is also efficient in variety of other applications. SCTP is a reliable transport protocol operating on top of a connectionless packet network such as IP. It offers the following services to its users: acknowledged error-free non-duplicated transfer of user data, data fragmentation to conform to discovered path MTU size, sequenced delivery of user messages within multiple streams, with an option for order-of-arrival delivery of individual user messages, optional bundling of multiple user messages into a single SCTP packet, and network-level fault tolerance through supporting of multi-homing at either or both ends of an association. SCTP delivers some of the equivalent properties of UDP and TCP: it is message-oriented like UDP and guarantees a reliable, in-sequence transport of messages with congestion control like TCP. SCTP differs by that it provides multi-homing and redundant paths to increase resilience and reliability. SCTP applications acquiesce their data to be transferred in messages (groups of bytes) to the SCTP transport layer. SCTP groups messages and control information into distinct portions (data chunks and control chunks), each identified by a chunk header. The protocol can fragment a message into a number of data chunks, but each data chunk contains data from only one user message. SCTP bundles the chunks into SCTP packets. The SCTP packet, which is submitted to the Internet Protocol, consists of a packet header, SCTP control chunks (when necessary), followed by SCTP data chunks (when available). One can characterize SCTP as

message-oriented, meaning it transports a sequence of messages (each being a group of bytes), rather than transporting an unbroken stream of bytes as does TCP. As in UDP, in SCTP a sender sends a message in one procedure, and that particular message is conceded to the receiving application process in a single action. Contrary to that, TCP is a stream-oriented protocol, transferring streams of bytes steadfastly and in organized manner. However, TCP does not inform the receiver about the number of times the sender application called on the TCP transport passing it groups of bytes to be sent out. At the sender, TCP simply affixes more bytes to a queue of bytes anticipating to be sent over the network, rather than maintaining a queue of individual distinct outbound messages which must be conserved per se. SCTP is referred to as 'multi-streaming' due to the aptitude for transmission of several independent streams of chunks in parallel; for example, transmitting web page images together with the web page text. Practically, SCTP encompasses pairing several connections into a single SCTP association, operating on messages (or chunks) rather than bytes. TCP preserves byte order in the stream by including a byte sequence number with each segment. SCTP, on the other hand, assigns a sequence number or a message-id to each message sent in a stream. This allows independent ordering of messages in different streams. However, message ordering is optional in SCTP; a receiving application may choose to process messages in the order of receipt instead of in the order of sending (IETF, 2007) [SCTP – Stream Control Transmission Protocol standard].

A further simplification in LTE (compared to the UMTS Iu interface, for example) is the direct mapping of the S1-AP (S1 Application Protocol) on top of SCTP which results in a simplified protocol stack with no intermediate connection management protocol. The individual connections are directly handled at the application layer. Multiplexing takes place between S1-AP and SCTP whereby each stream of an SCTP association is multiplexed with the signaling traffic of multiple individual connections. Another point of flexibility that comes with LTE lies in the lower layer protocols for which fully optionality has been left regarding the choice of the IP version and the choice of the data link layer (PALAT, S. and Godin, P., 2011, pp.25-55). On the *user plane*, the S1 interface is based on the GTP-U (GPRS Tunneling Protocol-User plane) and UDP, inherited from the UMTS networks. One of the advantages of using GTP-U is its inherent facility to identify tunnels and also facilitate intra-3GPP mobility. The IP version number and the data link layer have been left fully operational, as for the control plane stack. A transport bearer is identified by the GTP tunnel endpoints and the IP address (source Tunneling End ID (TEID), destination TEID, source IP address, destination IP address). The S-GW (Service Gateway) sends downlink packets of a given bearer to the eNB IP address (received in S1-AP) associated to that particular bearer. Similarly, the eNB sends upstream packets of a given bearer to the EPC IP address (received in S1-AP) associated to that particular bearer. The initialization of S1-MME control plane interface starts with the identification of the MMEs to which the eNB must connect, followed by the setting up of the Transport Network Layer (TNL). Only one SCTP association is established between one eNB and one MME, but with multiple pairs of streams for avoiding head-of-line blocking. When a UE is associated to a specific MME, a context is created and saved for the particular UE in the MME. This particular MME is selected by the NAS Node Selection Function (NSSF) in the first eNB from which the UE entered the pool. When the UE becomes active under the coverage of a particular eNB in the pool area, the MME provides the UE context information to this eNB using the 'INITIAL_CONTEXT_SETUP_REQUEST' message, which allows the eNB in turn to create a context and manage the UE while it is in active mode. Besides these functionalities, the S1 interface also enables load-balancing of the traffic that reaches the MME from the eNB and the UEs attached to it. Bearer management is initiated via S1 with the BEARER_SETUP_REQUEST and BEARER_SETUP_RESPONSE messages. When a handover process starts, the S1 interface communicates with the X2 interface in order to acquire information about the UE that is subject to the handover from the current to the next eNB (PALAT, S. and Godin, P., 2011, pp.25-55).

- **S3 interface** - S3 is a GTP signaling-only interface, used between the Serving GPRS Support Node (SGSN) and the Mobility Management Entities (MME) to support inter-system mobility. In other words, the S3 interface serves as a control interface between the MME and 2G/3G SGSNs (3GPP, 2010) [Specification TS 29.303 v9.1.0: Stage 3, Release 9].
- **S5/S8 interfaces** – The communication between the Service Gateway (SGW) and the Packet Gateway (PGW) is defined via the S5/S8 interfaces. Technically, the S5 is identical as S8 interface with the difference that S8 is used when roaming between different operators while S5 is network internal. The S5 / S8 interface will exist in two flavors one based on Gn/GTP (SGSN-GGSN) and the other will use the IETF specified Proxy Mobile IP (PMIP) for mobility control with additional mechanism to handle QoS. The motivation for the PMIP flavor of S5/S8 has mainly come from WiMAX/CDMA2000 operators and vendors interested in inter-working with E-UTRAN, GERAN or UTRAN, or re-using the 3GPP EPS specified mechanism also for intra WiMAX / CDMA2000 mobility. It has been agreed in 3GPP that the usage of PMIP or GTP on S5 and S8 should not impact RAN behavior or impact the terminals. The usage of PMIP or GTP on S5/S8 will not be visible over the S1 interface or in the terminal. In the non-roaming case, the S-GW and P-GW functions can be performed in one physical node. The S5/S8 is a many-to-many interface (3GPP, 2010) [Specification TS 23.402 v 9.5.0 Release 9].
- **S6a interface** – Handles the DIAMETER authentication procedure (IETF, 2003) [RFC3588 standard] from the MME towards the HSS database of UEs that request attachment procedures on the specific eNB (3GPP, 2015) [3GPP specification 29.272].
- **S10 interface -** This is a control interface between the MMEs which will be very similar to the S3 interface between the SGSN and MME. The interface is based on Gn/GTP-C (SGSNSGSN) with additional functionality and is a many-to-many interface (3GPP, 2010) [Specification TS 29.303 v9.1.0: Stage 3, Release 9].
- **S11 interface** – Establishes communication between the MME and the S-GW. The interface is based on Gn/GTP-Control (GTP-C) with some additional functions for paging coordination, mobility compared to the legacy Gn/GTP-C (SGSN-GGSN) interface (3GPP, 2015) [Specification 29.274: EPS; eGPRS, GTPv2-C, stage 3].
- **X2 interface** – Interconnects two eNBs. The protocol stack at which X2 resides is the same as the S1 interface; specifically, it uses the SCTP protocol to establish a communication between two or eNodeBs. This way, the succeeding eNB receives signaling information from the preceding eNB for a UE that is roaming and is subject to a handover from the latter to the former. The exchange of load information between eNBs is of key importance in the flat architecture used in LTE, as there is no central Radio Resource Management (RRM) node as in the case of UMTS with the Radio Network Controller (RNC). The exchange information can be of a load-balancing character or interference coordination (PALAT, S. and Godin, P., 2011, pp.25-55).
- **SGi/Gi interfaces** – The SGi interface connects the PGW to an external network (PDN), and the Gi interface connects the GGSN to an external network (PDN). The interface is based on the IP packet (user data/payload/data plane). It also enables exchange of signaling and routing redistribution (OSPF, BGP, RIP routing etc.). The interface can connect to the RADIUS/DIAMETER servers if the service is used. The implementation of SGi and Gi interfaces in real topology network combine all IP various packets in one routing table (virtual router or routing instance). It is possible to enable dynamic routing protocols such as OSPF, RIP, ISIS, BGP or EIGRP etc., for advertising PGW/GGSN IP address to external networks. Usually the PGW/GGSN can enable minimum basic OSPF routing and also enable redundancy

mechanism such as VRRP (Virtual Router Redundancy Protocol) (CISCO, 2017) for multiple nodes (3GPP, 2010) [Specification 29.061: Release 9; v9.3.0].

C. Quality of Service (QoS) and EPS bearers

In a real scenario, the UE runs multiple applications simultaneously, which may require different Quality of Service parameters. For example, one can use the UE for engaging in a VoIP call while at the same time browsing the Internet or downloading a file. The VoIP call requires lower latency and jitter, whereas the file transfer needs much lower packet loss rate. To support the different requirements for QoS factors, bearers are being established that can be associated with a particular QoS feature. Bearers can be classified into two categories: *Minimum Guaranteed Bit Rate (GBR) bearers* and *Non-GBR bearers.* The former ones are used for applications such as VoIP and have associated a GBR value for which dedicated transmission resources are permanently allocated at bearer establishment. If there are resources available, then higher bit rates than the defined GBR may be allowed for the particular bearer. The *Non-GBR* bearers do not guarantee any particular bit rate. Accordingly, they can be used for FTP applications, web browsing and similar appliances. For these bearers, no bandwidth resources are allocated permanently to the bearer (PALAT, S. and Godin, P., 2011, pp.25-55).

In the Access Stratum (AS), the eNB sets the bearers up and ensures that the adequate QoS parameters are assigned to each. A bearer has a Class Identifier (QCI) and an Allocation and Retention Priority (ARP) associated. The QCI is characterized by priority, packet delay budget and acceptable packet loss rate. The QCI label for a bearer determines the way it is handled in the eNB. The CQIs are standardized and thus the vendors can all have the same understanding of the underlying service characteristics and thus provide the corresponding treatment, including queue management, conditioning and policy strategy. This ensures that the LTE operator can expect uniform traffic handling behavior throughout the network regardless of the manufacturers of the eNB equipment (see Table 3) (PALAT, S. and Godin, P., 2011, pp.25-55).

Table 3. Standardized QoS Class Identifiers (QCI) for LTE (PALAT, S. and Godin, P., 2011, pp.25-55)

QCI	Resource type	Priority	Packet delay budget (ms)	Packet error loss rate	Example services
1	GBR	2	100	10^{-2}	Conversational voice
2	GBR	4	150	10^{-3}	Conversational video (live streaming)
3	GBR	5	300	10^{-6}	Non-conversational video (buffered streaming)
4	GBR	3	50	10^{-3}	Real-time gaming
5	Non-GBR	1	100	10^{-6}	IMS signaling
6	Non-GBR	7	100	10^{-3}	Voice, video (live streaming), interactive gaming
7	Non-GBR	6	300	10^{-6}	Video (buffered streaming)
8	Non-GBR	8	300	10^{-6}	TCP based (e.g. WWW, e-mail) chat, FTP, p2p file sharing, progressive video call etc.
9	Non-GBR	9	300	10^{-6}	

The priority and packet delay budget from the QCI label determine the RLC mode configuration, and how the scheduler in the MAC handles packets sent over the bearer (e.g. in terms of scheduling policy, queue management policy and rate shaping policy). The ARP of a bearer is used for call admission control (for example, to decide whether or not the requested bearer should be established in case of radio congestion. It also governs the prioritization of the bearer for pre-emption with respect to a new bearer establishment request. Once successfully established, a bearer's ARP does not have any impact on the bearer-level packet forwarding treatment. Such packet forwarding treatment should be explicitly determined by the other bearer-level QoS parameters such as QCI, GBR and MBR (PALAT, S. and Godin, P., 2011, pp.25-55).

As shown in Figure 5, an EPS bearer has to cross multiple interfaces (the S5/S8 interface from the P-GW to the S-GW, the S1 interface from the S-GW to the eNB and the radio interface (LTE-Uu) from the eNB to the UE. Across each interface, the EPS bearer is mapped onto a lower layer bearer, each with its own bearer identity. Each node must keep track of the binding between the bearer IDs across its different interfaces. An S5/S8 bearer transports the packets of an EPS bearer between a P-GW and an S-GW. The S-GW stores a one-to-one mapping between a S1 bearer and a S5/S8 bearer. The bearer is identified by the GTP tunnel ID across both interfaces. A S1 bearer transports the packets of an EPS bearer between the S-GW and the eNB. A radio bearer transports the packets of an EPS bearer between a UE and an eNB. An E-UTRAN Radio Access Bearer (E-RAB) refers to the concatenation of an S1 bearer and the corresponding radio bearer. An eNB stores one-to-one mapping between a radio bearer ID and a S1 bearer to create the mapping between the two (PALAT, S. and Godin, P., 2011, pp.25-55).

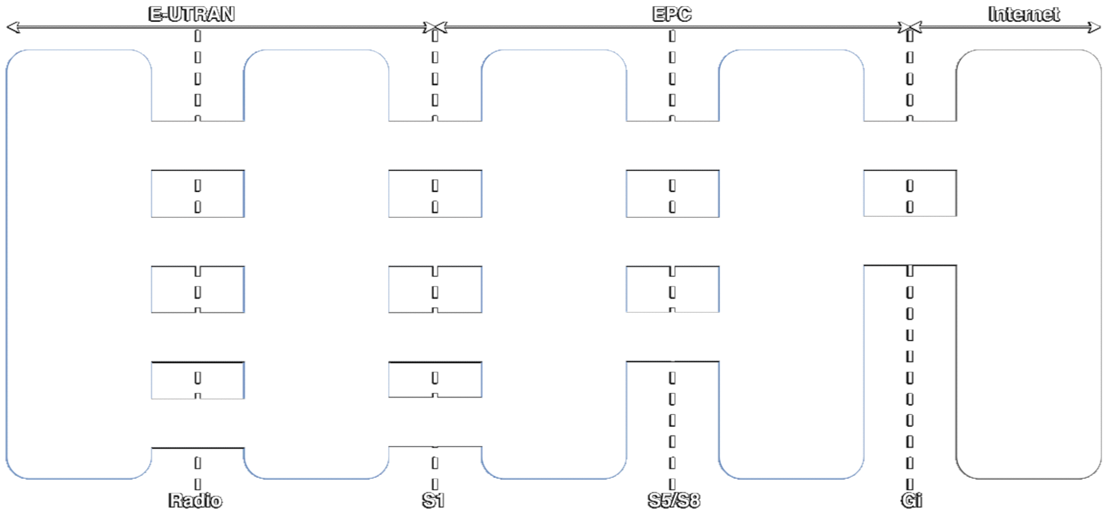


Figure 5. Overall EPS bearer service architecture

IP packets mapped to the same EPS bearer receive the same bearer-level packet forwarding treatment (scheduling policy, queue management policy, rate shaping policy, RLC configuration). Providing different bearer-level QoS thus requires that a separate EPS bearer is established for each QoS flow, and use IP packets must be filtered into the different EPS bearers. Packet filtering into different bearers is based on Traffic Flow Templates (TFTs). The TFTs use IP header information such as source and destination IP addresses and Transmission Control Protocol (TCP) port numbers to filter packets such as VoIP from web browsing traffic, so that each can be sent down the respective bearer with appropriate QoS. An Uplink TFT (UL TFT) associated with each bearer in the UE, filters IP packets to EPS bearers in the uplink direction. A Downlink TFT (DL TFT) in the P-GW is a similar set of downlink packet filters. As part of the procedure by which a UE attaches to the network, the IE is assigned an IP address by the PGW and at least one bearer is established, called the default bearer, and it remains established through the lifetime of the PDN connection, in order to provide the UE with always-on IP connectivity to that PDN. The initial bearer-level QoS parameter values of the default bearer are assigned by the MME, based on subscription data retrieved from the HSS. The PCEF may change these values in interaction with the PCRF or according to local configuration. Additional bearers called dedicated bearers can also be established at any time during or after completion of the attach procedure. A dedicated bearer can be either GBR or Non-GBR (the default bearer always has to be a non-GBR bearer since it is permanently established). The distinction between default and dedicated bearers should be transparent to the access network (i.e. E-UTRA). Each bearer has an associated QoS, and if more than one bearer is established for a given UE, Then each bearer must also be associated with appropriate TFTs. These dedicated bearers could be established by the network, based for example on a trigger from the IMS domain, or they could be requested by the UE. The dedicated bearers for a UE may be provided by one or more P-GWs. The bearer-level QoS parameter values for dedicated bearers are received by the P-GW from the PCRF and forwarded to the S-GW. The MME only transparently forwards those values received from the S-GW over the S11 interface to the E-UTRAN (PALAT, S. and Godin, P., 2011, pp.25-55).

2.1.2. The E-UTRAN radio network

The E-UTRAN system is depicted in Figure 6. It handles the radio communication between the mobile device and the evolved packet core and just has one part, the evolved Node B (eNB). Each eNB is a base station that controls the mobiles in one or more cells. A mobile communicates with just one base station and one cell at a time, so there is no equivalent of the soft handover state from UMTS. The base station that is communicating with a mobile is known as its serving eNB. The eNB has two main functions. Firstly, the eNB sends radio transmissions to all its mobiles on the downlink and receives transmissions from them on the uplink, using the analogue and digital signal processing functions of the LTE air interface. Secondly, the eNB controls the low-level operation of all its mobiles, by sending them signaling messages such as handover commands that relate to those radio transmissions. In carrying out these functions, the eNB combines the earlier functions of the Node B and the radio network controller, to reduce the latency that arises when the mobile exchanges information with the network. Each base station is connected to the EPC with the S1 interface. It can also be connected to nearby base stations by the X2 interface, which is used for signaling and packet forwarding during handover (COX, C., 2014).

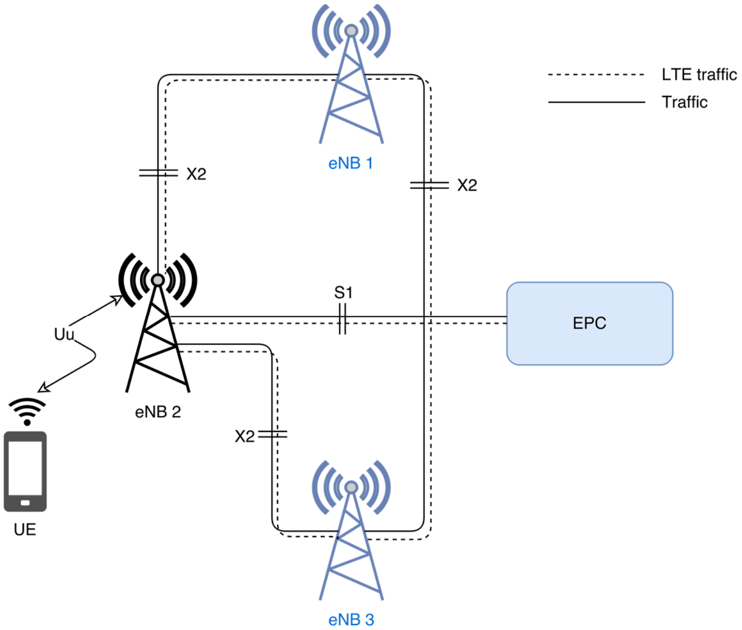


Figure 6. Architecture of the E-UTRAN radio access network

The architecture of the Evolved UMTS terrestrial radio access network is same as the 3G UMTS radio access network (Figure 6). The X2 interfaces serve for eNB to eNB communication and are optional. Nearby base-stations only need to communicate to each other because of handovers, and distant base stations do not need to interact at all. Another reason the X2 interface is optional is because one X2 communication can be carried via two S1 instances in a slower manner, due to the signaling through the EPC. X2 interfaces can be configured automatically via self-optimization parameter functions (COX, C., 2014).

A. *Transport network*

In a usual scenario, the S1 and X2 interfaces do not represent direct physical connections. As represented in Figure 7, the information is routed across an underlying IP transport network (which is usually optical). In reality, the base stations and the components have their IP addresses, which enables them to communicate between each other, and therefore the X2 and S1 are best understood as logical connections through which the devices exchange information (COX, C., 2014).

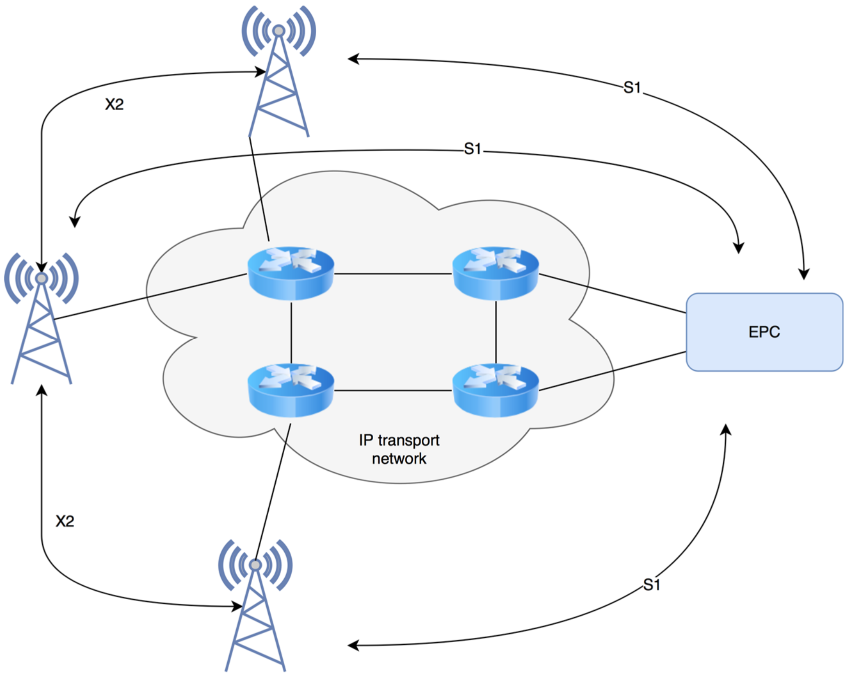


Figure 7. Architecture of the E-UTRAN transport network

B. *Physical, transport and logical channels*

In order to understand the structure in later stages of the virtualized OpenAirInterface5G LTE network, some elementary concepts of currently-employed LTE channels are described. Each channel has parameters that need to be set up for the network to run properly. In case of misconfiguration of some parameters of a channel, then the stability of the access stratum will suffer. The channels are divided in three categories, and they represent a base from which a UE establishes connection to the network. To be able to transmit data across the air interface, LTE defines various channels. These channels are employed to differentiate the different types of data and transport it through the radio access network. Namely, the different channels allow interfacing to the higher layers within the LTE protocol stack and logically define the segregation of the data. To efficiently support various QoS classes of services, LTE adopts a hierarchical channel structure.

There are three different channel types defined in LTE—logical channels, transport channels, and physical channels, each associated with a service access point (SAP) between different layers. These channels are used by the lower layers of the protocol stack to provide services to the higher layers (GHOSH, A. et al., 2011):

- **Physical channels:** Each physical channel corresponds to a set of resource elements in the time-frequency grid that carry information from higher layers. The basic entities that make a physical channel are resource elements and resource blocks. A resource element is a single subcarrier over one OFDM symbol, and typically this could carry one (or two with spatial multiplexing) modulated symbol(s). A resource block is a collection of resource elements and in the frequency domain this represents the smallest quanta of resources that can be allocated (GHOSH, A. et al., 2011).

- **Transport channels***:* The transport channels are used by the PHY to offer services to the MAC. A transport channel is basically characterized by how and with what characteristics data is transferred over the radio interface, that is, the channel coding scheme, the modulation scheme, and antenna mapping. Compared to UTRA/HSPA, the number of transport channels in LTE is reduced since no dedicated channels are present (GHOSH, A. et al., 2011).

- **Logical channels:** Logical channels are used by the MAC to provide services to the RLC. Each logical channel is defined based on the type of information it carries. In LTE, there are two categories of logical channels depending on the service they provide: *logical control channels* and *logical traffic channels* (GHOSH, A. et al., 2011).

Each channel categories can be found separately on the uplink (UL) and the downlink (DL). On the downlink, LTE has a variety of channels, each offering different functionality (see Figure 8).

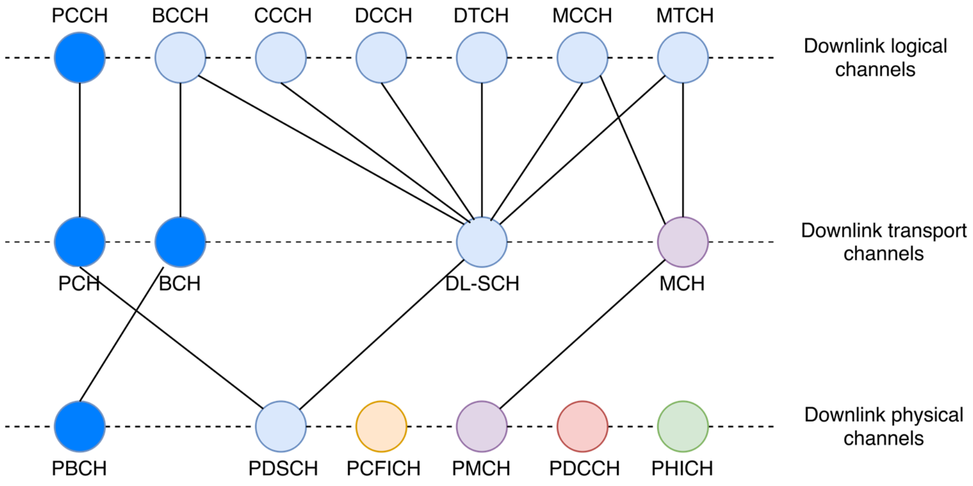

PCCH
BCCH
CCCH
DCCH
DTCH
MCCH
MTCH
Downlink logical channels
PCH
BCH
DL-SCH
MCH
Downlink transport channels
PBCH
PDSCH
PCFICH
PMCH
PDCCH
PHICH
Downlink physical channels

Figure 8. LTE Downlink channels

Starting from the physical channels, the differences are pointed out with regard the different requirements and operation:

- ***Physical Broadcast Channel (PBCH):*** This physical channel carries system information for UEs requiring accessing the network. It only carries what is termed Master Information Block, MIB, messages. The modulation scheme is always QPSK and the information bits are coded, and rate matched. The bits are then scrambled using a scrambling sequence specific to the cell to prevent confusion with data from other cells. The MIB message on the PBCH is mapped onto the central 72 subcarriers or six central resource blocks regardless of the overall system bandwidth. A PBCH message is repeated every 40ms, i.e. one TTI of PBCH includes four radio frames. The PBCH transmissions has 14 information bits, 10 spare bits, and 16 CRC bits (POOLE, I., 2017).

- ***Physical Downlink Shared Channel (PDSCH):*** As the name implies, The PDSCH channel is the main data bearing channel which is allocated to users on a dynamic and opportunistic basis. The PDCH is also used to transmit broadcast information not transmitted on the PBCH which include System Information Blocks (SIB) and paging & RRC signaling messages. PDSCH is also used to transfer application data. There are two types of messages being transmitted through the PDSCH channel:
    - *Paging messages.* These are broadcast using PDSCH channel. LTE UE in RRC IDLE mode monitor PDCCH for paging indications. Based on trigger, it will decode the paging message carried in PDSCH RBs.
    - *Downlink RRC Signaling messages.* These are carried by PDSCH. Signaling Radio Bearers (SRB) will use PDSCH. Every connection usually will have its own set of SRB (POOLE, I., 2017).

- ***Physical Control Format Indicator Channel (PCFICH):*** This channel is used at the starting of each 1ms subframe. It provides information about number of symbols used for PDCCH transmission. The signaling values for PCFICH depends upon channel bandwidth. The same is mentioned in the following Table 4 for different LTE channel bandwidths.

Table 4. PCFICH values for different channel bandwidths (RF WIRELESS WORLD, 2012)

	Channel Bandwidth					
	1.4 MHz	**3MHz**	**5MHz**	**10 MHz**	**15 MHz**	**20 MHz**
PCFICH values	2, 3, 4			1, 2, 3		

As denoted, 1.4MHz requires more time domain symbols compared to other channel bandwidths due to less carriers in frequency domain. Signaling value depends on eNodeB RRM (Radio Resource Management). It is directly connected to the number of active connections. Hence PDCCH signaling increases parallelly with the number of active connections (RF WIRELESS WORLD, 2012).

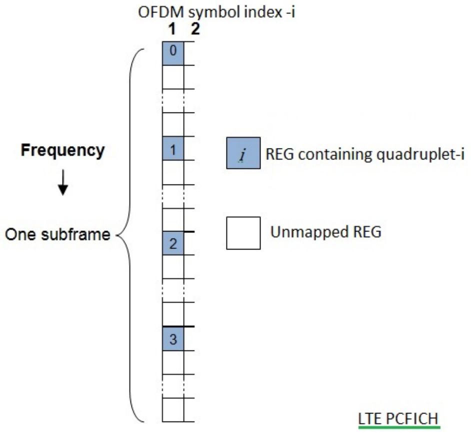


Figure 9. PCFICH subframe (RF WIRELESS WORLD, 2012)

From the Figure 9 it can be perceived that LTE PCFICH channel occupies 16 REs (Resource Elements) in first OFDMA symbol of each 1ms frame. PCFICH uses QPSK modulation and hence 16 REs will occupy 32 bits. This 16REs are divided into 4 quadruplets. The position of which in first OFDMA symbol depends on Channel BW and Physical layer cell identity.

As mentioned, each quadruplet is mapped to REG (Resource Element Group) with subcarrier index k = k' and is as per following equation:

$$k' = (Nsc\ per\ \frac{RB}{2}) \cdot (N_{CellID} mod2\ N_{DL-RB})$$

The rest of three quadruplets are mapped to REGs spaces at intervals of ($N_{DL\text{-}RB}$/2) * (Nsc per RB /2) from the first quadruplet and each other. This way LTE PCFICH channel information is spread across entire subframe as shown. The PCFICH carries CFI (Control Format Indicator) which has a value ranging from 1 to 3. This CFI is coded to occupy complete PCFICH capacity of 32 bits (RF WIRELESS WORLD, 2012).

Actual value = signaled value + 1 (for 1.4 MHz BW)

Actual value = signaled value (for all the channel BWs)

- ***Physical Multicast Channel (PMCH):*** This channel defines the physical layer structure to carry Multimedia Broadcast and Multicast Services (MBMS). This control channel occupies the first 1, 2, or 3 OFDM symbols in a subframe extending over the entire system bandwidth. For PMCH channel QPSK, 16QAM, 64QAM modulations are used. It carries MCH. Multicast Channel (MCH) characterized by:
  - o requirement to be broadcast in the entire coverage area of the cell
  - support for MBSFN combining of MBMS transmission on multiple cells
  - support for semi-static resource allocation e.g. with a time frame of a long cyclic prefix

In Downlink, MTCH logical channel can be mapped to DL-SCH and MCH transport channels (RF WIRELESS WORLD, 2012).

- ***Physical Downlink Control Channel (PDCCH):*** The main purpose of this physical channel is to carry mainly scheduling information of different types:
    - Downlink resource scheduling
    - Uplink power control instructions
    - Uplink resource grant
    - Indication for paging or system information

  The PDCCH contains a message known as the Downlink Control Information, DCI which carries the control information for a particular UE or group of UEs. The DCI format has several different types which are defined with different sizes. The different format types include: Type 0, 1, 1A, 1B, 1C, 1D, 2, 2A, 2B, 2C, 3, 3A, and 4 (POOLE, I., 2017).

- ***Physical Hybrid ARQ Indicator Channel (PHICH):*** As the name indicates, this channel is used to report the Hybrid ARQ status. It carries the HARQ ACK/NACK signal indicating whether a transport block has been correctly received. The HARQ indicator is 1 bit long - "0" indicates ACK, and "1" indicates NACK. The PHICH is transmitted within the control region of the subframe and is typically only transmitted within the first symbol. If the radio link is poor, then the PHICH is extended to a number symbols for robustness (POOLE, I., 2017).

  The transport channels are used by the PHY to offer services to the MAC layer. There are four LTE downlink transport channels (POOLE, I., 2017) (GHOSH, A. et al., 2011):

- ***Broadcast Channel (BCH):*** A downlink channel associated with the BCCH logical channel and is used to broadcast system information over the entire coverage area of the cell. It has a fixed transport format defined by the specifications (GHOSH, A. et al., 2011).

- ***Downlink Shared Channel (DL-SCH):*** Used for transmitting the downlink data, including both control and traffic data, and thus it is associated with both logical control and logical traffic channels. It supports H-ARQ, dynamic link adaption, dynamic and semi-persistent resource allocation, UE discontinuous reception, and multicast/broadcast transmission. The concept of shared channel transmission originates from HSDPA, which uses the *High-Speed Downlink Shared Channel* (HS-DSCH) to multiplex traffic and control information among different UEs. By sharing the radio resource among different UEs the DL-SCH is able to maximize the throughput by allocating the resources to the optimum UEs (GHOSH, A. et al., 2011).

- ***Paging Channel (PCH*):** Associated with the PCCH logical channel. It is mapped to dynamically allocate physical resources and is needed for broadcast over the entire cell coverage area. It is transmitted on the Physical Downlink Shared Channel (PDSCH) and supports UE discontinuous reception (GHOSH, A. et al., 2011).

- ***Multicast Channel (MCH):*** Associated with MCCH and MTCH logical channels for the multicast/broadcast service. It supports *Multicast/Broadcast Single Frequency Network* (MBSFN) transmission, which transmits the same information on the same radio resource from multiple synchronized base stations to multiple UEs (GHOSH, A. et al., 2011).

LTE defines seven logical downlink channels:

- ***Broadcast Control Channel (BCCH):*** A downlink common channel used to broadcast system control information to the mobile terminals in the cell, including downlink system bandwidth, antenna configuration, and reference signal power. Due to the large amount of information carried on the BCCH, it is mapped to two different transport channels: The Broadcast Channel (BCH) and the Downlink Shared Channel (DL-SCH) (GHOSH, A. et al., 2011).

- ***Multicast Control Channel (MCCH):*** A point-to-multipoint downlink channel used for transmitting control information to UEs in the cell. It is only used by UEs that receive multicast/broadcast services (GHOSH, A. et al., 2011).

- ***Paging Control Channel (PCCH):*** A downlink channel that transfers paging information to registered UEs in the cell, for example, in case of a mobile-terminated communication session (GHOSH, A. et al., 2011).

- ***Common Control Channel (CCCH):*** A bi-directional channel for transmitting control information between the network and UEs when no RRC connection is available, implying the UE is not attached to the network such as in the idle state. Most commonly the CCCH is used during the random-access procedure (GHOSH, A. et al., 2011).

- ***Dedicated Control Channel (DCCH):*** A point-to-point, bi-directional channel that transmits dedicated control information between a UE and the network. This channel is used when the RRC connection is available, that is, the UE is attached to the network (GHOSH, A. et al., 2011).

- ***Dedicated Traffic Channel (DTCH):*** A point-to-point channel, dedicated to one UE for the transfer of user information. A DTCH can exist in both uplink and downlink channels (GHOSH, A. et al., 2011).

- ***Multicast Traffic Channel (MTCH):*** A point-to-multipoint downlink channel for transmitting traffic data from the network to the UE. This channel is only used by UEs that receive MBMS. It is associated with the multicast/broadcast service (GHOSH, A. et al., 2011).

The Figure 10 depicts the classification of the channels instituted at the uplink. The logical channels will be omitted, because they are also found in the downlink.

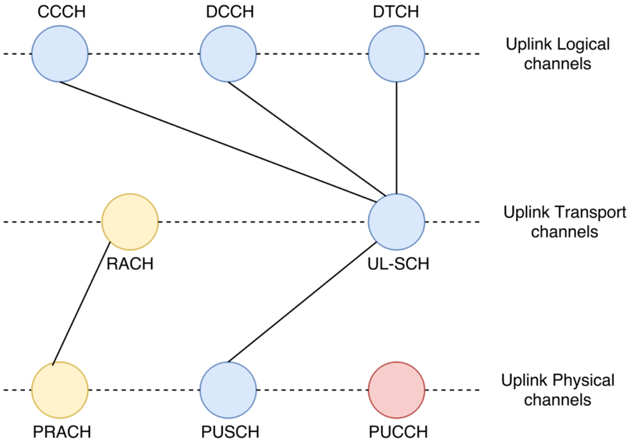


Figure 10. LTE Uplink channels

As on Figure 10, LTE defines three physical channels:

- ***Physical Uplink Control Channel (PUCCH):*** It carries uplink control information including Channel Quality Indicators (CQI), ACK/NAKs for H-ARQ in response to downlink transmission, and uplink scheduling requests. This LTE channel is used to carry UCI (Uplink Control Information). UCI can also be transported using PUSCH channel. An LTE UE can never transmit both PUCCH and PUSCH during the same subframe. If UE has application data OR RRC signaling, then UCI is carried over PUSCH. If UE does not have any application data OR RRC signaling, then UCI is carried over PUCCH (GHOSH, A. et al., 2011).

  As a stand-alone uplink physical channel, the PUCCH control signaling channel consists the following:

  - HARQ ACK/NACK
  - CQI-channel quality indicators
  - MIMO feedback - RI (Rank Indicator), PMI (Precoding Matrix Indicator)
  - scheduling requests for uplink transmission
  - BPSK or QPSK used for PUCCH modulation

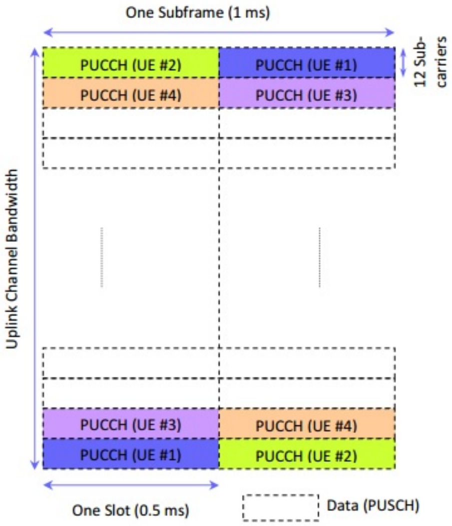


Figure 11. PUCCH subframe structure (RF WIRELESS WORLD, 2012)

PUCCH consists of 1 RB/transmission at one end of the system bandwidth which is followed by another RB in the following slot (at opposite end of the channel spectrum). This makes use of frequency diversity with 2dB estimated gain. A PUCCH Control Region encompasses every two such RBs (see Figure 11 and Table 5).

Table 5. Composition of the PUCCH control region (RF WIRELESS WORLD, 2012)

System BW in MHz	**1.25**	**2.5**	**5**	**10**	**15**	**20**
PUCCH control region	1	2	4	8	12	16
No. of resource blocks	2	4	8	16	24	32

The standard specifies 6 LTE PUCCH formats as mentioned in the Table 6 below. As stated, PUCCH format 2a and 2b are not applicable for extended CP.

Table 6. LTE PUCCH formats (RF WIRELESS WORLD, 2012)

LTE PUCCH Format	Modulation index	No. of bits per subframe	No. of Res occupied (Normal CP)	No. of Res occupied (Extended CP)
1	/	/	48+48=96 OR 48+36=84	
1a	BPSK	1	48+48=96 OR 48+36=84	
1b	QPSK	2	48+48=96 OR 48+36=84	
2	QPSK	20	120	
2a	QPSK+BPSK	21	120	Not applicable
2b	QPSK+BPSK	22	120	Not applicable

The LTE PUCCH channel is distributed 2 RBs at the edges of channel BW (Table 7). Each PUCCH transmission occupy 1 RB on each side of the channel bandwidth. These two RBs are distributed across two-time slots. RB numbering for PUCCH starts on outside edges and increases inwards (RF WIRELESS WORLD, 2012).

PUCCH has RBs allocated at the edge of channel BW to avoid fragmenting.

Table 7. Information carried by PUCCH format (RF WIRELESS WORLD, 2012)

LTE PUCCH format	No. of bits per subframe	Normal CP	Extended CP
1	/	Scheduling request	
1a	1	1 x HARQ-ACK OR 1 x HARQ-ACK +SR	
1b	2	2 x HARQ-ACK OR 2 x HARQ-ACK +SR	
2	20	CQI	CQI OR HARQ-ACK+CQI
2a	21	1 x HARQ-ACK + CQI	/
2b	22	2 x HARQ-ACK + CQI	/

- ***Physical Uplink Shared Channel (PUSCH):*** Carries user data and higher layer signaling. It corresponds to the UL-SCH transport channel (GHOSH, A. et al., 2011). Namely, the channel is used to carry RRC signaling messages, UCI (uplink Control Information) and application data. Uplink RRC messages are carried using PUSCH. SRB use PUSCH and each connection will have its unique SRB. The LTE PUSCH channel contains user information data and carries both user data as well as control signal data. Control information carried, can be MIMO related parameters and transport format indicators. The control data information is multiplexed with the user information before DFT spreading module in the uplink SC-FDMA physical layer. PUSCH supports QPSK, 16QAM and 64QAM (optional). The LTE eNodeB selects suitable modulation based on adaptation algorithm. UCI is transmitted using PUSCH instead of PUCCH when there is RRC and application data to be transferred at the same time instant (RF WIRELESS WORLD, 2012).

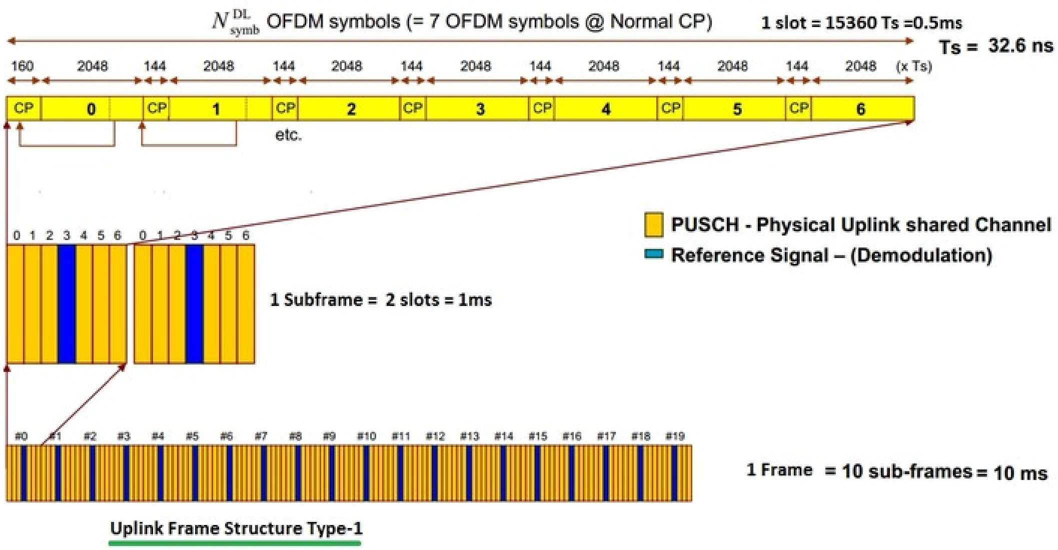

N_{symb}^{DL} OFDM symbols (= 7 OFDM symbols @ Normal CP)
1 slot = 15360 Ts =0.5ms
Ts = 32.6 ns
160 2048 144 2048 144 2048 144 2048 144 2048 144 2048 144 2048 (x Ts)
CP 0 CP 1 CP 2 CP 3 CP 4 CP 5 CP 6
etc.
PUSCH - Physical Uplink shared Channel
Reference Signal – (Demodulation)
0 1 2 3 4 5 6 0 1 2 3 4 5 6
1 Subframe = 2 slots = 1ms
#0 #1 #2 #3 #4 #5 #6 #7 #8 #9 #10 #11 #12 #13 #14 #15 #16 #17 #18 #19
1 Frame = 10 sub-frames = 10 ms
Uplink Frame Structure Type-1

Figure 12. PUSCH channel frame structure (RF WIRELESS WORLD, 2012)

As according to Figure 12, modulation type is conveyed to UE using PDCCH DCI format-0. This CI also signals RB allocation and TB size. LTE PUSCH channel uses QPSK when TTI bundling is enabled. If eNodeB directs UE to use 64QAM, but if UE does not support it, then 16QAM modulation type is selected (RF WIRELESS WORLD, 2012).

- ***Physical Random-Access Channel (PRACH):*** This channel carries the random-access preamble sent by UEs (GHOSH, A. et al., 2011). As shown a random-access preamble includes a CP, a sequence and a guard time (RF WIRELESS WORLD, 2012). There are 4 different RA (random access) preamble formats defined in LTE FDD specifications. The same have been mentioned in the Table 8 below. It consists of different preamble and CP duration to accommodate different cell sizes. The preamble format to be used in a specific cell is informed to the UE using PRACH configuration index. This is broadcasted in SIB-2. PRACH configuration index also indicates SFN and subframes. This gives the exact position of random access preamble.

Table 8. Random-access preamble formats (RF WIRELESS WORLD, 2012)

LTE PRACH preamble format	CP length	Sequence length	Guard time	Total length	Guard time equiv. dist.	Typical max. cell range
0	0.10ms	0.8ms	0.10ms	1ms	30Km	15Km
1	0.68ms	0.8ms	0.52ms	2ms	156km	78km
2	0.2ms	1.6ms	0.2ms	2ms	60Km	30Km
3	0.68ms	1.6ms	0.72ms	3ms	216Km	108Km

The preamble uses subcarrier spacing of 1.25 KHz instead of 15 KHz. As represented in Figure 13 and Figure 14, the random-access preamble occupies 1, 2 or 3 subframes in the time domain (1, 2, 3 ms) and 839 subcarriers in frequency domain (1.05 MHz). There is a 15 KHz guard band on both the sides and hence it uses total of 1.08MHz (equal to 6 RBs). The position of LTE random access preamble is defined by PRACH frequency offset parameter carried in SIB-2 (RF WIRELESS WORLD, 2012).

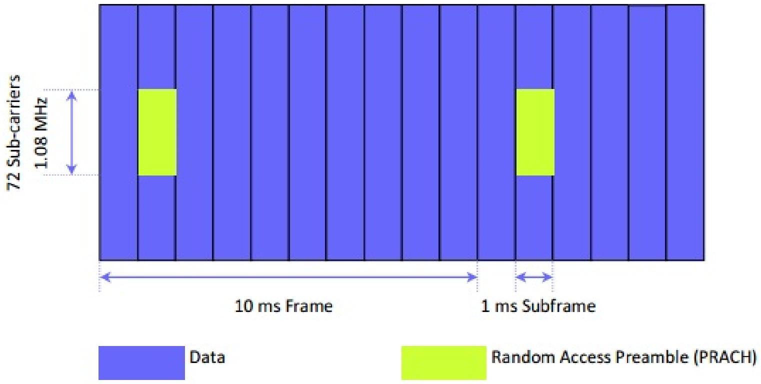


Figure 13. Position of PRACH in uplink frame (RF WIRELESS WORLD, 2012)

There is a maximum of one random access preamble in a subframe but more than one UEs can use it. Multiple UEs using same preamble resource allocations are differentiated by their unique preamble sequences. Accordingly, maximum of 64 preamble sequences are divided into group-A and group-B. LTE UE selects the sequence from these two groups based on size of uplink packet and radio conditions. This helps eNodeB to calculate PUSCH resources needed for UE uplink transfer. Sequences in Group-A are used for smaller size packets or larger size packets in poor radio conditions. Sequences in Group-B are used for larger size packets in good radio conditions (RF WIRELESS WORLD, 2012).

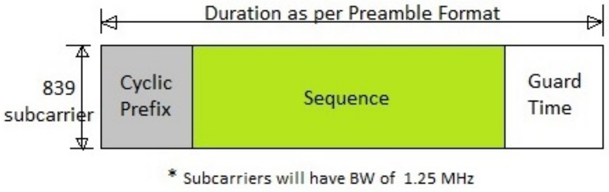

Duration as per Preamble Format
839 subcarrier
Cyclic Prefix
Sequence
Guard Time
* Subcarriers will have BW of 1.25 MHz

Figure 14. Structure of random access preamble (RF WIRELESS WORLD, 2012)

At the transport uplink layer, LTE has two dedicated channels: RACH and UL-SCH.

- ***Random Access Channel (RACH):*** This is the first message from UE to eNB when the device powers on. Even though the name designations for the channel are different in all cellular technology (CDMA, GSM, WCDMA, LTE) there is a specific signal that performs the same function. In CDMA, it is appointed as 'Access Probe', while in GSM it is known as 'Channel Request', and in WCDMA / LTE referred to as 'RACH'. From the aspect of eNB, seemingly the signal is received from the UE in almost random character (i.e. in Random timing, Random Frequency and in Random Identification) because it doesn’t have information when a user turns on the UE (Therefore, it is not completely random, as there is a certain range of agreement between UE and Network about the timing, frequency location and possible identification. However, in large scale it appears randomly). In terms of Radio Access Network implementation, handling RACH is one of the most challenging jobs. The RACH channel is shared, and therefore, there is a high probability that two or more devices transmit simultaneously. This can lead to transmission collisions in the medium and the access to the network can be restricted accordingly. In GSM for example, the upper limit of number of devices transmitting in one RACH timeslot is not specified. If there are collisions, then the device waits for random period before re-transmitting a RACH signal again (COX, C., 2014). The parameters for RACH access procedure include: access slots, preamble scrambling code, preamble signatures, and spreading factor for data part, available signatures and subchannels for each Access Service Class (ASC) and power control information. The Physical channel information for PRACH is broadcasted in SIB5/6 and the fast-changing cell parameters such as uplink interference levels used for open loop power control and dynamic persistence value are broadcasted in SIB7. RACH access procedure follows slotted-ALOHA approach with fast acquisition indication combined with power ramping in steps (KUMAR, S., 2017).

Maximum of 16 different PRACHs can be offered in a cell, in FDD, the various PRACHs are distinguished either by employing different preamble scrambling codes or by using common scrambling code with different signatures and subchannels. Within a single PRACH, a partitioning of the resources between the maximum 8 ASC is possible, thereby providing a means of access prioritization between ASCs by allocating more resources to high priority classes than to low priority classes. SC 0 is assigned highest priority and ASC 7 is assigned lowest priority. SC 0 shall be used to make emergency calls which has more priority. The available 15 access slots are split between 12 RACH subchannels. The RACH transmission consists of two parts, namely preamble transmission and message part transmission. The preamble part is 4096 chips, transmitted with spreading factor 256 and uses one of 16 access signatures and fits into one access slot. ASC is defined by an identifier i that defines a certain partition of the PRACH resources and is associated with persistence value $P(i)$. The persistence value for $P(0)$ is always set to one and is associated with ASC 0. The persistence values for others are calculated from signaling. These persistence values control the RACH transmissions. To start a RACH procedure, the UE selects a random number r, between 0 and 1 and if $r \leq P(i)$, the physical layer PRACH procedure is initiated else it is deferred by 10 ms and then the procedure is started again. Once the UE PRACH procedure is initiated, then the real transmission takes place (KUMAR, S., 2017).

As described above, the preamble part transmission starts first. The UE picks one access signature of those available for the given ASC and an initial preamble power level based on the received primary CPICH power level and transmits by picking randomly one slot out of the next set of access slots belonging to one of the PRACH subchannels associated with the relevant ASC. The UE then waits for the proper access indicator sent by the network on the downlink Acquisition Indicator Channel (AICH) access slot which is paired with the uplink access slot on which the preamble was sent. There are 3 scenarios possible (KUMAR, S., 2017):

a) If the Acquisition Indication (AI) received is a positive acknowledgement, then UE sends the data after a predefined amount of with a power level which is calculated from the level used to send the last preamble (KUMAR, S., 2017).
b) IF the AI received is a negative acknowledgement, the UE stops with the transmission and hands back control to the MAC layer. After a back off period, the UE will regain access according to the MAC procedure based on persistence probabilities (KUMAR, S., 2017).
c) If no acknowledgement is received, then it is considered that network did not receive the preamble. If the maximum number of preambles that can be sent during a physical layer PRACH procedure is not exceeded, the terminal sends another preamble by increasing

the power in steps. The ability of the UE to increase its output power, in terms of steps to a specific value is called as open loop power control. RACH follows open loop power control (KUMAR, S., 2017)

- ***Uplink Shared Channel (UL-SCH):*** The UL-SCH is used to transmit RRC signaling and application data. UCI can be added during physical layer processing before mapping on PUSCH physical channel. TBs belong to UL-SCH has variable size. The physical-layer model for Uplink Shared Channel transmission is described based on the corresponding physical layer processing chain. It should be noted that, in case PUSCH, the scheduling decision is partly made at the network side, if there is no blind decoding it is fully done at the network side. The uplink transmission control in the UE then configures the uplink physical-layer processing, based on uplink transport-format and resource-assignment information received on the downlink. As in Figure 15, the processing steps that are relevant for the physical-layer model, e.g. in the sense that they are configurable by higher layers, are highlighted in blue (3GPP, 2007-2012). After the UL-SCH passes through the modules as shown in the figure, UL-SCH code word is formed. This code word is modulated and later used to generate SC-FDMA signal. UL-SCH codeword is transmitted during 1ms subframe.

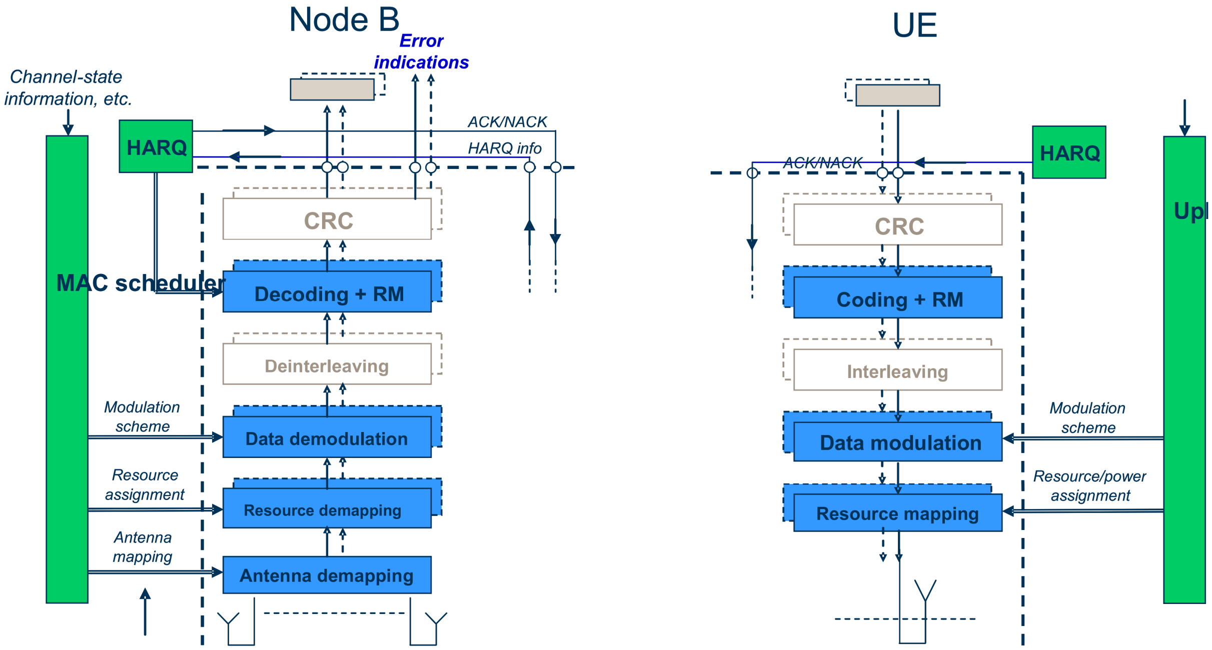


Figure 15. Physical-layer model for UL-SCH transmission (3GPP, 2007-2012)

2.1.3. Diversity processing

The methods which improve the robustness and reliability of a message signal by implementing two or more communication channels with various characteristics are denoted as a *diversity scheme* (AVIAT NETWORKS, 2017). Specifically, diversity techniques are used for tackling fading and co-channel interference and avoiding error bursts. In other words, a same signal can be transmitted in multiple versions, and then received and combined at the receiver. A redundant forward error correction code can be implemented, and different parts of the message can be transmitted over different channels. The diversity techniques are utilizing the multipath propagation, which is a known problem in wireless systems and discussed accordingly in the following chapters; which results in a diversity gain, measured in decibels (dB). There are several diversity techniques (MOLISCH, A. F., 2011):

- *Time diversity* – Multiple versions of a single signal are transmitted at different time intervals, which adds a redundant forward error correction code (FEC) and the message is spread in terms of bit-interleaving before it is sent. This aids avoidance of error bursts and simplifies the error correction procedure.
- *Frequency diversity* – Refers to the application of various frequency manipulation techniques for reaching improved spectral efficiency (an example for frequency diversity technique is OFDM – Orthogonal Frequency Division Multiplexing).
- *Space diversity* – Also known as *antenna diversity*, refers to the spatial utilization and combining of antennas in order to achieve better spectral efficiency. For example, such techniques involve the MIMO (Multiple-Input-Multiple-Output), beamforming or space-time coding.
- *Polarization diversity* – When multiple versions of a same signal are transmitted and received through antennas with different polarization.
- *Multiuser diversity* – In multiuser diversity techniques, there are methods such as opportunistic user scheduling that selects the best user candidate as a receiver, according to the qualities of each channel between the transmitter and each receiver. The channel quality information is then spread by the receiving user to the transmitter using restrictive levels of resolution, after which the transmitter (base station) can process the multiuser diversity.
- *Cooperative diversity* – Widely implemented in Distributed Antenna Systems (DAS) and attains antenna diversity gain by means of the cooperation amongst antennas belonging to each node (YEO, Y. et al., 2018).

2.1.4. Orthogonal Frequency Division Multiplexing (OFDM) and Orthogonal Frequency Division Multiple Access (OFDMA) for the downlink channel in LTE

The Orthogonal Frequency Division Multiple Access technique is based on the Orthogonal Frequency Division Multiplexing (OFDM). Namely, OFDM (3GPP, 2017) [according to 3GPP Specification 25.892] is one of the most prominent advances in access techniques. It enables bigger transmission rates with a significant equalization and detection convolutions. High transmission is accomplished through modulating a set of narrowband orthogonal subcarriers. An OFDM block is created as shown in Figure 16. The sequence of L-modulated symbols, $x0, x1, \ldots, xL-1,$ are converted into L parallel streams before taking the N -point Inverse Fast Fourier Transform (IFFT) (COCHRAN, W.T. et al., 1967) of each. The possible mismatch between L and N is overcome by zero padding the remaining $N - L$ inputs of the IFFT block. Next, the N outputs, $s0, s1, \ldots, sN-1$ are converted back to a serial stream before adding the Cyclic Prefix (CP). Finally, the resulting OFDM block is converted to its analog form prior to sending it over the channel (TAHA, A-E. M. et al., 2012).

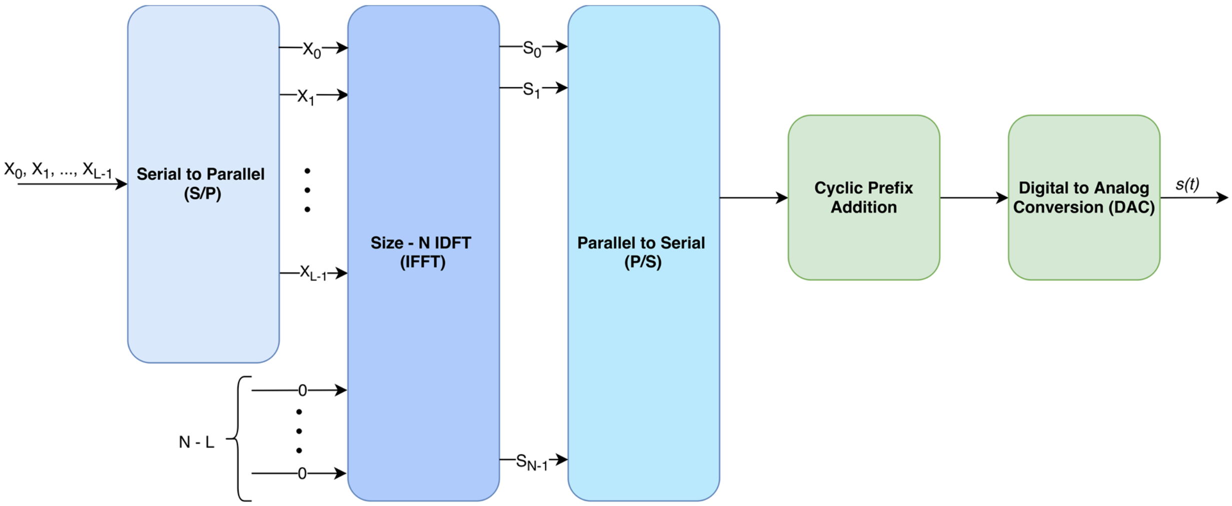


Figure 16. OFDM modulation with IFFT

With this architecture, an OFDM block can resist the Inter-Carrier Interference (ICI) by empowering orthogonal subcarriers, that is, as a result of using the IFFT (Inverse Fast Fourier Transform). It is also efficient in extenuating the channel time dispersion by introducing the CP (Cyclic Prefix). Truthfully, the insertion of the CP is a generally used method to produce a so-called guard period between consecutive OFDM symbols. The CP is basically a reiteration of the vestige of the preceding OFDM symbol. The span of this reiteration is made long enough to surpass the channel delay spread, hence extenuating the channel delay spread instigates Inter-Symbol-Interference (ISI). Additionally, the detection process becomes a circular convolution process which augments the signal detection capabilities and abridges the equalization procedure. OFDM Demodulation overturns the above-mentioned procedures. After converting the received signal back into the digital domain, the CP is detached. Following that, the signal is transformed into a parallel N data streams before performing an N -point FFT (Fast Fourier Transform) (COCHRAN, W.T. et al., 1967). Finally, the sequence is returned into a serial one. These steps are represented in Figure 17.

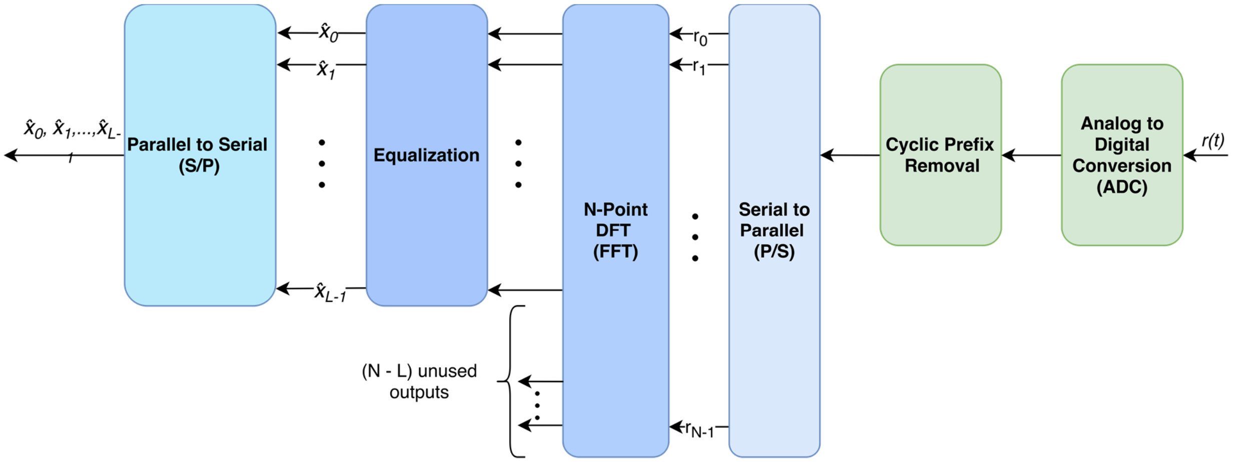


Figure 17. OFDM demodulation

Despite the many advantages of OFDM, actual implementations incurred some challenges. The most famous one is the high Peak-to-Average Power Ratio (PAPR) problem (GAMAGE, H. et al., 2016). Principally, high PAPR, which results from the coherent addition of the modulated subcarriers, reduces the efficiency of the power amplifier. The high PAPR also sophisticates the Analog to Digital (ADC) and Digital to Analog (DAC) conversion processes. While these two disadvantages can be overcome at the base station side, they form a serious challenge to the battery-powered Mobile Station (MS). Consequently, 3GPP replaced OFDM at the uplink in their IMT-Advanced proposal by SC-FDMA (TAHA, A-E. M. et al., 2012).

OFDM is employed in all modern wireless technologies, such as: IEEE 802.11 Wi-Fi, IEEE 802.16 WiMAX, 3G/UMTS, 4G/LTE, IEEE 802.15.3a Ultra-Wideband (UWB) Wireless PAN, IEEE 802.20 Mobile Broadband Access Technology (MBWA) as well as satellite systems as DVB-RCS Digital Video Broadcast – Return Channel via Satellite, Flash-OFDM cellular systems etc. Also, some wireline technologies are exploiting the benefits of OFDM, such as: ADSL and VDSL broadband access via POTS copper wiring, MoCA (Multimedia over Coax Alliance) networking, PLC Power Line Communication etc. For the purpose of more data services, a system has to provide high spectral efficiency or better spectrum utilization. Another trait that should be pointed out is resiliency to interference, where a system needs to enable excellent performance in unregulated and regulated frequency bands. The *multi-path problem* (CISCO, 2008) degrades a signal in such way that when a radio frequency (RF) signal is transmitted towards the receiver, the general behavior of the RF signal is to grow wider as it is transmitted further. As in Figure 18, on its way, the RF signal encounters objects that reflect, refract, diffract, absorb, scatter or interfere with the signal (BIEHLE, G., 2016). When an RF signal is reflected off an object, multiple wavefronts are created. As a result of these new duplicate wavefronts, there are multiple wavefronts that reach the receiver.

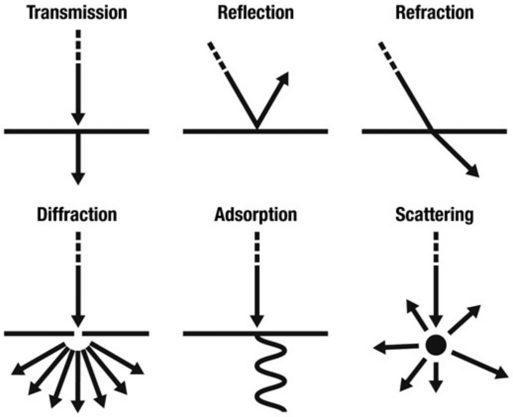
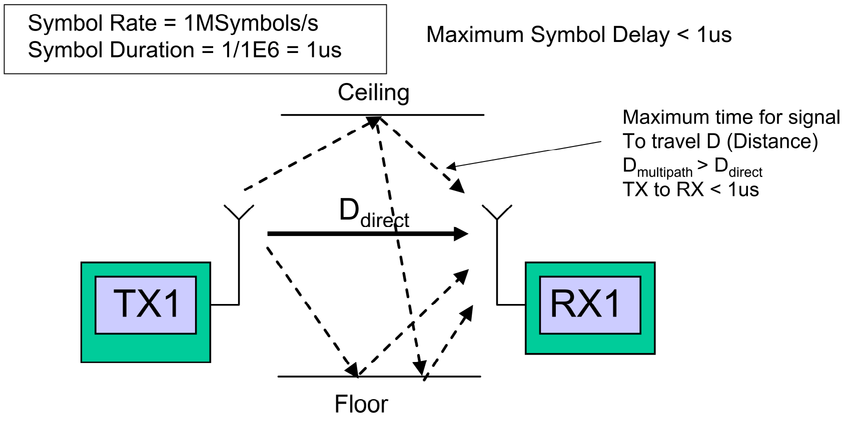


Figure 18. Laws of Reflection, Refraction, Diffraction, Absorption and Scattering (BIEHLE, G., 2016)

Multipath propagation occurs when RF signals take different paths from a source to a destination. A part of the signal arrives at the destination while another part bounces off an obstruction, then proceeds to the destination. As a result of the reflection, parts of the signal encounter delay and travel a longer path to the destination. Multipath can be defined as the combination of the original signal plus the duplicate wavefronts that result from reflection of the waves off obstacles between the transmitter and the receiver. Multipath distortion is a form of RF interference that occurs when a radio signal has more than one path between the receiver and the transmitter. This occurs in cells with metallic or other RF-reflective surfaces, such as furniture, walls, or coated glass (CISCO, 2008).

For example, as in Figure 19, the multi-path problem reflects on a Bluetooth transmitting and receiving systems (KEITHLEY INSTRUMENTS, 2008). With a symbol rate of 1 MSymbols/s, it is noticeable that Bluetooth uses a single carrier to transmit a single symbol at a time. This case is analogous to the 2G - GSM and CDMA systems. For that purpose, to increase the data throughput, the symbol rate has to be increased. With the cumulative symbol rate, parallelly the Multi-path distortion will intensify. For comparative incidence, in contrast to Bluetooth, W-CDMA uses 3.16 MSymbols/sec. If the Multi-path effect is fervent, then a solution would be to reduce the symbol rate by a third, or namely to 300kSymbols/s. This will reduce the data throughput as well, which is not a favored setting.



Symbol Rate = 1MSymbols/s
Symbol Duration = 1/1E6 = 1us
Maximum Symbol Delay < 1us
Ceiling
Maximum time for signal
To travel D (Distance)
$D_{multipath} > D_{direct}$
TX to RX < 1us
D_{direct}
TX1
RX1
Floor

Figure 19. The Multi-path problem (KEITHLEY INSTRUMENTS, 2008)

To tackle the problem of the directly-proportional relation between the symbol rate and Multi-path effect, a solution is to increase the number of carriers from a single one to multiple. The modern technologies such as Wi-Fi, WiMAX or LTE use multiple carriers to provide access to multiple users simultaneously and deliver robust connection in dense environments or locations obstructed by numerous physical objects (Figure 20).

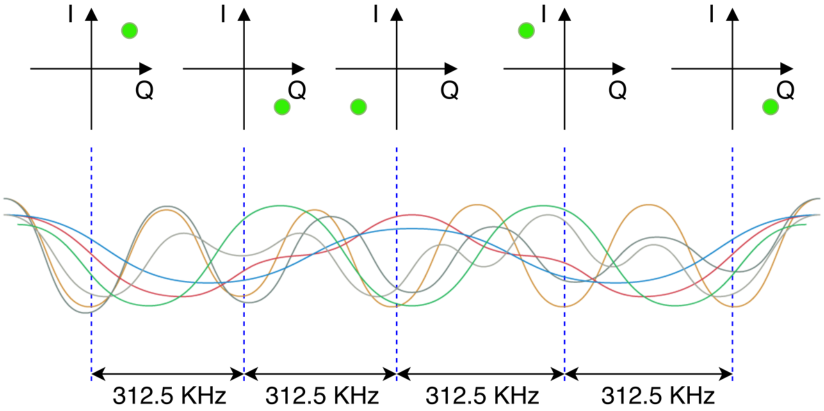


Figure 20. 802.11a-g Wi-Fi multiple carriers with 312.5 KHz sub-spacing

In order to calculate the data rate, it is required to multiply the symbol rate by the number of sub-carriers and the coded bits divided by the subcarriers, all of which is multiplied by ¾ the coding rate. Specifically: $(r_S \cdot c)(\frac{b_c}{c})\frac{3}{4}r_c$, where r_S is the symbol rate, c is the number of subcarriers, $\frac{b_c}{c}$ is the number of coded bits per sub-carrier and r_c is the coding rate. For example, 802.11a-g Wi-Fi uses 250 kbps symbol rate, 48 data sub-carriers, from which 6 coded bits per sub-carrier, which gives the actual data rate of 54 Mbps. A standard Wi-Fi symbol is 4us (useful symbol duration), composed of 3.2us IFFT and 0.8us long guard interval. If using a short guard interval of 0.4us then the total symbol time is 3.6us. The subcarrier spacing is equal to the reciprocal of symbol time. Since the useful symbol duration is 3.2us IFFT then the reciprocal of symbol duration would be $\frac{1\ cycle}{0.0000032\ sec}$ = 312500 cycles/sec, which is 312.5 KHz spacing (ROHLING, Hermann, 2011) .

OFDM subcarrier spacing creates "nulls", canceling out inter-carrier interference (ICI) without the need for guard bands or expensive bandpass filters. OFDM divides a given channel into many narrower subcarriers. The spacing is such that the subcarriers are orthogonal (GOLDBLATT, Robert, 1987), so they won't interfere with one another despite the lack of guard bands between them. This comes about by having the subcarrier spacing equal to the reciprocal of symbol time. All subcarriers have a complete number of sine wave cycles that upon demodulation will sum to zero. This indicates that the spacing of the subcarriers is directly associated to the useful symbol time, or specifically, the amount of time the transmitter spends performing IFFT. Because of this relationship, the resulting synchronization frequency response curves from each subcarrier create signal nulls in the adjacent subcarrier frequencies thus preventing inter-carrier interference (ICI) (GARCIA, M. and Oberli, C., 2009). OFDM is a form of frequency division multiplexing (FDD), which typically requires guard bands between carriers and specialized hardware with bandpass filters to remove interference. OFDM eliminates the need for these which increases spectral efficiency and reduces cost and complexity of the system since all functions can be completed with digital signal processing (DSP) (ELKHODR, M. et al., 2017). As shown in Figure 21, Each 20 MHz channel, whether it's 802.11a/g/n/ac, is composed of 64 subcarriers spaced 312.5 KHz apart. This spacing is chosen because 64-point FFT sampling is used. 802.11a/g for example, employs 48 subcarriers for data, 4 for pilot, and 12 as null subcarriers. 802.11n/ac use 52 subcarriers for data, 4 for pilot, and 8 as null (ROHLING, Hermann, 2011) (ANDREW, A., 2015).

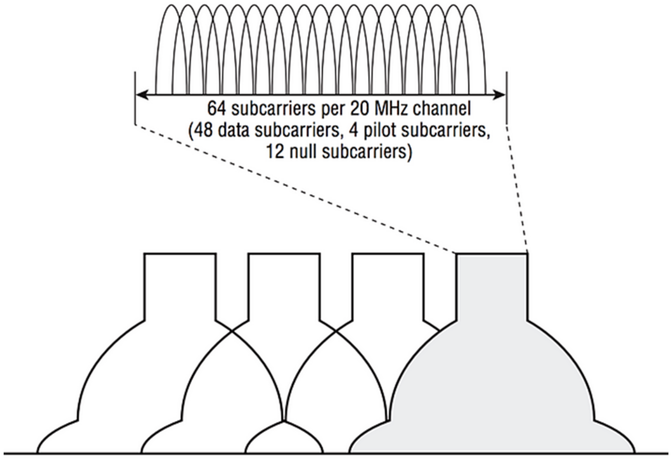

64 subcarriers per 20 MHz channel
(48 data subcarriers, 4 pilot subcarriers,
12 null subcarriers)

Figure 21. OFDM subcarriers in 802.11a-g Wi-Fi (ANDREW, A., 2015)

Another advantage of OFDM is that by using a reduced symbol rate of 250,000 symbols per second, the negative effects of multipath distortion are reduced. Since each symbol occupies more time, there is more resilience to delay spread which is caused by multipath when signal reflections cause multiple copies of the same transmitted symbol to arrive at the receiver at slightly different times. In contrast to the OFDM symbol rate, the 802.11b DSSS and Bluetooth both have over 1M symbols per second, where DSSS has 11M symbols per second if the 'chipping' rate is considered (ROHLING, Hermann, 2011) (ANDREW, A., 2015).

However, multipath also has a negative effect on OFDM, especially when clients are mobile (ANDREW, A., 2015). The orthogonality of the subcarriers can be lost when movement and multipath are present because signal delays (the delay spread) affect the reciprocal relationship of the subcarriers and the useful symbol time (IFFT). Without proper orthogonality between subcarriers, inter-carrier interference (ICI) would result from this doppler shifting. The solution for this, is to include a cyclic prefix (CP) with each symbol, which is part of the guard interval that allows channel estimation and equalization. Thus, contrary to popular belief, the guard interval is actually not empty airtime but actively used for cyclic prefixing to allow proper OFDM operation in a multipath environment . One of the primary reasons for using OFDM as a modulation format within LTE (and many other wireless systems for that matter) is its resilience to multipath delays and spread. However, it is still necessary to implement methods of adding resilience to the system. This helps overcome the inter-symbol interference (ISI) that results from this. In areas where inter-symbol interference is expected, it can be avoided by inserting a guard period into the timing at the beginning of each data symbol. It is then possible to copy a section from the end of the symbol to the beginning. As previously mentioned, this is known as the cyclic prefix (CP). The receiver can then sample the waveform at the optimum time and avoid any inter-symbol interference caused by reflections that are delayed by times-up to the length of the cyclic prefix, CP. The length of the cyclic prefix is important. If it is not long enough then it will not counteract the multipath reflection delay spread. If it is too long, then it will reduce the data throughput capacity. For LTE, the standard length of the cyclic prefix has been chosen to be 4.69 μs. This enables the system to accommodate path variations of up to 1.4 km. With the symbol length in LTE set to 66.7 μs. The symbol length is defined by the fact that for OFDM systems the symbol length is equal to the reciprocal of the carrier spacing so that orthogonality is achieved. With a carrier spacing of 15 kHz, this gives the symbol length of 66.7 μs (ROHLING, Hermann, 2011) (ANDREW, A., 2015).

Despite the slower symbol rate, there are much higher data rates due to the increase in carriers being modulated by an order of magnitude, from 1 (DSSS) to 48 (OFDM in 802.11a/g) and 52 (OFDM in 802.11n/ac) per 20 MHz channel. Therefore, a serial data stream is taken, and parallel data transmission is performed across the frequency domain. The sub-carriers are spaced at regular intervals called “sub-carrier frequency spacing” or offset (ΔF). The sub-carrier frequency relative to the center frequency is $k\Delta F$, where k is the sub-carrier number (Figure 22).

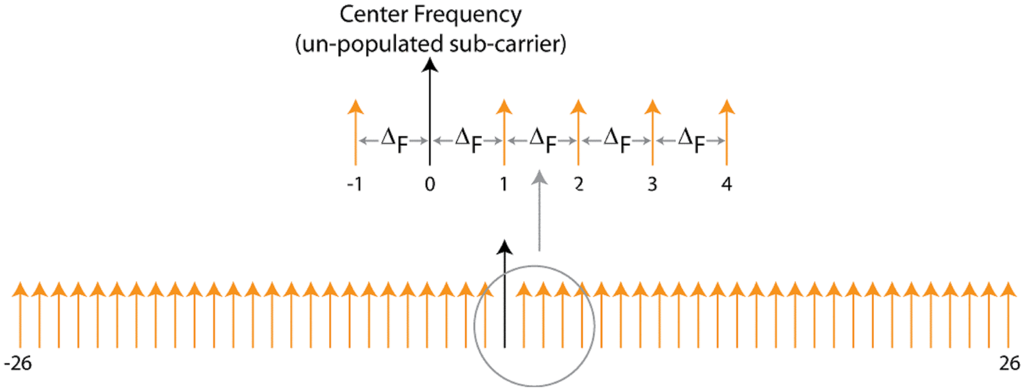


Figure 22. OFDM sub-carrier spacing (KEITHLEY INSTRUMENTS, 2008)

There are two types of frame structure in the LTE standard, Type 1 and Type 2. Type 1 uses Frequency Division Duplexing (uplink and downlink separated by frequency), and TDD uses Time Division Duplexing (uplink and downlink separated in time). FDD is the dominant frame structure used in most of the LTE deployments (3GPP, 2017).

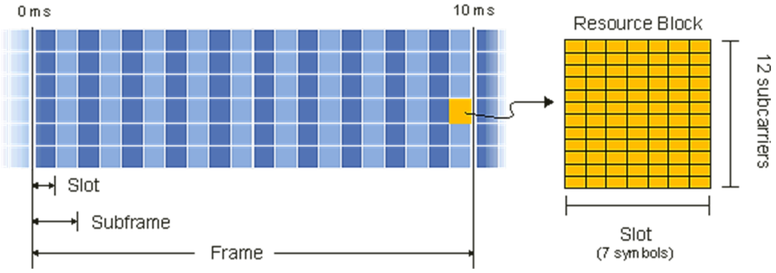


Figure 23. An FDD frame for 1.4 MHz channel and normal CP (3GPP, 2017)

According to Figure 23, a resource block (RB) is the smallest unit of resources that can be allocated to a user. The resource block is 180 kHz wide in frequency and 1 slot long in time. In frequency, resource blocks are either 12 x 15 kHz subcarriers or 24 x 7.5 kHz subcarriers wide. The number of subcarriers used per resource block for most channels and signals is 12 subcarriers. Frequency units can be expressed in number of subcarriers or resource blocks. For instance, a 5 MHz downlink signal could be described as 25 resource blocks wide or 301 subcarriers wide (DC subcarrier is not included in a resource block). The underlying data carrier for an LTE frame is the resource element (RE). The resource element, which is 1 subcarrier x 1 symbol, is the smallest discrete part of the frame and contains a single complex value representing data from a physical channel or signal (3GPP, 2017).

In FDD mode, the UL and DL frames are both 10ms long and are divided by frequency (Figure 24) or by time (Figure 25).

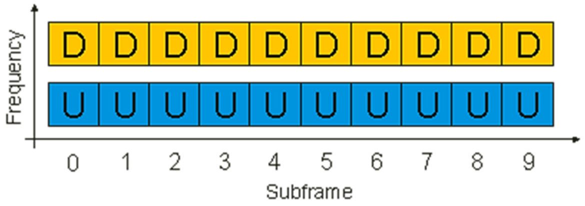


Figure 24. LTE frame Type-1 (FDD)

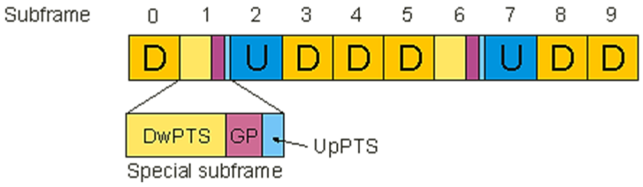


Figure 25. LTE frame Type-2 (TDD) (3GPP, 2017)

For full-duplex FDD, uplink and downlink frames are separated by frequency and are transmitted continuously and synchronously. For half-duplex FDD, the only difference is that a UE cannot receive while transmitting. The base station can specify a time offset (in PDCCH) to be applied to the uplink frame relative to the downlink frame. In TDD mode, the uplink and downlink subframes are transmitted on the same frequency and are multiplexed in the time domain. The locations of the uplink, downlink, and special subframes are determined by the uplink-downlink configuration. There are seven possible configurations given in the standard (3GPP, 2017).

The OFDM signal used in LTE comprises maximum of 2048 different sub-carriers, having a spacing of 15 kHz. Although it is mandatory for the devices to have capability to be able to receive all 2048 sub-carriers, not all need to be transmitted by the base station which only needs to be able to support the transmission of 72 sub-carriers. In this way all mobiles will be able to talk to any base station. Since the bandwidths defined by the LTE standard are 1.4, 3, 5, 10, 15, and 20 MHz, the Table 9 shows how many subcarriers and resource blocks there are in each bandwidth for uplink and downlink (3GPP, 2017).

Table 9. Frequency measures (3GPP, 2017)

Bandwidth	**Resource Blocks**	**Subcarriers (downlink)**	**Subcarriers (uplink)**
1.4 MHz	6	73	72
3 MHz	15	181	180
5 MHz	25	301	300
10 MHz	50	601	600
15 MHz	75	901	900
20 MHz	100	1201	1200

As described before, uplink user transmissions consist of uplink user data (PUSCH), random-access requests (PRACH), user control channels (PUCCH), and sounding reference signals (SRS). FDD and TDD uplink transmissions have the same physical channels and signals. The only difference is that TDD frames include a special subframe, part of which can be used for SRS and PRACH uplink transmissions (Figure 25). The following figure stands as an example for User 1 that has a PUSCH allocation of [RB 20, slots 4-5], and User 2 that has a PUCCH allocation of [subframe 2, PUCCH index 0]. User 3 has been given an SRS allocation of subcarrier 94 to 135 in subframe 2, and User 4 is transmitting in a PRACH allocation. A user cannot transmit both PUCCH and PUSCH data in the same slot (3GPP, 2017).

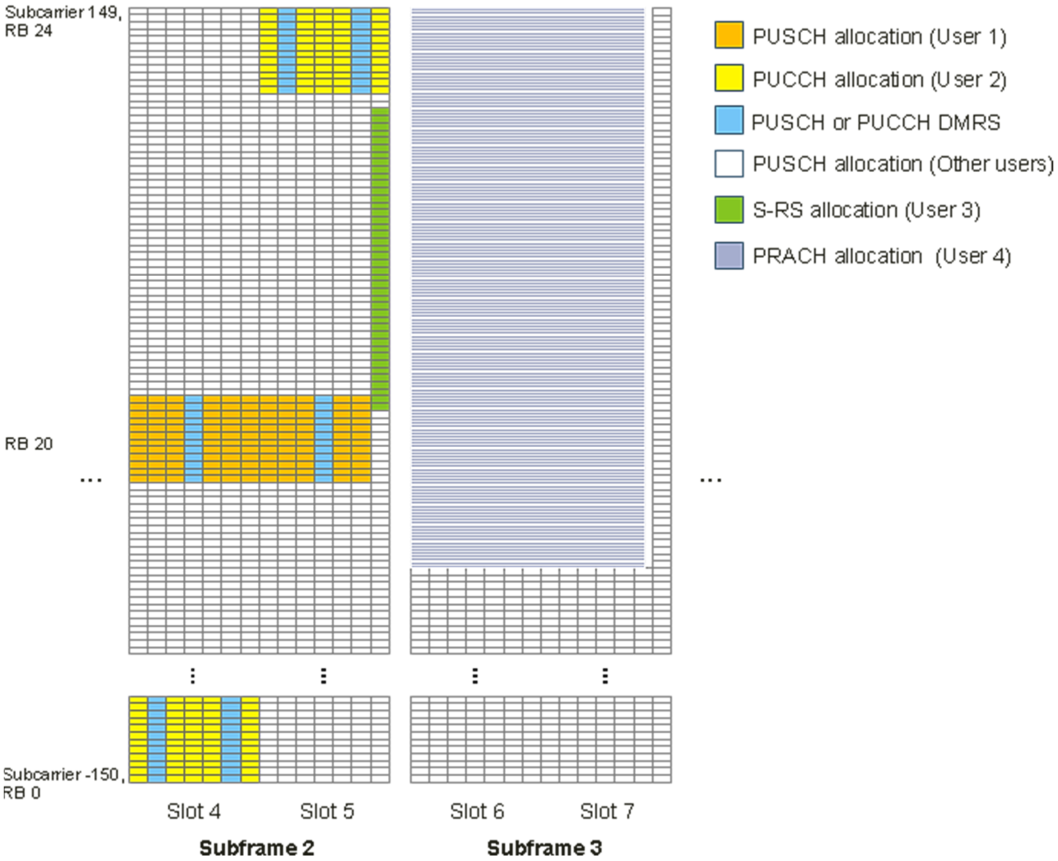


Figure 26. LTE uplink subframes 2-3; Bandwidth: 5 MHz = 300 subcarriers = 25 RB; Normal CP, PUCCH Type 2, 15 KHz subcarrier spacing (3GPP, 2017)

Within the OFDM signal it is possible to choose between three types of modulation for the LTE signal:

a) ***QPSK (= 4QAM)*** 2 bits per symbol

b) ***16QAM*** 4 bits per symbol

c) ***64QAM*** 6 bits per symbol

The exact LTE modulation format is chosen depending upon the prevailing conditions. The lower forms of modulation, (QPSK) do not require such a large signal to noise ratio but are not able to send the data as fast. Only when there is a sufficient signal to noise ratio can the higher order modulation format be used (ADRIO COMMUNICATIONS LTD., 2017).

A. Measurements based on constellations and Error Vector Magnitude (EVM) metrics

To measure the efficiency of the OFDM system, it is required to comprehend the concepts of parallel symbol transmissions in OFDM, as described previously. In Figure 27, the symbol that undergoes an inverse Fast Fourier Transform is coupled in *I* and *Q* components that form the waveforms, which is called serial symbol transmission. However, OFDM utilizes the efficiency of parallel symbol transmission with coupling multiple symbols using inverse Fast Fourier Transform, that results also in modulated *I* and *Q* waveforms. In this case, multiple carriers will transmit multiple symbols in parallel and the carriers may have modulations, such as: BPSK, QPSK, 16QAM, 64QAM etc.

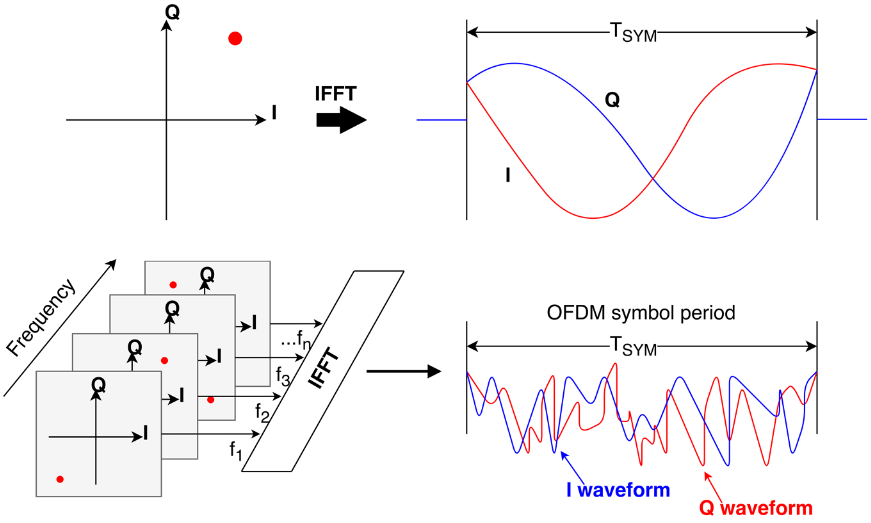


Figure 27. From symbol to waveform (serial and parallel symbol transmissions)

Quadrature Amplitude Modulation (QAM) is a form of modulation which is widely used for modulating data signals onto a carrier used for radio communications. It is widely used because it offers advantages over other forms of data modulation such as PSK, although many forms of data modulation operate alongside each other (ADRIO COMMUNICATIONS LTD., 2017).

Quadrature Amplitude Modulation, QAM is a signal in which two carriers shifted in phase by 90 degrees are modulated and the resultant output consists of both amplitude and phase variations. Since both amplitude and phase variations are present, it may also be considered as a mixture of amplitude and phase modulation. A motivation for the use of quadrature amplitude modulation comes from the fact that a straight amplitude modulated signal, i.e. double sideband even with a suppressed carrier, and occupies twice the bandwidth of the modulating signal. This is very wasteful of the available frequency spectrum. QAM restores the balance by placing two independent double sideband suppressed carrier signals in the same spectrum, as one ordinary double sideband suppressed carrier signal. Quadrature amplitude modulation, QAM, when used for digital transmission for radio communications applications can carry higher data rates than ordinary amplitude modulated schemes and phase modulated schemes. As with phase shift keying (PSK), etc., the number of points at which the signal can rest, i.e. the number of points on the constellation is indicated in the modulation format description, e.g. 16QAM uses a 16-point constellation. When using QAM, the constellation points are normally arranged in a square grid with equal vertical and horizontal spacing and as a result the most common forms of QAM use a constellation with the number of points equal to a power of 2 i.e. 4, 16, 64 etc. By using higher order modulation formats, i.e. more points on the constellation, it is possible to transmit more bits per symbol. However, the points are closer together and they are therefore more susceptible to noise and data errors. Normally a QAM constellation is square and therefore the most common forms of QAM 16QAM, 64QAM and 256QAM. The advantage of moving to the higher order formats is that there are more points within the constellation and therefore it is possible to transmit more bits per symbol. The downside is that the constellation points are closer together and therefore the link is more susceptible to noise. As a result, higher order versions of QAM are only used when there is a sufficiently high signal to noise ratio. To provide an example of how QAM operates, the constellation diagram in Figure 28 shows the values associated with the different states for a 16QAM signal. From this it can be seen that a continuous bit stream may be grouped into fours and represented as a sequence (ADRIO COMMUNICATIONS LTD., 2017).

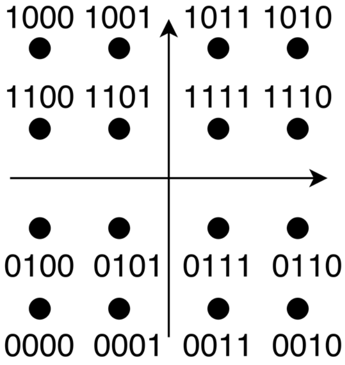


Figure 28. 16QAM modulation constellation

Although QAM appears to increase the efficiency of transmission for radio communications systems by utilizing both amplitude and phase variations, it has a number of drawbacks. The first is that it is more susceptible to noise because the states are closer together so that a lower level of noise is needed to move the signal to a different decision point. Receivers for use with phase or frequency modulation are both able to use limiting amplifiers that are able to remove any amplitude noise and thereby improve the noise reliance. This is not the case with QAM. The second limitation is also associated with the amplitude component of the signal. When a phase or frequency modulated signal is amplified in a radio transmitter, there is no need to use linear amplifiers, whereas when using QAM that contains an amplitude component, linearity must be maintained. Unfortunately, linear amplifiers are less efficient and consume more power, and this makes them less attractive for mobile applications (ADRIO COMMUNICATIONS LTD., 2017).

To measure the efficiency of the eNB base station, a constellation diagram is formed within a signal analyzer which inspects the radio access network. A constellation diagram is a representation of a digital modulation scheme in the complex plane, in the particular case a 16QAM modulation scheme. If the constellation does not look linear, it is due to excess or shortage of gain at the *I* or *Q* components of the modulated signal. That indicates the necessity to adjust the gains of the particular channel properly. In such case, the constellation offset can be observed at the signal analyzer (Figure 29).

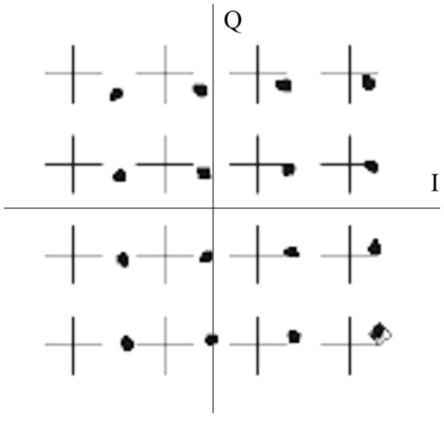


Figure 29. Origin offset example of 16-QAM constellation (KEITHLEY INSTRUMENTS, 2008)

The *I* and *Q* components of the signal are forming a correlation angle of 90˚, which when summed forms the modulated signal (Figure 30).

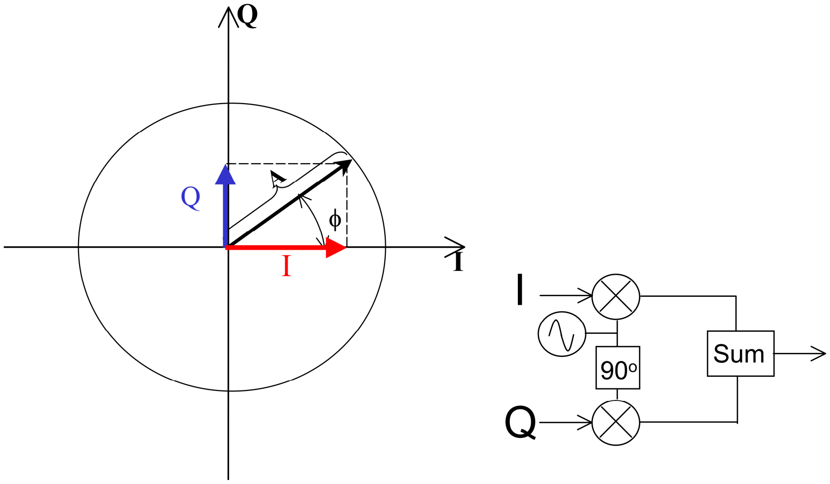


Figure 30. Digital modulation of a signal (KEITHLEY INSTRUMENTS, 2008)

The amplitude of the signal is represented as the length of A, that is $\sqrt{I^2 + Q^2}$ (Pythagorean Theorem). And the phase (angle ϕ) is $tan^{-1}(\frac{Q}{I})$. If the signal is represented on a complex plane, it would look as in Figure 31, that is $S(t) = Acos(2\pi f_C(t) + \phi$, where f_C is the signal frequency.

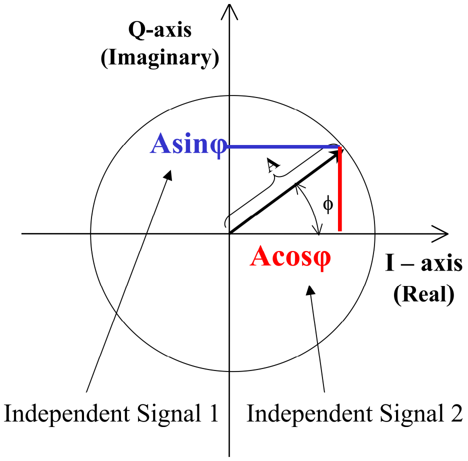


Figure 31. Representation of signal on complex plane (KEITHLEY INSTRUMENTS, 2008)

The *Error Vector Magnitude* (EVM) is a metric of performance that derives the relationships among signal-to-noise ratio (SNR) and the bit error rate (BER). Namely, *"Error Vector Magnitude (EVM) is a performance metric for assessing the quality of communication. EVM expresses the difference between the expected complex voltage of a demodulated symbol and the value of the actual received symbol"* (SHAFIK, R. A. et al., 2006).

Bit Error Rate (BER) is a used performance metric which describes the probability of error in terms of number of 28 mistaken bits per bit transmitted. BER is a direct effect of channel noise for Gaussian noise channel models. For fading channels, BER performance of any communication system is worse and can be directly related to that of the Gaussian noise channel performance. Considering M-ary modulation with coherent detection in Gaussian noise channel and perfect recovery of the carrier frequency and phase, it can be shown that (SHAFIK, R. A. et al., 2006):

$$P_b = \frac{2(1-\frac{1}{L})}{\log_2 L} Q\left[\sqrt{\left[\frac{3\log_2 L}{L^2-1}\right]\frac{2E_b}{N_0}}\right]$$

, where L is the number of levels in each dimension of the M-ary modulation system, E_b is the energy per bit and $\frac{N_0}{2}$ is the noise power spectral density. Q is the Gaussian co-error function and is given by:

$$Q(x) = \int_x^{\alpha} \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} dy$$

Assuming raised cosine pulses with sampling at data rate, the error rate in terms of signal to noise ratio would then be:

$$P_b = \frac{2(1-\frac{1}{L})}{\log_2 L} Q\left[\sqrt{\left[\frac{3\log_2 L}{L^2-1}\right]\frac{2E_S}{N_0 \log_2 M}}\right]$$

, where $\frac{E_S}{N_0}$ is the signal-to-noise ratio for the M-ary modulation system and raised cosine, pulse shaping at data rate. Therefore, the BER performance in terms of SNR is defined and used as a main tool for many adaptive systems. Consequently, the EVM (Error Vector Magnitude) measurements are performed on the vector signal analyzers, real-time analyzers or other instruments that capture a time record and internally perform a FFT to enable frequency domain analysis. Signals are down-converted before EVM calculations are made. Since different modulation systems such as: BPSK, 4-QAM, 16-QAM etc., have different amplitude levels, to calculate and compare EVM measurements effectively some normalization is typically carried out. The normalization is derived such that the mean square amplitude of all possible symbols in the constellation of any modulation scheme equals one. Thus, EVM is defined as the root-mean-square (RMS) value of the difference between a collection of measured symbols and ideal symbols. These differences are averaged over a given, typically large number of symbols and are often shown as a percent of the average power per symbols of the constellation. Therefore, EVM can be given as:

$$EVM_{RMS} = \frac{\frac{1}{N}\sum_{n=1}^{N}\left|S_n - S_{0,n}\right|^2}{\frac{1}{N}\sum_{n=1}^{N}\left|S_{0,n}\right|^2}$$

, where S_n is the normalized nth symbol in the stream of measured symbols, $S_{0,n}$ is the ideal normalized constellation point of the nth symbol and N is the number of unique symbols in the constellation. The expression cannot be replaced by their unnormalized value, since the normalization constant for the measured constellation and the ideal constellation are not the same. The normalization scaling factor for ideal symbol is represented by:

$$|A| = \sqrt{\frac{1}{\frac{P_v}{T}}} = \sqrt{\frac{T}{P_v}}$$

, where P_v is the total power of the measured constellation of T symbols. For RMS voltage levels of inphase and quadrature components, V_I and V_Q and for T>>N, it can be shown that P_v is expressed as:

$$P_v = \sum_{t=1}^{T}\left[\left(V_{I,t}\right)^2 + \left(V_{Q,t}\right)^2\right](W)$$

The normalization factor for ideal case can be directly measured from *N* unique ideal constellation points as:

$$|A_0| = \sqrt{\frac{N}{\sum_{n=1}^{N}\left[\left(V_{I,t}\right)^2 + \left(V_{Q,t}\right)^2\right]}}$$

Accordingly, the EVM per root-mean-square can be extended by:

$$EVM_{RMS} = \left[\frac{\frac{1}{T}\sum_{t=1}^{T}\left|I_t - I_{0,t}\right|^2 + \left|Q_t - Q_{0,t}\right|^2}{\frac{1}{T}\sum_{t=1}^{T}\left|I_{0,t}\right|^2 + \left|Q_{0,t}\right|^2}\right]$$

, where $I_t = \left(V_{I_t}\right)|A|$ is the normalized in-phase voltage for measured symbols and $I_{0,t}\left(V_{I_0,t}\right)|A_0|$ is the normalized in-phase voltage for ideal symbols in the constellation, $Q_t = \left(V_{Q_t}\right)|A|$ is the normalized quadrature voltage for measured symbols and $Q_{0,t}\left(V_{Q_0,t}\right)|A_0|$ is the normalized quadrature voltage for ideal symbols in the constellation. This definition is used as a standard definition for the EVM according to the IEEE 802.11a – 1999 (SHAFIK, R. A. et al., 2006).

To represent the EVM into percentage or dB, it is converted accordingly:

$$EVM_{(\%)} = \sqrt{\frac{P_{error}}{P_{reference}}} \cdot 100\%$$

And

$$EVM_{(dB)} = 10\log_{10}\left(\frac{P_{error}}{P_{reference}}\right)$$

, where *P* is the RMS power.

The measurements taken from a signal analyzer would represent the EVM as a ratio of measured amplitude to intended amplitude in percentage (Figure 32), denoted by the red line. The blue line indicates the measured signal and the black line is the intended signal. The angle φ the black and blue lines form is the phase error, or IQ Error Phase. If the portion of this image is imagined to be one quadrant of the *(x,y)* axis at a signal analyzer constellation, then the unit circle is depicted by the purple dashed line. At this point, the distance that we obtain from the dashed line and the red dot is the actual magnitude error, or more specifically IQ Error Magnitude.

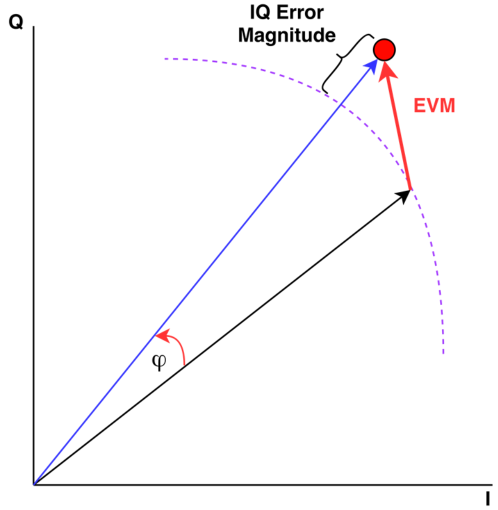


Figure 32. EVM ratio of measured amplitude to intended amplitude

When error occurs, the signal analyzer would then simply indicate the constellation imbalance as in Figure 33 and Figure 34.

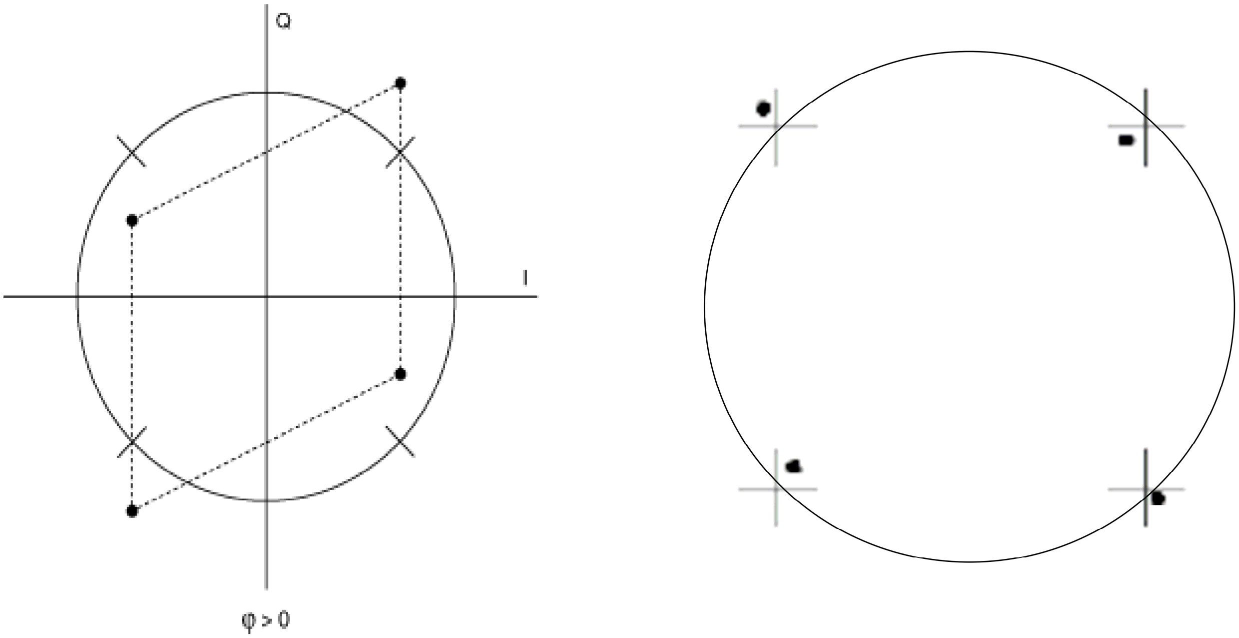


Figure 33. Quadrature error examples - QPSK constellations (KEITHLEY INSTRUMENTS, 2008)

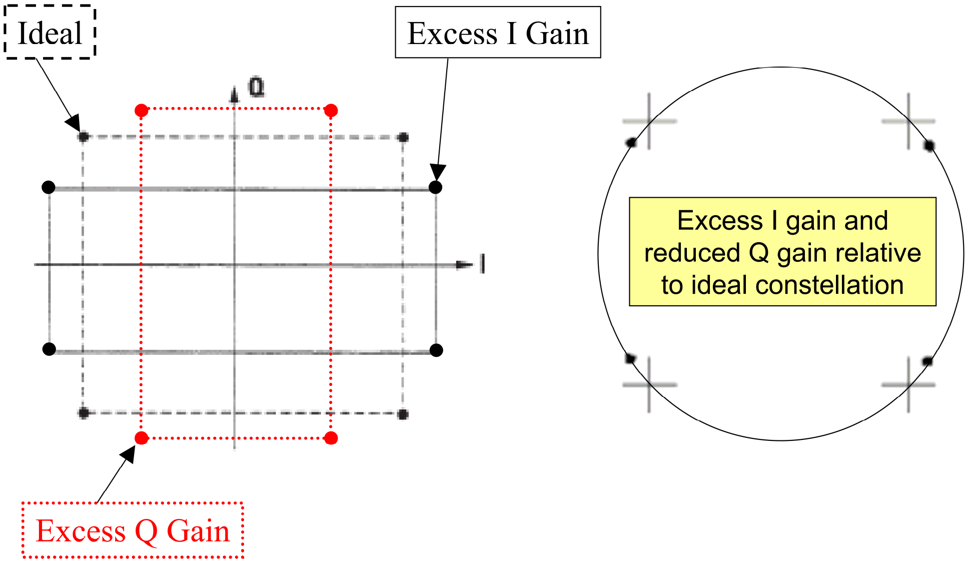

Ideal
Excess I Gain
Q
I
Excess I gain and reduced Q gain relative to ideal constellation
Excess Q Gain

Figure 34. Modular imbalances examples - QPSK constellations (KEITHLEY INSTRUMENTS, 2008)

If the signal is subdued to the effects of gain imbalances, then the constellation imperfections are clearly indicated at the signal analyzer plot. For example, since Quadrature Amplitude Modulation is the widely used modulation scheme in this work, the power amplifier nonlinearity can contribute to EVM as shown in Figure 35.

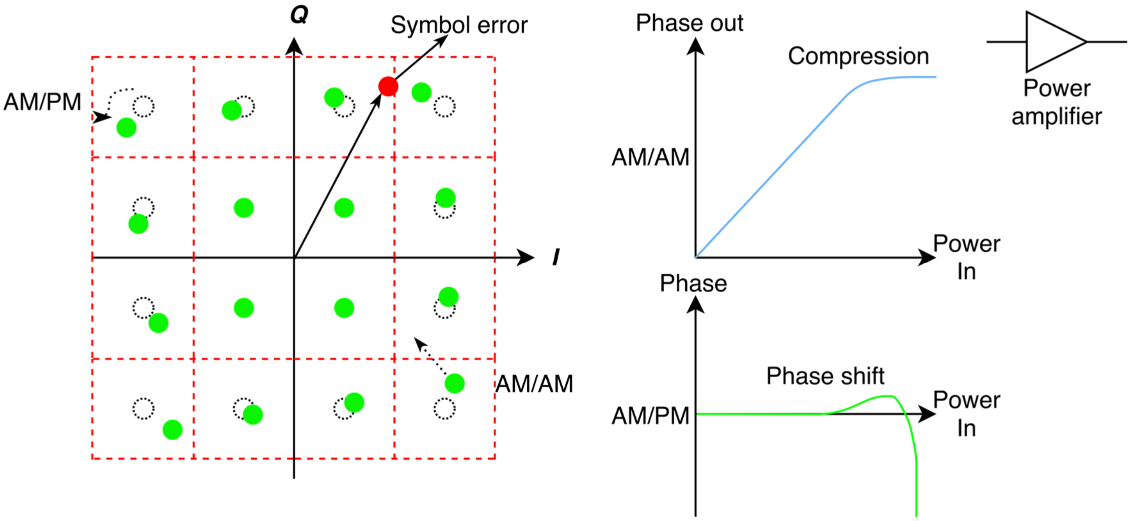


Figure 35. EVM due to power amplifier nonlinearity

Another factor is the Inter Symbol Interference (ISI), which can contribute to have symbols received at delayed intervals, and thus the constellation would appear as in Figure 36.

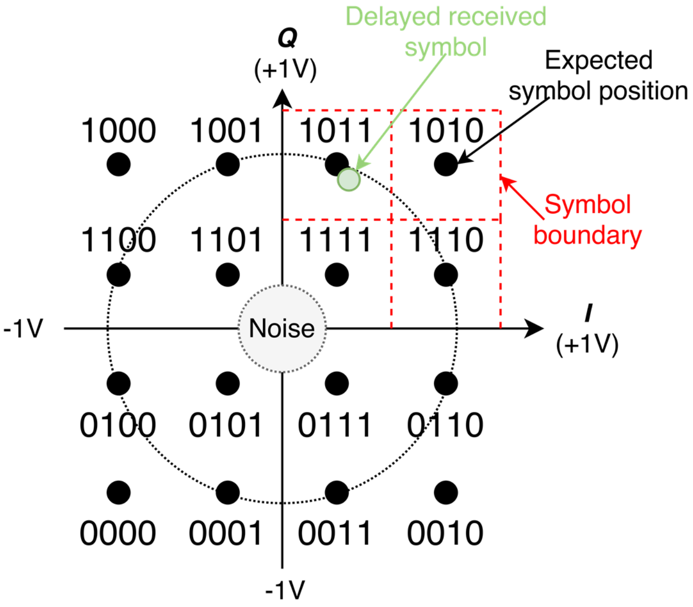


Figure 36. Inter Symbol Interference in case of 16-QAM constellations

Finally, the constellation display at the signal analyzer is an actual composite of all OFDM sub-carrier symbols, for particular frequency (Figure 37) and at a particular time (Figure 38), accordingly.

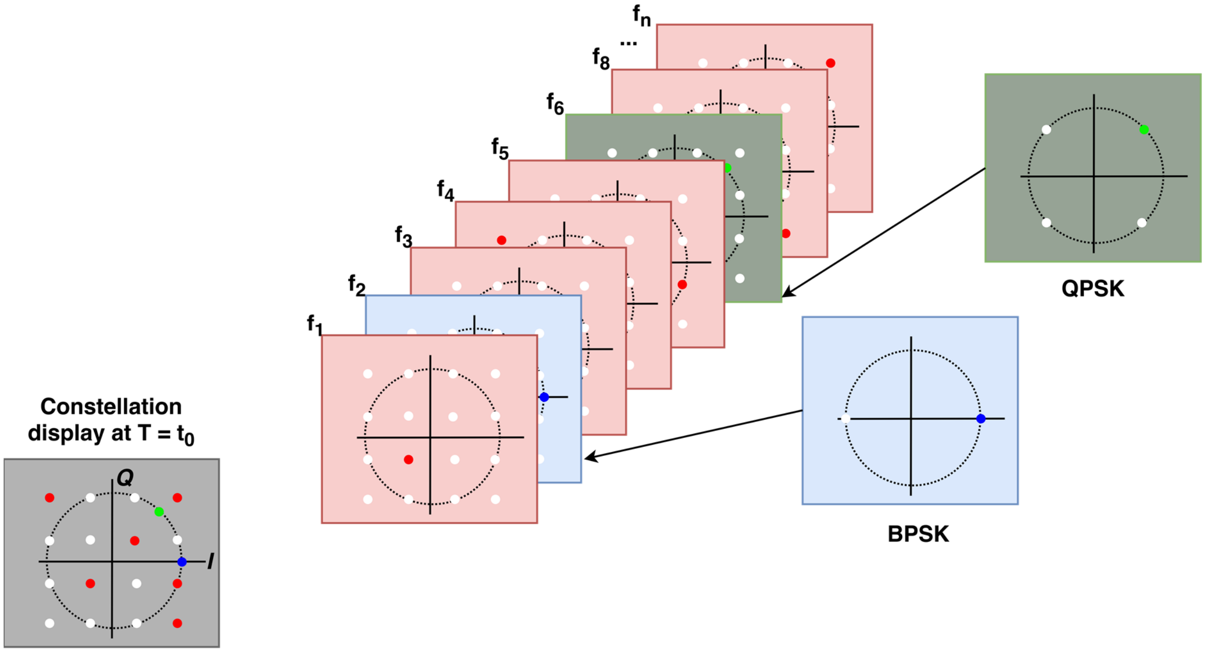


Figure 37. Constellation display - a composite of all OFDM sub-carrier symbols with a particular frequency *f*

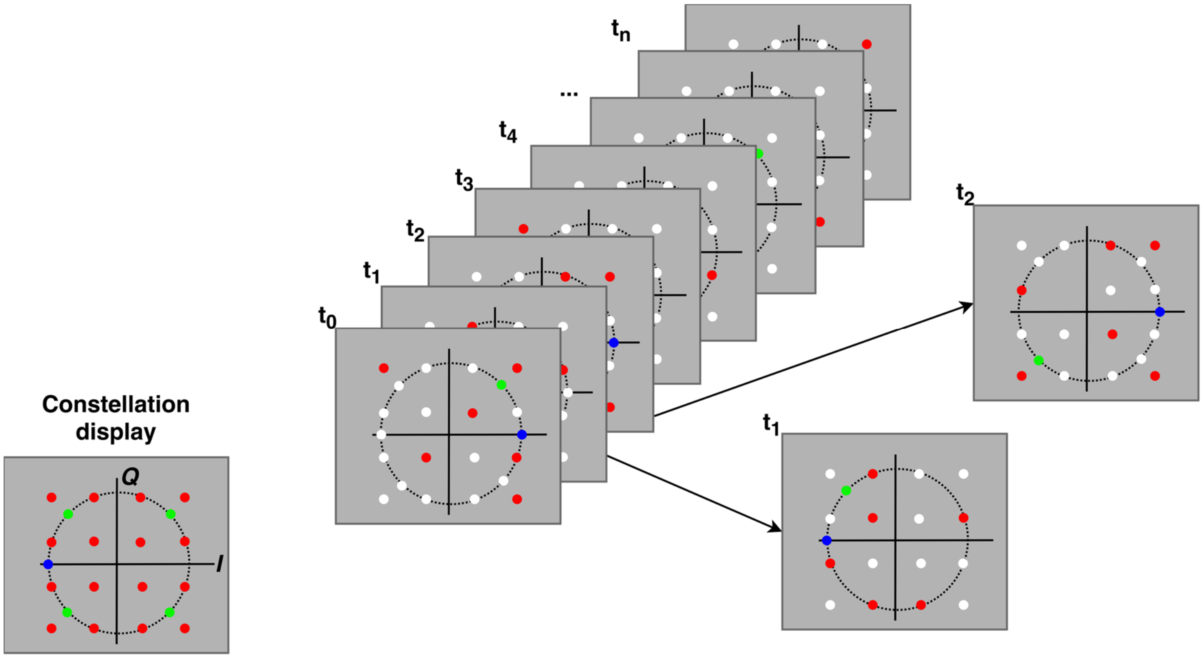


Figure 38. Constellation display - a composite of all OFDM sub-carrier symbols at a particular time *t*

OFDM as a modulation technique is not a multi-user, because all sub-carriers in a channel are used to facilitate a single link. To expand the functionality to multiple users, OFDMA (Orthogonal Frequency Division Multiple Access) assigns different number of sub-carriers to different users in a similar fashion as in CDMA. The parallel multi-symbol transmission described in Figure 27 is assigned logical number per sub-channel. The transmission on both DL and UL channels is performed in bursts, which defines a single OFDMA symbol number. As shown in Figure 39 and Figure 40, the physical sub-channels are changed per each symbol, using a PN sequence (3GPP, 2017).

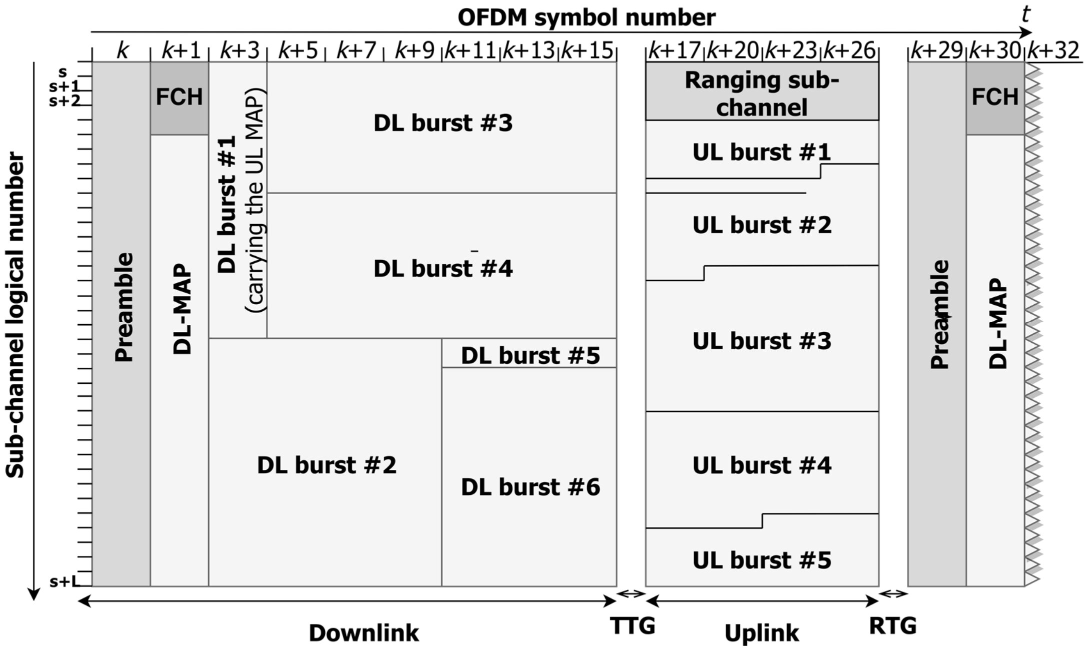


Figure 39. Dynamic symbol mapping in OFDMA

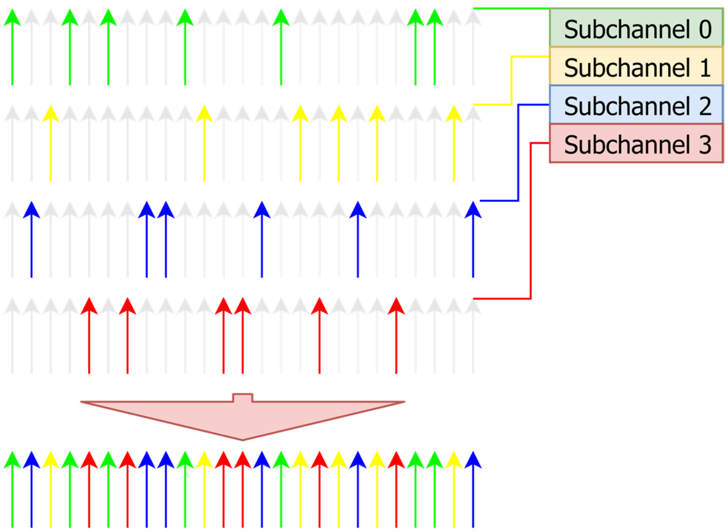


Figure 40. PN sequence for each physical sub-channel

2.1.5. Single-Carrier Frequency Division Multiple Access (SC-FDMA) for uplink channel

Similar to OFDMA, SC-FDMA divides the transmission bandwidth into multiple parallel sub-carriers maintaining the orthogonality of the subcarriers by the addition of the cyclic prefix (CP) as a guard interval. However, in SC-FDMA the data symbols are not directly assigned to each subcarrier independently like in OFDMA. Instead, the signal which is assigned to each subcarrier is a linear combination of all modulated data symbols transmitted at the same time instantaneously. For the LTE uplink, a different concept is used for the access technique. Although still using a form of OFDMA technology, the implementation is called Single Carrier Frequency Division Multiple Access (SC-FDMA). One of the key parameters that affects all mobiles is that of battery life. Even though battery performance is improving all the time, it is still necessary to ensure that the mobiles use as little battery power as possible. With the RF power amplifier that transmits the radio frequency signal via the antenna to the base station being the highest power item within the mobile, it is necessary that it operates in as efficient mode as possible. This can be significantly affected by the form of radio frequency modulation and signal format. Signals that have a high peak to average ratio and require linear amplification do not lend themselves to the use of efficient RF power amplifiers. As a result it is necessary to employ a mode of transmission that has as near a constant power level when operating. Unfortunately, OFDM has a high peak-to-average ratio. While this is not a problem for the base station where power is not a particular problem, it is unacceptable for the mobile. As a result, LTE uses a modulation scheme known as SC-FDMA - Single Carrier Frequency Division Multiplex which is a hybrid format. This combines the low peak to average ratio offered by single-carrier systems with the multipath interference resilience and flexible subcarrier frequency allocation that OFDM provides (ADRIO COMMUNICATIONS LTD., 2017). The difference between OFDMA and SC-FDMA is depicted in Figure 41.

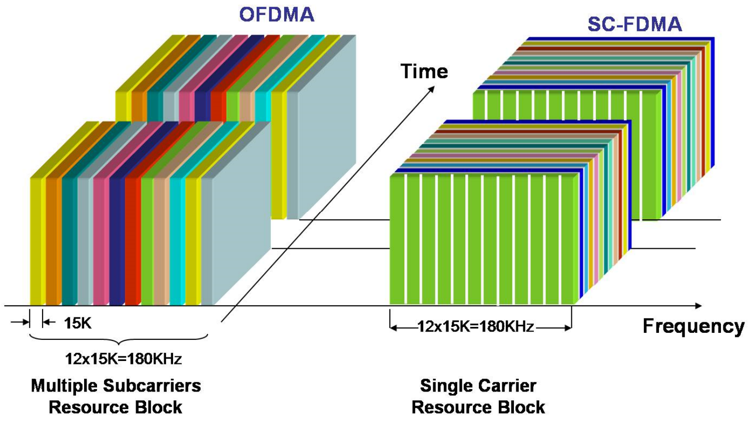


Figure 41. LTE uses SC-FDMA at the uplink (UL)

2.1.6. Multiple-antenna techniques

The wireless technologies, including LTE, utilize the cutting-edge radio antenna technologies in order to achieve maximal throughput, better spectral efficiency and accommodate much higher number of users. Besides the technology advances, there are various antenna techniques used for increasing the efficiency of the radio system. Multiple antennas can be used to achieve a multiplexing gain, a diversity gain, or an antenna gain, thus enhancing the bit rate, the error performance, or the signal-to-noise-plus-interference ratio of wireless systems, correspondingly. The field of multiple-antenna systems, often called multiple-input multiple-output (MIMO) systems, is a major subject of research and is evolving rapidly. For an optimal level of quality of service, not only high bit rates are needed, but also a good error performance (MOLISCH, A. F., 2011). However, the disruptive characteristics of wireless channels, mainly caused by multipath signal propagation (due to reflections and diffraction) and fading effects, make it challenging to accomplish both of these goals at the same time. Particularly, given a fixed bandwidth, there is always an essential compromise between bandwidth efficiency (high bit rates) and power efficiency (small error rates). Conventional single-antenna transmission techniques aiming at an optimal wireless system performance, operate in the time domain and/or in the frequency domain. Specifically, channel coding and modulation (i.e. OFDM) are ordinarily used to permeate the negative effects of multipath fading. However, regarding the ever-growing demands of wireless services, the antenna technologies are advancing very fast. In fact, when using multiple antennas, the previously vacant spatial domain can be exploited. The immense potential of using multiple antennas for wireless communications has only become clear during the last decade. At the end of the 1990s, multiple-antenna techniques were shown to provide an innovative method for achieving both higher bit rates and smaller error rates. In addition to this, multiple antennas can also be utilized in order to alleviate co-channel interference, which is additional major source of disruption in all wireless communication systems. Altogether, multiple-antenna techniques form a key technology for modern wireless communications (MIETZNER, J. et al., 2009).

To accommodate the exponentially-higher expected number of connected devices after 2020, the 5G radio systems have massive advancements in antenna technologies, as well as techniques used to back their operation. Since there is no particular definition about the structure of a 5G wireless communication system at this point, it can be acknowledged that an evolution of the present radio technologies is taking place, of which, *Massive MIMO* (ARAÚJO, D. C. et al., 2016, pp.1938-1946) and *Millimeter wave* (TOKGOZ, K. K. et al., 2018, pp.168-170) technologies are considered to be the key radio progressions for 5G wireless communications. Traditionally, the antenna in mobile communication systems is a passive element and is separated from the RF transceivers. For massive MIMO, at either lower microwave band or millimeter wave band, the active antenna will be seamlessly integrated with RF transceivers and even with RoF or ADC (DAC) and E/O (O/E). Therefore, the antenna for 5G wireless communications will have distinct characteristics compared to traditional antennas (WEI, H. et al., 2014).

A. Smart antennas

The initial usage of smart antennas dates back in the beginning of the 20th century, which was mostly directed towards military appliances and intelligence gathering. The first commercial applications of smart antennas start with the rapid growth of cellular technologies in the 1980s. Since then, the remaining wireless technologies are acting as an active driver for the development of the smart antennas in the 1990s, namely satellite broadcasting systems, indoor wireless networks, fixed and mobile wireless systems etc. The resulting efforts thus have produced techniques such as MIMO (Multiple-Input Multiple-Output) as well as implementation of access techniques (as OFDMA or SDMA), where multiple antennas are used for enhancing the spectrum and accommodating larger number of users. A smart antenna is comprised of an *antenna array,* combined with signal processing in both space and time. Spatial processing enables multiple degrees of freedom in the system design, which can increase the global performance of the system. The concept of antenna arrays is not new and is widely implemented in radar and aerospace technology through the last century (CHRYSSOMALLIS, M., 2000, pp.129-136).

The smart antenna works in the manner of propagation of each path differently for each different antenna element. This allows collection of elements to distinguish individual paths to within a certain resolution. Therefore, smart antenna transmitters can encode different streams of data onto different paths or linear combination of paths, which increases the data rate and provides diversity gain. Also, this procedure automates the placement of the antenna in that way that the smart antenna adapts electronically to the environment (ANKIT, D. P., 2013). There are three antenna groups:

- *Phased antenna array systems* - A phased array antenna is comprised of numerous radiating elements, each containing a phase shifter. Beams are formed by shifting the phase of the signal emitted from each radiating element, in order to provide constructive/destructive interference, as well as to steer the beams in the anticipated direction. This type of antenna array system is widely used in radar technologies.
- *Switched beam systems* – Switched-beam antennas have several fixed beam patterns. This approach is not very flexible, but its simplicity allows unsophisticated deployments. Switched-beam smart antenna systems are shown to either increase the capacity or extend the radio coverage by increasing the carrier-to-interference ratio (CIR), consisting of a multiple narrow-beam directional antenna along with a beam-selection algorithm. Switched-beam smart antennas offer a potentially more desirable solution than adaptive antenna arrays since they are based on well-known technology, require no complicated beam-forming (combining) network, and require no significant changes to the existing cellular infrastructure. Switched-beam antennas are based on the retro-targeting concept (STÜBER, G. L., 1996). The choice of the triggered receive beam is constructed on the received signal-strength indicator (RSSI) [and also the supervisory audio tone (SAT) for the advanced mobile phone systems (AMPS)]. Forward-channel transmissions (BS's–MS's) are over the best received beam, i.e., the same beam is used for both reception and transmission. Beam forming is accomplished by using physically directive antenna elements to create aperture and, thus, gain (HO, M-J. et al., 1998, pp.10-19)
- *Adaptive antenna array systems* - These type of antenna systems allow the beam to be continually steered (directed) to any direction, in order to allow for the maximum signal to be received and the interference minimized. Adaptive antenna arrays have been successfully used in TDMA mobile wireless systems to mitigate rapid dispersive fading, suppress cochannel interference, and, therefore, improve communication capacity. For systems with flat fading, the direct matrix

inversion (DMI), or the diagonal loading DMI (DMI/DL) algorithm for antenna diversity can be used to enhance desired signal reception and suppress interference effectively. The DMI/DL algorithm, can be also used for spatial-temporal equalization in TDMA systems to suppress both inter-symbol and cochannel interference. The use of adaptive antenna arrays in the OFDM systems suppresses cochannel interference. The difficulty of adaptive antenna arrays for OFDM systems stems from the fast change of parameters for the MMSE-DC because OFDM systems have much longer symbol duration than that of single carrier or TDMA systems. Hence, the parameter estimation approaches for TDMA systems are not applicable to OFDM systems (LI, Y. and Sollenberger, N. R., 1999, pp.217-229).

B. Adaptive Beamforming

Beamforming, also known as *spatial filtering*, is a signal processing technique used in sensor arrays for directional signal transmission or reception (GOLBON-HAGHIGHI, M. H., 2016, pp.163-199). This is realized by coalescing components in an antenna array in such a way that signals at specific angles experience constructive interference while others experience destructive interference. Beamforming can be used at both the transmitting and receiving sides in order to achieve spatial selectivity. The improvement compared with omnidirectional reception/transmission is known as the directivity of the array (FORENZA, A. et al., 2005, pp.3188-3192). In other words, using beamforming technique, it is possible to direct the radiation towards the user device in order to achieve better connectivity and lower transmission error rate (Figure 42).

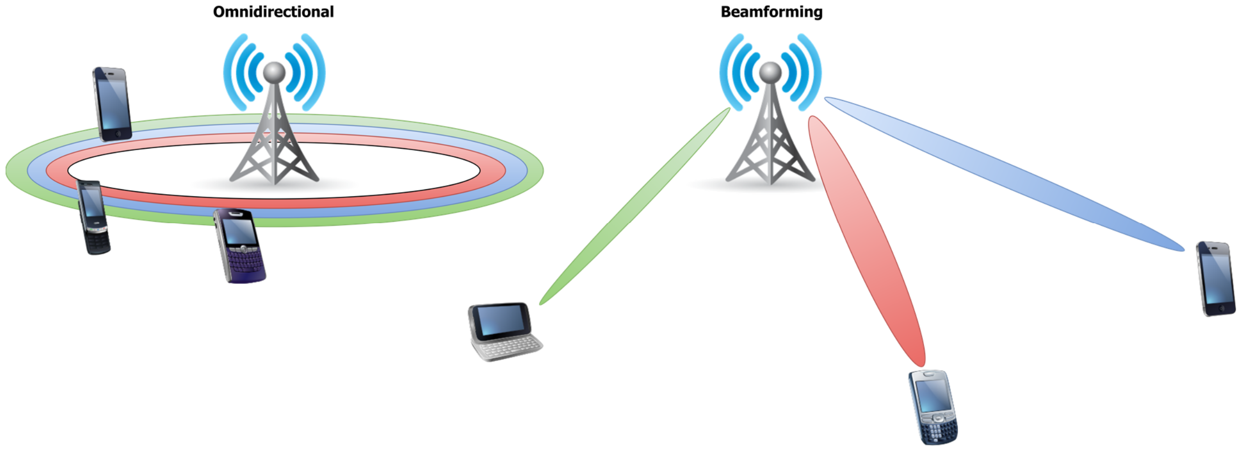


Figure 42. Omnidirectional radiation pattern and Beamforming

Smart antennas are widely used for wireless communications due to their ability to increase the coverage and capacity of communication systems. Utilization of an adaptive algorithm is one of the core technologies of smart antenna. Adaptive beamforming can receive signal from a certain direction by adjusting the array weight vector to enhance the desired signal and suppress the interference and noise (YANG, Y. Z. a. X., 2016, pp.522-525). Adaptive beamforming is a key technology of smart antenna; the core is to obtain the optimum weights of the antenna array by some adaptive beamforming algorithms, and finally adjust the main lobe to focus on the arriving direction of the desired signal, as well as suppress the interfering signal. By these ways, the antenna can receive the interesting signal efficiently. In practical application, the speed of convergence, complexity, and robustness are the main factors to be considered when choosing an adaptive beamforming algorithm (YONG-JIANG, S. et al., 2012, pp.1-3).

C. Antenna Diversity (Spatial Diversity)

The base station is the most complex element in any wireless system, because it is responsible for various processing, including the diversity schemes, combining, modulation, coding, error correction etc. These functions instigate high power consumption. In the subsequent chapters, it will be disclosed that the virtualization of the base station function and the radio access network, entail high-power computing devices to placate the immense demand for computing resources. In addition to diversity processing algorithms, there are numerous antenna diversity techniques that can be applied for accomplishing improvement and augmenting the consistency of a wireless link. Practically, in urban and indoor settings, no clear line-of-sight (LOS) between the transmitter and the receiver exists (DOBKIN, D. M., 2011). Instead, the signal is reflected by multiple paths before being received. Therefore, the multi-path effects can induce phase shifts, time delays, attenuations, and distortions that can detrimentally interfere between each other at the aperture of the receiving antenna. Correspondingly to the increased demand for processing power, an antenna diversity technique needs additional hardware integration to accommodate the peculiarity of such scenario.

Antenna diversity is particularly effective at extenuating multipath conditions. This is due to multiple antennas offering a receiver several observations of a single signal. Each antenna will confront a diverse interference scenery. Thus, if one antenna is facing a deep fade, it is probable that another antenna has an adequate signal. Collectively, a system of such scopes can deliver a robust link. While this is principally observed in receiving systems (diversity reception) (TATARINSKIY, S. N. et al., 2006, pp.1014-1014), the equivalent is also demonstrated valuable for transmitting systems (transmit diversity) (LOZANO, A. and Jindal, N., 2010, pp.186-197) as well.

For example, to alleviate the effects of multipath fading, multiple-antenna diversity systems engage multiple antennas and a digital central receiver using diversity combiner. The multiple-antenna adherences are handled at distinct receivers and sent to a central receiver. The central receiver combines all the individual receiver information to form universal information on which symbol was transmitted. A multiple-antenna diversity scheme can be represented by course-resolution information or high-resolution information. The exact diversity scheme that functions by course-resolution information is designated for the case of non-coherent frequency-shift keying in slow Rayleigh fading and additive Gaussian noise. This technique is more cost effective and can be used instead of high-resolution scheme without noticeable loss in performance which simplifies receiver design and construction (AZIZ, A. M., 2009, pp.1-10).

Antenna diversity can be achieved in multiple ways, in accordance to the situation and the environment; also, the expected interference, which can direct designers to implement one method or combine several for signal quality improvement:

Spatial diversity – Implementation with multiple physically-separated antennas with same properties. In some situations, a space of separation on the order of a wavelength is sufficient, but sometimes the antennas need to be distanced more from each other. Sectorization of a cell in the mobile network is an example of a separation of antennas kilometers apart, that is a mechanism for combatting co-channel interference and spectrum reuse (HANLEN, L. and Fu, M., 2006, pp.133-142).

Pattern diversity – Colocation of multiple antennas with different radiation patterns. This type of diversity includes usually directional antennas that are separated by short distances. The benefit in this case is that the directional antennas can achieve higher gain than omnidirectional antennas (YANG, S. L. S. et al., 2008, pp.71-79).

Polarization diversity – Combining pairs of antennas with different polarization radiating patterns (horizontal or vertical, left-hand or right-hand circular polarization) (KADIR, M. F. A. et al., 2008, pp.128-131).

Transmit/Receive diversity – Two separate collocated antennas for transmission and receiving. This configuration omits the necessity for duplexer and protect sensitive receiver components from the high power used to transmit (DIGHE, P. A. et al., 2003, pp.694-703).

Adaptive arrays – A situation where a single antenna with active elements can be used, which can change the radiation pattern in accordance to the requirements of the environment conditions. Phase shifters and attenuators are used within active electronically scanned arrays (AESAs), to provide an instantaneous scan ability as well as radiation pattern and polarization control. An example is a radar antenna, which can switch to different modes of operation (searching, tracking, mapping or jamming countermeasures) (SRAR, J. A. et al., 2010).

D. Spatial multiplexing (SMX)

Spatial multiplexing, as the name indicates, is a transmission technique used in MIMO (Multiple-Input-Multiple-Output) wireless systems for transmission of impartial and separately encoded data signals, so-called *streams*, from each of the multiple transmit antennas. Therefore, the space dimension is reused (multiplexed), multiple times. To take advantage of the additional throughput capability, MIMO utilizes several sets of antennas. In many MIMO systems, just two are used, but there is no reason why further antennas cannot be employed and this increases the throughput. In any case for MIMO spatial multiplexing, the number of receive antennas must be equal to or greater than the number of transmit antennas (LOZANO, A. and Jindal, N., 2010, pp.186-197).

E. Space-Division Multiple Access (SDMA)

SDMA is an access technique, similar to OFDM, with the difference that SDMA creates parallel spatial pipes next to higher capacity pipes through spatial multiplexing and/or diversity. This can enable superior performance in radio systems that require multiple-access. Combined with techniques such as beamforming, the SDMA can allow the base station to save power and avoid wasting energy on transmissions when there are no reachable mobile units, which can also minimize interference. Instead of receiving signals coming from all directions including noise and interference signals, the receiving antenna can collect adapted signal from the smart transmitting antennas at the base station; which using phased array technologies, radiates an adapted pattern according to the requirements for the corresponding user devices. The radiation pattern of the base station is adapted to each UE to obtain highest gain in the direction of the given user with utilization of phased array techniques. In GSM cellular networks, the base station can acquire the distance (but not direction) of a mobile phone by use of a technique called "timing advance" (TA) (HUNT, A. et al., 2016, pp.643-647). The base transceiver station (BTS) can determine how far the mobile station (MS) is by interpreting the reported TA. This information, along with other parameters, can then be used to power down the BTS or MS, if a power control feature is implemented in the network. The power control in either BTS or MS is implemented in most modern networks, especially on the MS, as this ensures a better battery life for the MS. This is also why having a BTS close to the user results in less exposure to electromagnetic radiation. This is why one may be safer to have a BTS close to them as their MS will be powered down as much as possible. For example, there is more power being transmitted from the MS than what one would receive from the BTS even if they were 6 meters away from a BTS mast. However, this estimation might not consider all the Mobile stations that a particular BTS is supporting with EM radiation at any given time (HARTMANN, C., 2017). In the same manner, 5th generation mobile networks will be focused in utilizing the given position of the MS in relation to BTS in order to focus all MS Radio frequency power to the BTS direction and vice versa, thus enabling power savings for the Mobile Operator, reducing MS SAR index, reducing the EM field around base stations since beam forming will concentrate RF power when it will be used rather than spread uniformly around the BTS, reducing health and safety concerns, enhancing spectral efficiency, and decreased MS battery consumption (TELECOMPAPER, 2013).

F. MIMO (Multiple-Input Multiple-Output)

One of the best advantages of LTE systems is that they employ the power of multiple-antenna transmission. As stated previously, the combining of antennas can not only increase the throughput of the system but also minimize transmission error as well as cope with the multipath problem. There are two different scenarios where MIMO is used, specifically: Single-User MIMO (SU-MIMO) or Multi-User MIMO (MU-MIMO), although a common set of concepts captures the essential MIMO benefits in both cases. Single-User MIMO was established in the first version of LTE, whereas the Multi-User MIMO starts with the deployment of LTE Releases 9 and 10 (SÄLZER, T. et al., 2011, pp.249-277), and is the concept of research focus in this thesis.

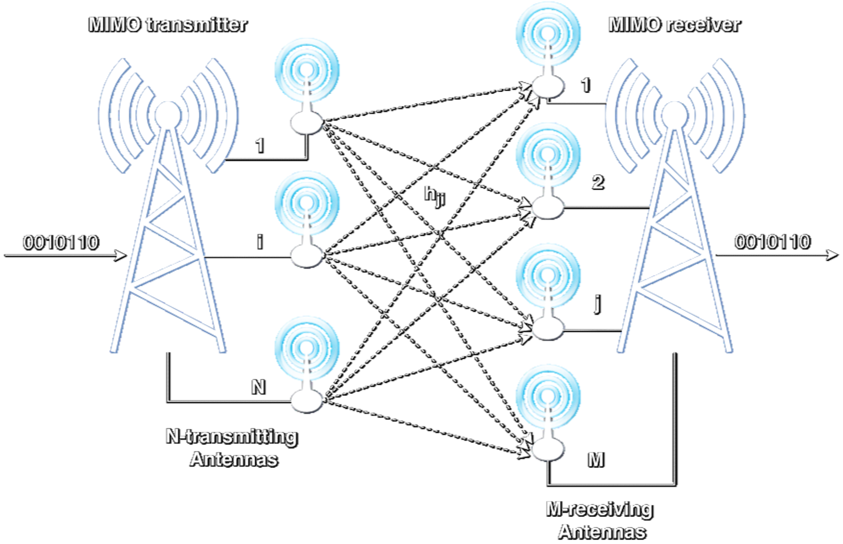


Figure 43. A MIMO system with N-transmit and M-receive antennas, giving an MxN channel matrix with MN links

While traditional wireless communications such as Single-Input Single-Output (SISO) exploit time or frequency domain pre-processing and decoding of the transmitted and received data respectively, the use of additional antenna elements at either the base station (eNB) or UE side, opens an extra spatial dimension to signal precoding and detection. Space-time processing methods exploit this dimension with the aim of improving the link's performance in terms of one or more possible metrics, such as the error rate, communication data rate, coverage area and spectral efficiency. Depending on the availability of multiple antennas at the transmitter and/or the receiver, such techniques are classified as Single-Input Multiple-Output (SIMO), Multiple-Input Single-Output (MISO) or MIMO. Thus, in the scenario of multi-antenna-enabled base station communicating with a single antenna UIE, the uplink and downlink are referred to as SIMO and MISO respectively. When a multiple-antenna terminal is involved, a full MIMO link may be obtained, although the term MIMO is sometimes also used in its widest sense, thus including SIMO and MISO as special cases. While a point-to-point multiple-antenna link between a base station and a UE is referred to as Single-User MIMO (SU-MIMO), Multi-User MIMO (MU-MIMO) features several UEs communicating simultaneously with a common base station using the same frequency and time domain resources. By extension, considering a multicellular context, neighboring base stations sharing their antennas in virtual MIMO fashion to communicate with the same set of UEs in different cells comes under the term Coordinated MultiPoint (CoMP) transmission/reception. This latter scenario is not supported in the first versions of LTE but is included in LTE-Advanced (SÄLZER, T. et al., 2011, pp.249-277).

Massive-MIMO for 5G

Massive MIMO is the currently most compelling sub-6 GHz physical-layer technology for future wireless access. The main concept is to use large antenna arrays at base stations to simultaneously serve many autonomous terminals. The rich and unique propagation signatures of the terminals are exploited with smart processing at the array to achieve superior capacity. Massive MIMO splendidly offers two most desirable benefits (LARSSON, E. G. and Van der Perre, L., 2017):

a) **Excellent spectral efficiency**, achieved by spatial multiplexing of many terminals in the same time-frequency resource. Efficient multiplexing requires channels to different terminals to be sufficiently different, which has been shown to hold, theoretically and experimentally, in diverse propagation environments. Specifically, it is known that Massive MIMO works as well in line-of-sight as in rich scattering (LARSSON, E. G. and Van der Perre, L., 2017).
b) **Superior energy efficiency**, by virtue of the array gain, that permits a reduction of radiated power. Moreover, the ability to achieve excellent performance while operating with low-accuracy signals and linear processing further enables considerable savings (LARSSON, E. G. and Van der Perre, L., 2017).

The key technological characteristics of Massive MIMO are:

a) **Fully digital processing**; each antenna has its own RF and digital baseband chain. Signals from all antennas at each base station are processed coherently together. Core advantages of fully digital processing include the avoidance of specific assumptions on propagation channel, the possibility to measure the complete channel response on the uplink and respond fast to changes in the channel. Interestingly, recent assessments show that the full digital processing may not only offer superior performance but also better energy efficiency, a trend which may be reinforced by the ongoing development of tailored low-power circuits (LARSSON, E. G. and Van der Perre, L., 2017).
b) The reliance on **reciprocity of propagation and TDD operation**, enabling downlink channels to be estimated from uplink pilots, and obviating the need for prior or structural knowledge of the propagation channel (LARSSON, E. G. and Van der Perre, L., 2017).
c) **Computationally inexpensive precoding/decoding** algorithms, taking the form of maximum-ratio (known also as conjugate beamforming) or zero-forcing processing. Massive MIMO functions equally well with single-carrier transmission and OFDM. Notably, conjugate beamforming with OFDM is equivalent to time-reversal in a single-carrier system (LARSSON, E. G. and Van der Perre, L., 2017).
d) **Array gain**, resulting, in principle, in a closed-loop link budget enhancement proportional to the number of base station antennas (LARSSON, E. G. and Van der Perre, L., 2017).
e) **Channel hardening**, which effectively removes the effects of fast fading. Operationally, each terminal-base station link becomes a scalar channel whose gain stabilizes to a deterministic and frequency-independent constant. This greatly simplifies resource allocation problems (LARSSON, E. G. and Van der Perre, L., 2017).
f) The provision of **uniformly good quality of service to all terminals in a cell** - facilitated by the link budget improvement offered by the array gain, and the interference suppression capability offered by the spatial resolution of the array. Typical baseline power control algorithms achieve max-min fairness among the terminals (LARSSON, E. G. and Van der Perre, L., 2017).
g) **Autonomous operation of the base stations**, with no sharing of payload data or channel state information with other cells, and no requirements of accurate time synchronization (LARSSON, E. G. and Van der Perre, L., 2017).
h) The possibility to **reduce accuracy and resolution** of transceiver frontends, and the digital processing and number representations in computations (LARSSON, E. G. and Van der Perre, L., 2017).

The attractive properties of propagation -- penetration of solid objects and diffractive behavior -- and the maturity of hardware renders Massive MIMO primarily a below-6 GHz technology for radio access.

This is also the region where spectrum is most valuable. Arrays have attractive form factors even for large numbers of antennas: in the 3.5 GHz TDD band, a half-wavelength-spaced rectangular array with 200 dual-polarized elements is about 0.6 x 0.3 meters large; in practice, larger antenna spacing may be desired and is easily afforded. However, systems operating at higher frequencies up to millimeter-waves may also benefit from the application of Massive MIMO, especially when these systems would need to support multi-user access in potentially non-Line-of-Sight scenarios (LARSSON, E. G. and Van der Perre, L., 2017).

G. Multi-beam antennas for 5G radio

Using the previously-mentioned traits, the 5G radio is expected to utilize the same techniques in order to achieve better global efficiency. With the demanding system requirements for the fifth-generation (5G) wireless communications and the severe spectrum shortage at conventional cellular frequencies, multi-beam antenna systems operating in the millimeter-wave frequency bands have attracted a lot of research interest and have been actively investigated. They represent the key antenna technology for supporting a high data transmission rate, an improved signal-to-interference-plus-noise ratio, an increased spectral and energy efficiency, and versatile beam shaping, thereby holding a great promise in serving as the critical infrastructure for enabling beamforming and massive multiple-input multiple-output (MIMO) that boost the 5G (HONG, W. et al., 2017, pp.6231-6249).

The idea behind 5G is to increase transmission bit rates by using frequency bands higher than those of existing frequency bands and widening the signal bandwidth. However, as radio propagation loss increases in high frequency bands, the application of massive-element antennas each consisting of more than 100 antenna elements has been studied as 5G multi-antenna technology (SUYAMA, S. et al., 2016, pp.29-39). Application of a massive-element antenna makes it possible to compensate for the radio propagation loss by adaptively controlling antenna directivity and increase bit rate by the spatial multiplexing of signals. A basic 5G architecture proposed model, consisting of C/U separation by Phantom cell concept is comprised of multiple instances of small cell (or quasi-macro cell) in an overlay configuration. In this particular scheme, the macro cell uses the Ultra High Frequency band (UHF) (0.3-3 GHz) employed by the existing system while overlaid small cells use higher frequency band, namely, the low Super High Frequency band (SHF) from 3-6GHz, high SHF band (6-30 GHz), and Extremely High Frequency band (EHF) from 30-300 GHz. This model establishes a connection link for the Control Plane (C-plane) that handles control signals via the macro cell and a connection link specifically for the User Plane (U-plane) that handles user data via overlaid cells, i.e., C/U split connections. Another operation that is supported is the introduction of Massive-element antennas in high-frequency band cells. To achieve higher bit rates than 10Gbps requires bandwidths of several 100 MHz. Particularly, to resolve this issue, massive-elements antennas are introduced in high frequency bands. When using a flat antenna array with a uniform antenna spacing as a massive-element antenna in the 20 GHz band, and when setting the element spacing to half the wavelength (7.5 mm), it becomes possible to mount 256 elements in an area approximately of 12 cm^2. Generally, for the same area, the number of elements that can be mounted can be significantly increased when using higher frequency bands with shorter wavelengths. A massive-element antenna can be used to generate sharp beams by controlling the amplitude and phase of transmitted and received signals from each element (beamforming) (SUYAMA, S. et al., 2016, pp.29-39).

H. Evolution of the antenna systems

The development of new types of antennas can be granted mostly to the availability of 3D printing technologies. The 3D printing enables usage of various meta-materials that have not been previously tested, and currently, they can yield some unprecedented performance for the actual antennas that they are used for. Metamaterials are made by arranging naturally occurring materials in a specific pattern that produces an electromagnetic response that is not found in nature. The periodic structures created are at scales that are smaller than the wavelengths of the phenomena they influence and can create materials with negative indexes that control electromagnetic energy in ways that cannot be done with natural materials. In traditional active electronically scanned arrays (AESA), phase shifters embedded in control circuitry steer the beam direction. Metamaterial-based AESAs can steer the beam without phase shifters, which reduces system complexity, eliminates a source of power loss and simplifies waste-heat dissipation. There are a couple of companies using unique metamaterial structures developed for this application. For example, as represented in the October 2016 issue of Microwave Journal (ELSALLAL, M. W. et al., 2016), The MITRE Corporation is investigating a new generation of 3D printing to realize the complex geometries of wideband phased array and metamaterial designs using commercial, low-cost, compact, desktop printers. Samples of the 3D printed plastic and conductive ink printed at room temperature were characterized over frequency. The polylactic acid (PLA) dielectric constant and loss tangent are found to be stable up to 18 GHz. The PLA internal architecture was varied to achieve lower effective dissipation factors, which extends usefulness to high frequency applications. Micro-strip line samples were fabricated with simulated and measured insertion loss data validating the high conductivity through mm-Wave frequencies. A 3D printed monopole Wi-Fi antenna was built and tested, showing good performance and agreement with simulations (HINDLE, P., 2018).

2.1.7. Security architecture of 4G LTE

The communication networks should provide adequate level of security in terms of services, and therefore, a suitable cost-to-benefit ratio of deployment is taken into consideration. A complete security of a system is impossible, and the focus of establishing a secure environment should be directed towards minimizing the potential vulnerabilities of the network, since attackers tend to exploit those in order to achieve a goal. Accordingly, a system is secure as its least reliable security asset. The least secure entities in a mobile network are the access stratum (the radio network) and the mobile terminals (UE). The main features of a secure network are as follows: Confidentiality, Integrity and Non-repudiation. To reach the particular goals, a mobile network should fulfill some requirements for mechanisms such as: Authentication, Access control and Network availability. The wireless/mobile networks also implement the security features of the fixed networks, with particular modification of some protocols and methods that are adjusted to correspond to the requirements of the nature of the networks (BOUDRIGA, N., 2010).

The security architecture in LTE can be categorized for both Non-Access Stratum (NAS) layer security and Access Stratum (AS) layer security (see Figure 44). The NAS security is carried out for NAS messages and belongs to the domain of UE and MME. Accordingly, the NAS message communication between the UE and MME is protected and ciphered with additional NAS security header. Conversely, the AS security is carried out for the RRC and user-plane data, and belongs in the sphere of the UE and eNB. The ciphering and integrity protection are being carried out by the PDCP layer in the UE and eNB side. The RRC messages have their integrity protected and ciphered, and the user-plane data is only being ciphered.

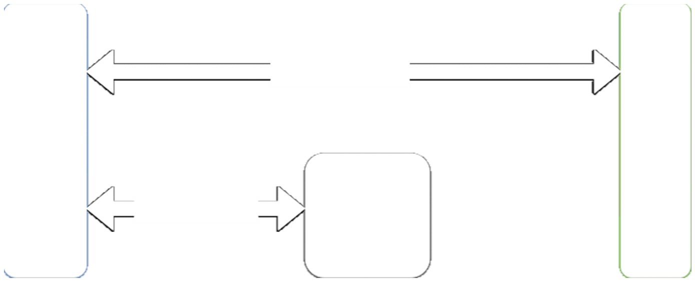


Figure 44. Security distribution in LTE

The Evolved Packet System (EPS) is designed to interwork with legacy systems, and has thus adopted the security mechanisms from 3G UMTS for the sake of backward-compatibility; but however, many new extensions and enhancements are introduced (FOSBERG, D. et al., 2013). In LTE, after the UE has been identified, the MME fetches authentication data from the home network. Then, the MME triggers the authentication and key agreement protocol (AKA) with the UE, which is the actual DIAMETER service. After this protocol is finalized, the MME and the UE share a secret key K_{ASME} (ASME refers to Access Security Management Entity). The management entity in the EPS system is the actual MME. At this point, the MME and UE can derive further keys from the K_{ASME}. For confidentiality and integrity protection of the signaling data between the MME and UE derived keys are being used. Additional key is being derived and thus transported to the eNB. Furthermore, three more keys are subsequently being derived, both at the eNB and in the UE. Two of these keys are then used for confidentiality and integrity protection of the signaling data between the eNB and the UE (AS security in Figure 44). The third key is used for confidentiality protection of the U-plane data between the UE and eNB, which is the NAS security part. Besides the security of signaling and UP data originated or terminated by the UE, there is also confidentiality and integrity protection for the signaling and user data being transported over the interface between the eNB and the EPC, namely the S1-MME interface. The signaling data is transported over the S1-U interface, between the UE and the S-GW. As a cryptographic measurement for protection applied to the S1 interfaces, the IPsec mechanism (IETF, 2011) [Standard RFC6071] is employed. Additionally, the X2 interface is being protected by IPSec with keys that are not specific to the UE where cryptographic protection is utilized (FOSBERG, D. et al., 2013).

A. Authentication and key agreement protocol (AKA)

In LTE, the authentication is based on the AKA procedure. The key agreement and exchange is a crucial process, which enables secure access of the users to the network core (EPC). The AKA procedure is a crucial process of the Diameter service, which takes place via the S6a interface between the MME and the HSS. The EPS AKA procedure is combination of a procedure for generation EPS authentication vectors (AVs) in the HSS upon request from MME, a procedure to mutually authenticate and establish a new shared key between the serving network (SN) and the UE and a procedure to distribute authentication data inside and between serving networks. The MME invokes the procedure by requesting EPS AVs from the HSS. The Authentication Information Request shall include the IMSI (International Mobile Subscriber Identity – used to identify the user or acellular network and is a unique identification associated with all cellular networks), the SN id (Serving Network ID – refers to the network accessed by the UE) of the requesting MME, and an indication that the authentication information is requested for EPS. The SN id is required for the computation of KASME in the HSS. Upon the receipt of the Authentication Information Request from the MME, the HSS may have pre-computed AVs available and retrieve them from the HSS database, or it may compute them on demand. The HSS sends an Authentication Information Answer back to the MME that contains an ordered array of n EPS AVs (1 . . . n). If n > 1, the EPS AVs are ordered based on sequence number. The 3GPP specification TS 33.401 (3GPP, 2015) [specification TS 33.401] recommends n = 1, so that only one AV is sent at a time, because the need for frequently contacting the HSS for fresh AVs has been reduced in EPS through the availability of the local master key KASME, which is not exposed in a way similar to Ciphering Key in 3G (CK) and Integrity Key in 3G (IK) in UMTS and, hence, does not need to be renewed very often. Based on the local master key, and keys derived from it, an MME can offer secure services even when links to the HE are unavailable. Furthermore, pre-computed AVs are no longer usable when the user moves to a different SN owing to the binding of the local master key KASME to the SN id. However, pre-computation may still be useful when the next request for AVs is likely to be issued by an MME in the same SN, which may be the case, for example, for a user in his home network. Each EPS AV is good for one run of the AKA procedure between the MME and the USIM (FOSBERG, D. et al., 2013).

The purpose of this procedure is the authentication of the user and the establishment of a new local master key KASME between the MME and the UE, and, furthermore, the verification of the freshness of the AV and authentication of its origin (the user's home network) by the USIM. KASME is used in subsequent procedures for deriving further keys for the protection of the user plane (UP), RRC signaling and NAS signaling. The MME invokes the procedure by selecting the next unused EPS AV from the ordered array of EPS AVs in the MME database (if there is more than one). If the MME has no EPS AV, it requests one from the HSS. The MME then sends the random challenge RAND and the authentication token for network authentication AUTN from the selected EPS AV to the ME, which forwards it to the USIM. The MME also generates a key set identifier in EPS (eKSI) and includes it in the Authentication Request. When a user moves around, the MME serving the UE may change. When the UE then sends an Attach Request, or a Tracking Area Update Request [TS23.401], the UE will, in general, use its temporary identity, the GUTI, in order to protect the confidentiality of its permanent identity, the IMSI. But the new MME is not able to make sense of the GUTI, so it has only two choices: request the permanent identity from the UE and break identity confidentiality in this way, or ask the old MME, which issued the GUTI, to translate the GUTI to the user's IMSI. The old MME will also send back authentication data to the new MME. Exactly what kind of authentication data is allowed to be exchanged between old and new MME depends on whether the two MMEs reside in the same or in different SNs (FOSBERG, D. et al., 2013).

When two parties engage in security-related communication, for example when running an authentication protocol or exchanging encrypted data, they need an agreed set of security parameters, such as cryptographic keys and algorithm identifiers, for the communication to be successful. Such a set of security parameters is called a security context. There are different types of security context depending on the type of communication, and the state the communicating parties are in. Additionally, entities may store security context data locally even when not engaged in communication. The distinction between locally stored security context data and security context shared between two communicating parties for the purpose of running a security protocol is useful in principle, but it is a bit academic and not much adhered to in practice. As the potential for confusion is low, the common practice is being followed and declared only in terms of security contexts. Several different types of security context have been defined for EPS so as to have shorthand notations available for the various sets of security parameters used in particular situations (FOSBERG, D. et al., 2013).

There are several security contexts for LTE, among which few will be elucidated for clarification. One example is the EPS security context that is comprised of EPS NAS security context and EPS AS security context, or specifically contexts for the Non-Access Stratum and the Access Stratum. The EPS NAS security context is used for protecting the NAS of EPS between the UE and the MME, and it may even exist when the UE is in de-registered state. This context consists of KASME with the associated key set identifier eKSI, the UE security capabilities and the NAS uplink and downlink COUNT values. These counters are relevant also for security as they are used as input parameters to key derivations in certain state and mobility transitions and, in conjunction with integrity protection, for preventing message replay. Separate pairs of NAS COUNT-values are used for each EPS NAS security context. The EPS NAS security context is called full if it additionally contains the keys KNASint and KNASenc ('NAS keys' for short) and the identifiers of the selected NAS integrity and encryption algorithms, otherwise it is called partial. An EPS security context containing a full or partial EPS NAS security context is also called full or partial, respectively. However, both KNASint and KNASenc can be derived from the KASME when the NAS integrity and encryption algorithms are known. Thus, they need not necessarily be stored in the memory (FOSBERG, D. et al., 2013).

The EPS AS security context is used for the AS of EPS between the UE and the eNB, and it only exists when cryptographically protected radio bearers are established and is otherwise void. For an EPS AS security context to exist, the UE needs to be in connected state. This context consists of the cryptographic keys at AS level (i.e. between the UE and the eNB) with their identifiers, the NH, the Next Hop Chaining Counter parameter (NCC) used for NH access key derivation (see Section 9.4), the identifiers of the selected AS level cryptographic algorithms for integrity protection of RRC and (in the context of relay nodes) UP, and ciphering of RRC and UP, and the counters used for replay protection (FOSBERG, D. et al., 2013).

When the USIM is enhanced for EPS, a part of the EPS native security context is stored on the USIM under certain conditions. When the USIM is not enhanced for EPS, the nonvolatile part of the ME memory takes on an equivalent role and stores that part of the EPS native security context. The idea is that, in both cases, an EPS native security context shall be kept even when the UE de-registers or is switched off. When the UE registers again and goes to connected state, the EPS native security context can be retrieved from storage and used to protect the initial NAS message. By re-using the stored context, a new run of EPS AKA can be avoided. A mapped context is never stored on the USIM. A mapped EPS security context is kept in a transition to idle state, and, if available, is used to protect the initial NAS message when the UE transitions back to connected state. A mapped EPS security context is deleted when the UE de-registers (FOSBERG, D. et al., 2013).

B. DIAMETER protocol in LTE

In 1980, the signaling protocol SS7 was introduced by the International Telecommunications Union to control telephone call sessions through point-to-point connectivity. Only after scalability and management issues became apparent, the need for centralized management was clear and a new network entity called the signal transfer point was introduced to manage, connect and route SS7 traffic. SIP, the communication protocol to support voice calls over the Internet, has similar origins. Originally SIP was intended to connect network entities point-to-point, which lasted a while until management, interoperability and routing requirements evolved and the session border controller was introduced in 2003 to handle and solve SIP management issues. Beginning in the 1970s, the data plane was also initially designed on point-to-point connections. Few people imagined that there would be so many connections, and much data and signaling traffic to render this architecture obsolete. But, as we know, data and signaling traffic increased, which expedited the need for switches, routers and load balancers to support signaling traffic management and scaling. When Diameter was first introduced by the Internet Engineering Task Force (IETF), it included the concepts of Diameter agents that can proxy, route and balance Diameter traffic to provide scalability and management requirements (IETF, 2003) [Diameter base protocol standard]. However, when the 3GPP promoted Diameter as the foundation for signaling in IMS and EPC architectures, it left Diameter Agents out, perhaps thinking that the introduction of a distributed architecture would avoid the need for Diameter signaling management (RUSSELL, Travis, 2016).

Unlike legacy signaling protocols, which were predominately circuit-switch based, Diameter protocol is always packet based and uses TCP or SCTP as a transport protocol to enhance reliability. However, TCP creates twice as much network traffic due to the need to ACK all messages (meaning every message must send a receipt message). Sending a receipt automatically doubles the number of signaling messages. As clearly seen, the move to an all-IP network significantly increases the amount of signaling. Although the move from circuit switch to packet switch will bring many other advantages, it generates tons more data traffic, and data is one of the major forces behind growth in signaling. Operators need to confront the blitz of signaling from a multitude of fronts never seen before and must be managed before damage is caused to their networks. LTE introduced a major change to the overall 2G GSM architecture and thus, network elements were consolidated to only support IP transport. Everything from the radio to the packet core runs over IP transport. The protocols used in the network were changed as well, with SS7 being replaced by Diameter (for authentication, authorization and accounting) and the Session Initiation Protocol (SIP). SIP provides the signaling for voice and multimedia in the network, replacing SS7 ISUP call control. However, 3GPP decided against simple VoIP as the means of supporting voice and standardized on IP Multimedia Subsystem (IMS) as the architecture for the SIP network. Voice over LTE (VoLTE) requires IMS to support voice in 4G networks (RUSSELL, Travis, 2016).

DIAMETER is an evolution from the older protocol named RADIUS (Remote Authentication Dial-In User Service), originally developed to support PPP connections. RADIUS managed authentication, authorization and accounting (AAA) over these dial-up connections. Networks have evolved majorly over the years, and RADIUS became too limited for modern-day services, especially in the modern mobile/wireless networks. Since the RADIUS protocol is acronym, Diameter is not. The name emerged from an engineering joke, starting from the point that Diameter is twice the protocol RADIUS is, analogously to the mathematical terms for calculation of radius and diameter of a circle. Disregarding that fact, the real distinction between these two protocols is in their functionality. Namely, RADIUS was not able to provide fail-over procedures. There is no means for servers to communicate that they are going out of service, or for the orderly termination of sessions for any reason. When an error emerges in RADIUS, there are no existing procedures for attempts of rectification of the error. The session simply fails, which is unacceptable for many modern-day services. Also, RADIUS assumes that security is managed in the back office building systems rather than in the network. This stems from the notion that network connections between service providers can be trusted. In today's world, security must be implemented at all levels, using layered security architecture; starting from the transmission level and up to the application level. One of the issues in SS7 is the lack of authentication at the transport layer (between networks). This is resolved, as stated previously, by using IPSec. Diameter, on the contrary, supports the use of encryption at the transport layer. This is as important in today's implementations where Diameter is replacing SS7 Mobile Application Part (MAP) applications in the 4G LTE wireless networks. RADIUS uses the UDP protocol as its transport, which is very unreliable. As previously stated, Diameter is based on the SCTP, which is a congregation of the UDP and TCP protocols with connection control (RUSSELL, Travis, 2016).

DIAMETER is the most complicated AAA protocol existing (RUSSELL, Travis, 2016), and herein the brief explanation that follows is a synopsis of the rationale. Diameter is an agent-based protocol, which is designed as Peer-to-Peer (P2P) architecture. There are two constituents upon the architecture is built on: a Diameter node (the client) and a Diameter agent (the server). It is possible also to implement few Diameter agents (RUSSELL, Travis, 2016):

- **Relay Agent (DRA)** – Used to forward messages to the appropriate destination in dependence to the information contained in the message. The relay agent can aggregate requests from different realms (regions/FQDN) to a specific realm. That eliminates the onerous configurations of network access servers for every Diameter server alteration.
- **Proxy Agent** – Can be used to forward messages with a difference from the relay agent that it can modify the message composition and deliver value-added services; administer rules on different messages or perform various managerial processes for a particular realm.
- **Redirect Agent** – Represents a centralized configuration repository for other Diameter nodes. When a message is received, the agent checks the routing table and returns a response message together with redirection information to the original sender. This is useful for other Diameter nodes, since a local routing table can then be omitted on all the nodes individually and they can look up for a redirect agent instead.
- **Translation Agent** – It is a special agent that converts message from one AAA protocol to another. The translation agent is useful when operators need to integrate a user database of two application domains, while keeping the original AAA protocols. Another case where the translation agent is useful would be a situation of migration to Diameter protocol from another establishment, where it can provide backward compatibility for smoother migration.

The summary of the agents is represented in Figure 45:

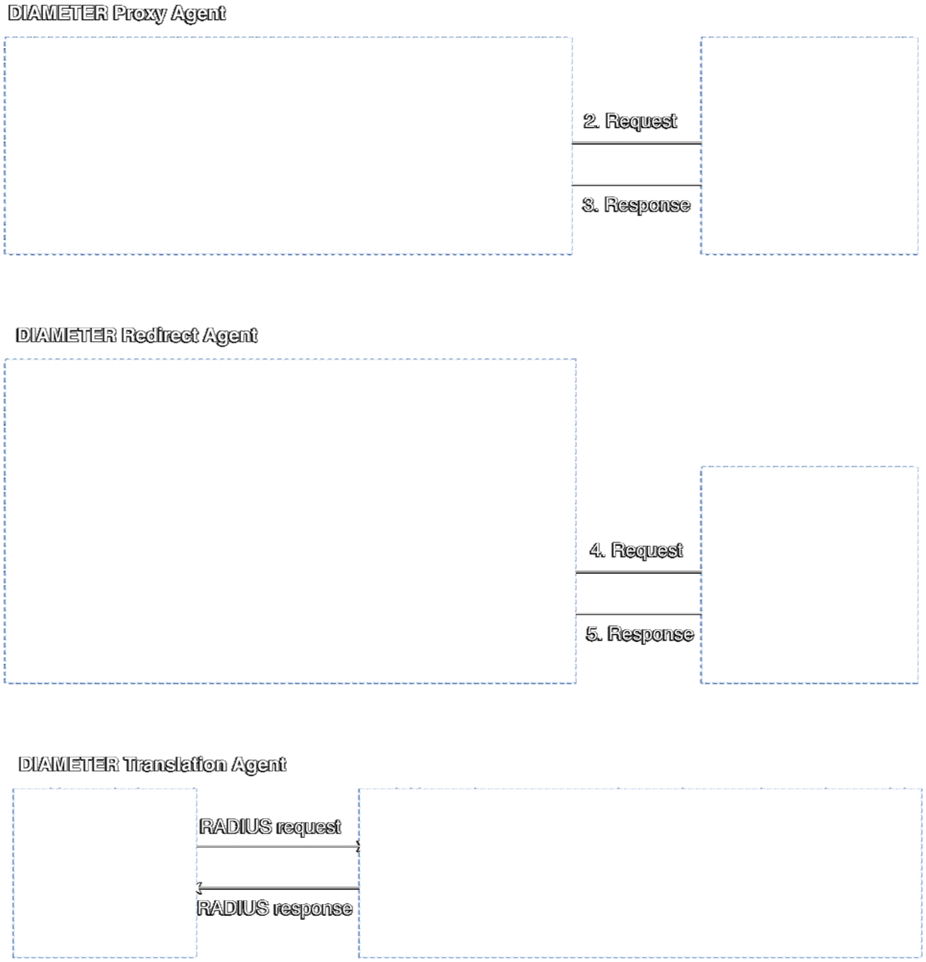


Figure 45. DIAMETER agents

A Diameter message is the elementary unit used to deliver a notification or issue a command to other Diameter nodes. The Diameter protocols defines several type of messages that are identified by their function (command code). A Diameter packet format is given in Figure 46.

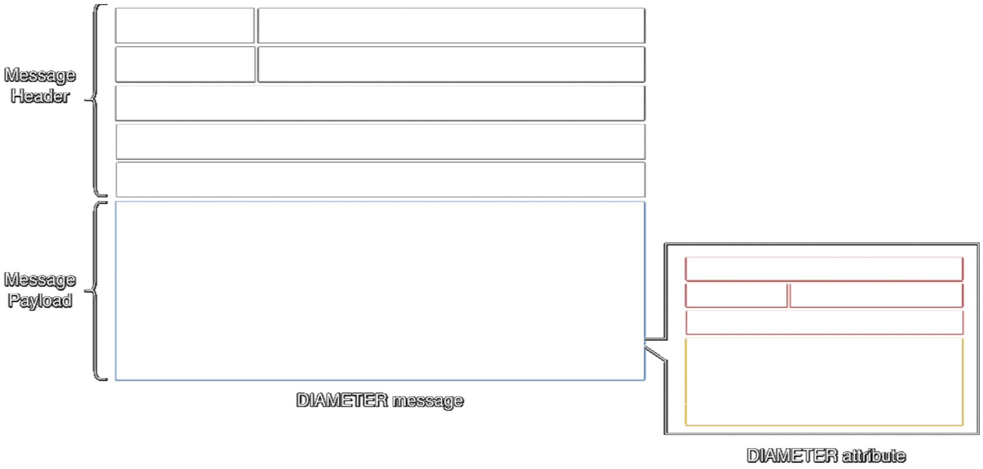


Figure 46. DIAMETER packet format

Diameter works on the principle of peer discovery: the Diameter agent needs to broadcast which application it supports, along with the provided security level. Thus, it can be decided how the Diameter clients can depend on the desired Diameter application, security level, and realm info to look up suitable first-hop Diameter nodes to which they can forward Diameter messages. The Diameter node maintains a peer table in which host addresses are stored, as well as other information (status, security-related information etc.). Also there is a peer routing table maintained, which contains four important columns that require extra attention for message routing: Realm name, Application name, Action to be taken for the target message and Reference to an entry in the Peer Table. After the peer is discovered, the following procedure is to establish a connection with that peer. Here, the AKA procedure takes place, which in Diameter is initiated via the SCTP protocol. Due to the fact that Diameter is a peer-to-peer based protocol, multiple connections per node may exist; this inclines on the existence of a session, or a logical connection between two Diameter nodes that has multiple connections. Each session in Diameter is associated with a client-generated Session-ID that is globally and generally exclusive. As indicated in Figure 45, a Diameter session is established in the request-response paradigm as in the case with other client-server communication models.

C. Protection of Signaling and User data

Protecting communication over the air and inside the network is important so that confidentiality of information can be assured and attacks on the communication channels can be more easily mitigated. Evolved Packet System (EPS) has two layers of security for signaling: the first layer is between User Equipment (UE) and the base stations, and the second layer is between UE and the core network. The user plane data packets are protected between UE and base stations and further in the network in hop-by-hop manner. In this chapter, we describe in detail how the communication between UE and network and inside the network is protected. Long Term Evolution (LTE) has separate signaling and user planes. The signaling plane is further divided into signaling between UE and base stations (i.e. Access Stratum, AS) and between UE and core network (i.e. Non-Access Stratum, NAS). Signaling protection consists of ciphering and integrity protection with replay protection; for the user plane (data) on the air interface only ciphering is provided, with the exception of the Un air interface between a relay node and a Donor evolved NodeB (DeNB) (RUSSELL, Travis, 2016).

D. EPS cryptographic algorithms

One principle that has been used in the design of EPS security is that of algorithm agility: the system should be flexible in the sense that new algorithms can be introduced and outdated ones can be removed, both without major hassle. Therefore, it is expected that in the future new algorithms would appear in EPS, but they are potentially not even invented at the time of writing and hence naturally not yet discussed in this chapter. The need for better algorithm agility has stemmed from experiences with 2G and 3G systems where new algorithms have been introduced and one algorithm (A5/2) has also been removed from the 3rd Generation Partnership Project (3GPP) system. A general principle for any standardized mechanisms (including non-security-related ones) is that options should only be introduced if they serve a clear benefit for the system as a whole. If the difference between one option and another is more like a matter of taste, or if the benefit of each option over the others materializes only in a small minority of all circumstances, options should not be introduced because they complicate the system, add development cost and put the interoperability at risk. Hence, the number of different algorithms should be kept small and introduction or removal of algorithms should be done only after it is clear that such action adds value to the system as a whole (RUSSELL, Travis, 2016).

There are four types of cryptographic algorithms used in LTE:

- **Null algorithms** - When the protection needs to be explicitly turned ‘off’ instead of just not ‘on’. The start of no-protection has to be done explicitly, it is simplest from the system point of view to use procedures for starting no-protection similar to those that are used for starting protection. Bear in mind here that the start of protection needs to be done explicitly as well, mainly for synchronization reasons. Thus, instead of choosing a proper algorithm to be put in place in order to start protection, we choose a Null algorithm to be put in place to start no-protection. Indeed, a Null algorithm is not a cryptographic algorithm; in fact it is not really an algorithm at all. A way of realizing a Null algorithm is to do some very simple operation, just in order to make it explicit that a Null algorithm has indeed been in use. This is the option that has been chosen for the Null integrity algorithm in EPS: regardless of the message content or key or any other parameter, a 32-bit string of all zeros is appended to the message as the result of applying the Null integrity algorithm (RUSSELL, Travis, 2016).

- **Ciphering algorithms** - The encryption mechanisms used in EPS are very similar to those used in 3G. There are many differences between EPS and 3G in how keys are generated and managed but, once the correct key is in place, the usage of the key is very similar in these systems. This is fortunate in the sense that it allows terminals to use some internal components for both Long Term Evolution (LTE) and 3G. The both levels of security, the NAS and AS can have the same encryption algorithm. It would be easy to draw the conclusion that the same set of algorithms that is in use for 3G would also be a good choice for EPS. Presented the opportunity in a new system, the EPS adopts new elevated approaches. The two 3G algorithms are, at the time of writing, UEA1 (UMTS Encryption Algorithm) based on KASUMI and UEA2 based on SNOW 3G. It is notable that the leading general-purpose algorithm Advanced Encryption Standard (AES) is not among the two. Briefly, AES was not ready yet when KASUMI-based UEA1 was chosen, while SNOW 3G-based UEA2 was the preferred choice as the base algorithm, over AES, because its design was more different from that of KASUMI.

  To use the AES algorithm, it is possible to adopt another approach, since the AKA protocol does not require standardization of the cryptographic algorithms. As for the purpose of this project, the MILENAGE set is employed, which is developed accordingly to the [TS35.205]; [TS35.206]; [TS35.207] and [TS305.208] 3GPP specifications (3GPP, 2017) (RUSSELL, Travis, 2016). The MILENAGE algorithms use a core function of a block cipher, in which both block size and key size are 128 bits. For instance, the (basic form of) the AES algorithm can be used as the core function (RUSSELL, Travis, 2016).
- **Integrity protection algorithms** - Many of the facts explained for the background of EPS ciphering algorithms also apply to integrity algorithms. The integrity protection mechanisms are similar in both 3G and LTE, although there are big differences in key management. Each integrity algorithm applies as such to both AS-level and NAS-level protection. In order to have a good security margin against progress in cryptanalysis, two different algorithms are in place from the beginning of EPS. From an implementation point of view, especially for terminals, it would be good to have algorithms that are usable also for some other purposes. There is a typical practice of using the same core cryptographic functions for both ciphering and integrity purposes. This practice is also mainly due to re-usability benefits, and there are no cryptographic reasons behind it. However, no heavy arguments were found that would have spoken against such a practice, so it was decided that the two integrity algorithms that are supported from the start are based on AES and SNOW 3G (RUSSELL, Travis, 2016).
- **Key derivation algorithms** - The EPS key hierarchy is significantly more complex than that of 3G or GSM. One consequence is that there has to be a standardized way to derive keys from each other. From the security point of view, it is crucial that the derivation is one-way: it should not be possible to use physically less protected keys on the lower layers of the hierarchy to get information about the physically more protected keys that are higher up in the hierarchy. In addition, two keys derived from the same key should be independent. Notably, the difference in

the physical protection refers rather to the network side; on the UE side there are fewer differences. Although 3G access security did not require defining a standardized Key Derivation Function (KDF), it has been needed for other 3GPP features. Most notably, the Generic Bootstrapping Architecture (GBA) includes the derivation of new keys as one of its core features. EPS key derivation re-uses the standard KDF of GBA. The core of the KDF is the cryptographic hash function SHA-256. It is used in the keyed HMAC (Keyed-Hash Message Authentication Code) mode [RFC2104], where the key for HMAC is the 'mother' key from which the lower layer key is derived. The other input parameter for HMAC is called the message, a name motivated by the primary use of HMAC for message integrity purposes (RUSSELL, Travis, 2016).

2.2. LTE-Advanced and LTE-Advanced Pro as a step before 5G

In LTE-Advanced focus is on higher capacity: The driving force to further develop LTE towards LTE–Advanced – LTE. Release 10 was to provide higher bitrates in a cost efficient way and, at the same time, completely fulfil the requirements set by ITU for IMT Advanced, also referred to as 4G. In this thesis, the latest implementation of the Release 10 and forward is used for the research, where the experiments incorporate software-defined solutions that include LTE-A attributes. Namely, the aim of the LTE-A features is:

- Increased peak data rate, DL 3 Gbps, UL 1.5 Gbps
- Higher spectral efficiency, from a maximum of 16bps/Hz in R8 to 30 bps/Hz in R10
- Increased number of simultaneously active subscribers
- Improved performance at cell edges, e.g. for DL 2x2 MIMO at least 2.40 bps/Hz/cell.

LTE-A starts with the 3GPP Release 10 in 2011, powered with new functionalities such as Carrier Aggregation (CA), enhanced use of multi-antenna techniques and support for Relay Nodes (RN). The most straightforward way to increase capacity is to add more bandwidth. Since it is important to keep backward compatibility with R8 and R9 mobiles the increase in bandwidth in LTE-Advanced is provided through aggregation of R8/R9 carriers. Carrier aggregation can be used for both FDD and TDD. Each aggregated carrier is referred to as a component carrier. The component carrier can have a bandwidth of 1.4, 3, 5, 10, 15 or 20 MHz and a maximum of five component carriers can be aggregated. Hence the maximum bandwidth is 100 MHz. The number of aggregated carriers can be different in DL and UL, however the number of UL component carriers is never larger than the number of DL component carriers. The individual component carriers can also be of different bandwidths. In LTE-Advanced, the possibility for efficient heterogeneous network planning – i.e. a mix of large and small cells - is increased by introduction of Relay Nodes (RNs). The Relay Nodes are low power base stations that will provide enhanced coverage and capacity at cell edges, and hot-spot areas and it can also be used to connect to remote areas without fiber connection. The Relay Node is connected to the Donor eNB (DeNB) via a radio interface, Un, which is a modification of the E-UTRAN air interface Uu. Hence in the Donor cell the radio resources are shared between UEs served directly by the DeNB and the Relay Nodes. When the Uu and Un use different frequencies the Relay Node is referred to as a Type 1a RN, for Type 1 RN Uu and Un utilize the same frequencies, see figure 7. In the latter case there is a high risk for self-interference in the Relay Node, when receiving on Uu and transmitting on Un at the same time (or vice versa). This can be avoided through time sharing between Uu and Un, or having different locations of the transmitter and receiver. The RN will to a large extent support the same functionalities as the eNB – however the DeNB will be responsible for MME selection (3GPP, 2011) [Release 10 specification 36.912].

One of the imperative advantages of LTE Advanced is the capability to utilize advanced topology networks; optimized heterogeneous networks with a mix of macrocells with low-power nodes such as picocells, femtocells and new relay nodes (3GPP, 2011) [Release 10 specification 36.912]. LTE-Advanced exploits these capabilities for application of smaller cells and bringing the services closer to the end user, while ensuring fairness and quality of experience. LTE Advanced also introduces multicarrier for usage of ultra-wide bandwidth, namely up to 100 MHz of channel bandwidth that can support very high data rates. In the research phase, various suggestions have been studied as candidates for LTE Advanced (LTE-A) technologies. The proposals (3GPP, 2011) [Release 10 specification 36.912] are represented as:

- Support for relay node base stations
- Coordinated multipoint (CoMP) transmission and reception
- UE Dual TX antenna solutions for SU-MIMO and diversity MIMO, commonly referred to as 2x2 MIMO
- Scalable system bandwidth exceeding 20 MHz, up to 100 MHz
- Carrier aggregation of contiguous and non-contiguous spectrum allocations
- Local area optimization of air interface
- Nomadic / Local Area network and mobility solutions
- Flexible spectrum usage
- Cognitive radio
- Automatic and autonomous network configuration and operation
- Support of autonomous network and device test, measurement tied to network management and optimization
- Enhanced precoding and forward error correction
- Interference management and suppression
- Asymmetric bandwidth assignment for FDD
- Hybrid OFDMA and SC-FDMA in uplink
- UL/DL inter eNB coordinated MIMO
- SONs, Self-Organizing Networks methodologies

Within the range of system development, LTE-Advanced and WiMAX 2 can employ up to 8x8 MIMO configuration and 128 QAM in downlink direction. Particularly, that will enable a 100 MHz aggregated channel bandwidth, and approximately 3.3 Gbit peak download rates per sector of the base station under ideal conditions. Advanced network architectures combined with distributed and collaborative smart antenna technologies provide several years road map of commercial enhancements (3GPP, 2011) [Release 10 specification 36.912].

2.3. Virtualization and cloud computing

A key concept for the progression towards 5G networks are the cloud computing and virtualization paradigms. Virtualization offers many advantages in the present, which are being exploited for the benefit of developing the 5G evolution initiative. In other words, the existing hardware deployed to serve the 4G LTE and LTE-Advanced infrastructures is being emulated into software and virtualized, i.e. adapted to operate on a generic computing machine (a PC or a server). From there, a congregated research is being carried out in order to achieve automation of deployment and portability of emulated mobile network platforms. As an attempt to move the manual configuration into automated solution, the networking industry formulates the concepts of network virtualization (NV), network function virtualization (NFV) and the software-defined networking (SDN). Factually, the three concepts serve the purpose of network configuration automation and scalability in order to support virtualized and cloud environments. Another reason those software-defined schemes exist is explicit increment of networking agility, as well as simplifying service and application delivery methods. Therefore, the concept of *network mobility* can be solved with the option for programmability. Additionally, those three concepts are independent from each other and can be implemented individually, without the impairment of their function. Namely, *virtualization* refers to the *"process of abstracting computing resources such that multiple applications can share a single physical hardware"* (VAEZI, Mojtaba and Zhang, Ying, 2017).

As denoted, the virtualization refers mostly to server virtualization, where a particular physical server has an abstraction formed and is decomposed into virtual entities. The virtual constituents are assembled into a *hypervisor* which is in fact the virtualization software (like KVM, VirtualBox or VMware). The virtual constituents are in actual fact a virtual CPU, virtual RAM, and virtual NIC etc. Besides the represented entities, the storage can also be virtualized. This allows alleviated sharing of resources between users. Subsequently, a network can be virtualized as well, which encompasses creating virtual links, subnetworks, gateways and layer-2 bridges, etc. Since the server virtualization exists for an extensive period of time, numerous virtualization software is available. There are some major benefits the virtualization has brought into the world of computation management. With the improvement of availability, the servers are more user-friendly and available to supply bigger number of consumers efficiently. The users can create virtual machines and migrate the operations they perform in the form of images and run the same image in another environment. This inclines on the fact that virtualization also improves mobility, which is a very important factor. Another improvement is the improvement in the efficiency if exploitation of the hardware. A single virtual machine performs segmentation and is able to run a distinct operating system than the one at which the virtualization software is running. This allows the users to execute different software on different platforms, and at the same time distributing the resources of the physical machine more efficiently. Additionally, storage aggregation augments the global manageability of storage and delivers improved distribution of storage resources. At the same time, the capability of backup in the virtual environment is a big advantage. In case of failure, the servers can be configured to automatically migrate the data to another machine, without compromising the work they perform at the given moment, which in fact will also prevent data loss (VAEZI, Mojtaba and Zhang, Ying, 2017).

2.3.1. OpenStack cloud platform

Cloud computing has attracted considerable attention over the past few years. It offers the possibility to move an infrastructure to a platform where the requirement for hardware is no longer obligatory, but instead invest for uptime. With an interface that enables increasing and decreasing the number of virtual machines in a cloud, one builds a cluster that can adapt the number of servers to actual user demand, thereby both decreasing cost and evading saturated servers. A dedicated virtual machine (VM) model is not working when it comes to compute-intensive applications. Yet while considering Docker containers is a feasible idea, avoiding "noisy neighbor" problems that are common on shared infrastructure with SaaS offerings and performance problems for stateful applications like databases, is very desirable. OpenStack is a set of software tools for building and managing cloud computing platforms for public and private clouds. Backed by some of the biggest companies in software development and hosting, as well as thousands of individual community members, many regard OpenStack as the future of cloud computing. OpenStack is managed by the OpenStack Foundation, a non-profit organization that practices both development and community-building around the project. OpenStack allows users to deploy virtual machines and other instances that handle different tasks for managing a cloud environment, continuously. The horizontal scaling is eased, which means that tasks that benefit from running concurrently can easily serve more or fewer users simultaneously by just running up more instances. For example, a mobile application that needs to communicate with a remote server might be able to divide the work of communicating with each user across many different instances, all communicating with one another but scaling quickly and easily as the application gains more users. And most importantly, it is open source software, which means that anyone who chooses to can access the source code, make any changes or modifications they need, and freely share the changes back out to the community at large. It also means that OpenStack has the benefit of thousands of developers all over the world working in tandem to develop the strongest, most robust, and most secure product that they can (OPENSTACK, 2017).

The cloud provides computing for end users in a remote environment, where the actual software runs as a service on reliable and scalable servers rather than on each end-user's computer. Cloud computing can refer to a lot of different entities, but typically the industry discusses about running different items "as a service" - software, platforms, and infrastructure. OpenStack falls into the latter category and is considered Infrastructure as a Service (IaaS). Providing infrastructure means that OpenStack makes it easy for users to quickly add new instance, upon which other cloud components can run. Typically, the infrastructure then runs a "platform" upon which a developer can create software applications that are delivered to the end users. OpenStack is comprised of many different dynamic parts. Because of its open nature, anyone can add additional components to help it meet the demands. One of the advantages that OpenStack brings is that it helps prevent vendor lock-in to the underlying software and hardware. This is made possible by managing the resources through OpenStack, instead of using the vendor's part directly. This means that a vendor's component can potentially be replaced with another vendor's easily. The drawback of this approach is that OpenStack only supports common required features for all kinds of supported modules and may miss some features specific to a vendor's constituents. On the other hand, it should not go unnoticed that, due to the lack of an accepted standard for cloud platforms, using OpenStack implies a type of lock-in to OpenStack itself, with no guarantee of portability to a different cloud framework (OPENSTACK, 2017).

However, disregarding the type of cloud infrastructure employed, there are several repercussions that need to be addressed when it comes to implementation of the future 5G networks. Since the main goals of 5G are to improve capacity, reliability and energy efficiency, while reducing latency and massively increasing connection density; a crucial part of 5G is the empowerment of real-time application support. The given applications such as self-driving cars, robotics, medical appliances or online-gaming, require as lower network latency as possible. In the case of the present cloud technologies, the orientation towards latency minimization is instead diverted to providing service reliability and robustness. The focus of next generation mobile communication is to provide seamless communication for machines and devices building the Internet-of-Things (IoT) along with personal communication. New applications such as tactile Internet, high-resolution video streaming, tele-medicine, tele-surgery, smart transportation, and real-time control dictate new specifications for throughput, reliability, end-to-end (E2E) latency, and network robustness. Additionally, intermittent or always-on type connectivity is required for machine-type communication (MTC) serving diverse applications including sensing and monitoring, autonomous cars, smart homes, moving robots and manufacturing industries. Several emerging technologies including wearable devices, virtual/augmented reality, and full immersive experience (3D) are shaping the demeanor of human end users, and they have special requirements for user satisfaction. Therefore, these use cases of the next generation network push the specifications of 5G in multiple aspects such as data rate, latency, reliability, device/network energy efficiency, traffic volume density, mobility, and connection density. Current fourth generation (4G) networks are not capable of fulfilling all the technical requirements for these services (PARVEZ, I. et al., 2017).

One secret behind the manipulation with latency and service reliability is the situating of the core network and the way it is accessed by the eNB. Although, there are major advancements in the radio-access entity, coding, modulation and access techniques, the main goal of this thesis is to allude on the importance of the concepts of cloud computing merged with Software-Defined Networking and virtualizing of a Network Function. Consequently, the current cloud infrastructures are clustered in existing datacenters, where the virtualized core network can be deployed. Analogously, the eNB processing should reside on the premises of the core network, which can directly impact the performance at which UE is accessing the NAS

(Non-Access Stratum). This concept is known as Virtualized cloud Radio Access Network (C-RAN). A C-RAN over passive optical network (PON) architecture is introduced called virtualized-CRAN (V-CRAN), which can dynamically associate any radio unit (RU) to any digital unit (DU) so that several RUs can be coordinated by the same DU, and the concept of virtualized BS (V-BS) that can jointly transmit common signals from multiple RUs to a user. This concept of splitting the core network (CN) into multiple entities allow for greater granular control and flexibility of computation resources placement and scaling (PARVEZ, I. et al., 2017). Given that the splits are deployed in a distributed cloud, the computational units for the eNB should be executed in the vicinity of the cloudified mobile core network. This will allow a direct communication between the eNB and the core network, specifically referred to as *Edge computing* (FARRIS, I. et al., 2017, pp.1-13).

2.4. Multi-platform containers and their role in service deployment and software-defined networking

To allow the deployment of splits and granular control over the core network constituents as well as the virtualized eNB application, a virtualization concept of containers is necessary to be apprehended. The old way of deploying applications was to install the applications on a host using the operating system package manager. This had the disadvantage of entangling the applications' executables, configuration, libraries, and lifecycles with each other and with the host OS. One could build immutable virtual-machine images to achieve predictable rollouts and rollbacks, but VMs are heavyweight and non-portable. The new way is to deploy containers based on operating-system-level virtualization rather than hardware virtualization. These containers are isolated from each other and from the host: they have their own filesystems, they cannot see each other's' processes, and their computational resource usage can be bounded. They are easier to build than VMs, and because they are decoupled from the underlying infrastructure and from the host filesystem, they are portable across clouds and OS distributions. Because containers are small and fast, one application can be packed in each container image. This one-to-one application-to-image relationship unlocks the full benefits of containers. With containers, immutable container images can be created at build/release time rather than deployment time, since each application doesn't need to be composed with the rest of the application stack, nor married to the production infrastructure environment. Generating container images at build/release time enables a consistent environment to be carried from development into production. Similarly, containers are vastly more transparent than VMs, which facilitates monitoring and management. This is especially true when the containers' process lifecycles are managed by the infrastructure rather than hidden by a process supervisor inside the container. Finally, with a single application per container, managing the containers becomes tantamount to managing deployment of the application (KUBERNETES, 2017). Containers, among virtual machines, are the prevalent entities used for establishing the mobile network. Specifically, as an open-source solution, Docker is being tested and used in conjunction with other automation tools such as Kubernetes and Docker-compose. It is shown that with using Docker containers, even the performance of high-performance computing platforms (HPC) remain uncompromised while the portability is being preserved. This is mostly due to the instantaneous access to faster library imports and access to the OS kernel (HALE, J. S. et al., 2017, pp.40-50).

2.4.1. Docker

The tool Docker is a very powerful open-source tool for software containerization. Docker introduces containers that can wrap a piece of software in a complete filesystem which contains everything needed to run: code, runtime, system tools, and system libraries - anything that can be installed on a server. This guarantees that the software will always be executed in the exact form, disregarding the environment (see Figure 47).

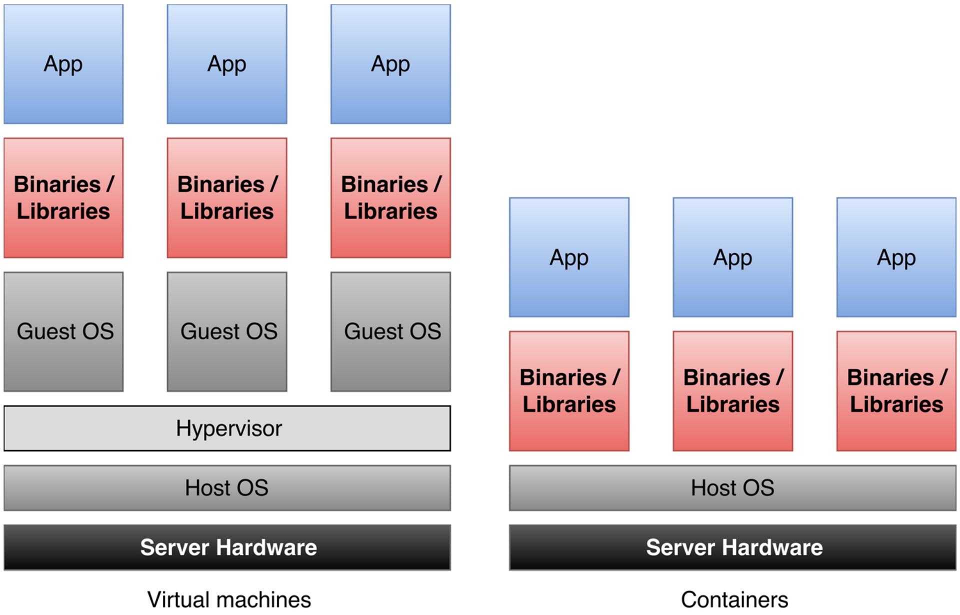


Figure 47. Containers and virtual machines have similar resource isolation and allocation benefits, but a different architectural approach allows containers to be more portable and efficient

Containers running on a single machine share the same OS kernel; they start instantly and use less RAM. Images are constructed from layered filesystems and share common files, making disk usage and image downloads much more efficient. A Linux container is a virtualization instance in which the kernel of an operating system enables multiple isolated user-space instances in a Linux operating system. Also, Docker supports containerization on a Windows operating system. One can build and run Windows-based containers, or also run previously-created containers with a Linux image base, on a Windows OS or even MAC (DOCKER, INC., 2017).

Unlike virtual machines (VMs), containers do not need to run a complete operating system (OS) image for each instance. Instead, containers are able to run separate instances of an application within a single shared OS. This new feature gives developers the flexibility to build once and move applications without the need to rewrite or redeploy their code, which makes up for faster integration and access to analytics, big data and services. As noted from Figure 47, virtual machines include the application, the necessary binaries and libraries, and an entire guest operating system, all of which can amount to tens of GBs. On the contrary, containers include the application and all its dependencies - but share the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud (DOCKER, INC., 2017). Docker offers a wide palette of features, tools and plugins to expand the functionality and ease the management of deployed containers. Some of the more important appliances are described as follows:

A. Docker-Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, a YAML file is used to configure the application's services. Then, with a single command, all the services are created and started from the particular configuration. It is useful when an application is dissected into multiple containers and deployed on a host. Compose has traditionally been focused on development and testing workflows, which is the basis for Continuous-Integration/Continuous Delivery (CI/CD) paradigms (DOCKER, INC., 2018).

B. Docker Cloud

The Docker Cloud provides a hosted registry service with build and testing facilities for Dockerized application images; tools to aid the set up and management of host infrastructure; and application lifecycle features to automate deploying (and redeploying) services created from images. The Docker Cloud is used as a registry in the project for saving the built container images of the mobile network infrastructure, under the repository *"brunodzogovic"* for the current project (DOCKER, 2018) [repository: https://hub.docker.com/u/brunodzogovic/]. Docker Cloud uses the hosted Docker Cloud Registry, which allows publishing Dockerized images on the internet either publicly or privately. Docker Cloud can also store pre-built images, or link to a source code so it can build the code into Docker images, and optionally test the resulting images before pushing them to a repository. Before anything is performed with the images, they need a place to be initially run. Docker Cloud allows linking to the existing infrastructure or cloud services provider, so the new nodes can be automatically provisioned. Once the nodes are set up, one can deploy images directly from Docker Cloud repositories (DOCKER, INC., 2018).

C. Docker Hub

The registry service which allows linking code to repositories is called Docker Hub. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline (DOCKER, INC., 2018). A "Docker ID" is created (*"brunodzogovic"* in this case) [repository: https://hub.docker.com/u/brunodzogovic/], which allows setting up a repository to push the built images. Besides repositories, the Docker Hub provides also automated builds and automated creation of images when changes are made to the source code in the repository, webhooks, workgroups and integration with GitHub and Bitbucket workflows (DOCKER, INC., 2018).

D. Docker networking

The most important concept of Docker is connecting containers together in a fully-operational network stacks. The networking in Docker is modular and pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

- **Bridge:** The default network driver. If a driver is not specified, this is the type of network that the Docker daemon will create. Bridge networks are usually used when applications run in standalone containers that need to communicate.

- **Host:** For standalone containers, one can remove network isolation between the container and the Docker host, and use the host's networking directly. The "host" driver is only available for swarm services on Docker 17.06 and higher.

- **Overlay:** Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other, for example with VXLAN. Overlay networks are also used to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers.

- **MACVLAN:** MACVLAN networks allow assigning a MAC address to a container, making it appear as a physical device on the existing network. The Docker daemon routes traffic to containers by their MAC addresses. Using the MACVLAN driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack.

- **IPVLAN:** The IPVLAN driver offers a very similar functionality as the MACVLAN driver, with one major exception of using L3 multiplexing/demultiplexing among slave nodes. This property makes the master device share the L2 with its slave devices. IPVLAN has two modes of operation - L2 and L3. For a given master device, one can select one of these two modes and all slaves on that master will operate in the same (selected) mode. The RX mode is almost identical except that in L3 mode the slaves will not receive any multicast / broadcast traffic. L3 mode is more restrictive since routing is controlled from the other (mostly) default namespace.

- **None:** For this container, disable all networking. Usually used in conjunction with a custom network driver. 'None' is not available for swarm services.
- **Network plugins:** It is feasible to install and use third-party network plugins with Docker, such as Calico or Open vSwitch. These plugins are available from the Docker Store or from third-party vendors.

E. Docker storage

It is possible to store data within the writable layer of a container, but there are some downsides: The data doesn't persist when that container is no longer running, and it can be difficult to extract the data out of the container if another process needs it. A container's writable layer is tightly coupled to the host machine where the container is running. It is thus not simple to move the data elsewhere. Writing into a container's writable layer requires a storage driver to manage the filesystem. The storage driver provides a union filesystem, using the Linux kernel. This extra abstraction reduces performance as compared to using data volumes, which write directly to the host filesystem (DOCKER, INC., 2018).

Docker offers three different ways to mount data into a container from the Docker host*: volumes, bind mounts, or tmpfs volumes*. Usually, volumes are almost always the right choice. Volumes are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker. Bind mounts may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time. Tmpfs mounts are stored in the host system's memory only and are never written to the host system's filesystem (DOCKER, INC., 2018). In order to solve the storage persistence issue, a third-party solution may be implemented, such as etcd (COREOS, 2018).

2.4.2. Etcd

Etcd stands for *"/etc distributed"* key value store, that is an open-source distributed key value store which provides shared configuration and service discovery for Linux Container clusters. Etcd runs on each machine in a cluster and gracefully handles leader election during network partitions and the loss of the current leader. Application containers running on a cluster can read and write data into etcd. Common examples are storing database connection details, cache settings, feature flags, etc. Etcd is the essence on top of which Kubernetes is built, and therefore, Kubernetes leverages the etcd distributed key-value store. It takes care of storing and replicating data used by Kubernetes across the entire cluster, and thanks to the Raft consensus algorithm (ONGARO, D. and Ousterhout, J., 2014), etcd can recover from hardware failure and network partitions (COREOS, 2018).

A. Layer-4 Etcd gateway

Etcd L4 gateway is a simple TCP proxy that forwards network data to the etcd cluster. The gateway is stateless and transparent; it neither inspects client requests nor interferes with cluster responses. The gateway supports multiple etcd server endpoints and works on a simple round-robin policy. It only routes to available endpoints and hides failures from its clients. Other retry policies, such as weighted round-robin, are supported. Every application that accesses etcd must first have the address of an etcd cluster client endpoint. If multiple applications on the same server access the same etcd cluster, every application still needs to know the advertised client endpoints of the etcd cluster. If the etcd cluster is reconfigured to have different endpoints, every application may also need to update its endpoint list. This wide-scale reconfiguration is both tedious and error prone. Etcd gateway solves this problem by serving as a stable local endpoint. A typical etcd gateway configuration has each machine running a gateway listening on a local address and every etcd application connecting to its local gateway. The consequence is only that the gateway needs to update its endpoints instead of updating each and every application. In summary, to automatically propagate cluster endpoint changes, the etcd gateway runs on every machine serving multiple applications accessing the same etcd cluster (COREOS, 2018).

Clusters are usually built from a large collection of machines with the ability to run any workload at any given time. In order for a cluster to perform at high levels of efficiency, the workloads should be distributed appropriately across all machines in the cluster. Then clusters need a way of coordinating with each other. For example, a job scheduler needs to notify a machine that it has work to do. Once that work has been completed machines may need to communicate that fact to some other component in the cluster. A distributed system needs a reliable coordination mechanism, and therefore, it is important that this communication happens in an apt and reliable manner to keep everything running efficiently. Essentially, something has to manage the state of the cluster, which is the actual etcd that is matter of discussion (COREOS, 2018).

B. Role-based access control (RBAC)

Etcd has its own authentication features and a role-based access control. Etcd defines one special user *root* and one special role *root*. The root user, which has full access to etcd, must be created before activating authentication. The idea behind the root user is for administrative purposes: managing roles and ordinary users. The root user must have the root role and is allowed to change anything inside etcd. The role root may be granted to any user, in addition to the root user. A user with the root role has both global read-write access and permission to update the cluster's authentication configuration. Furthermore, the root role grants privileges for general cluster maintenance, including modifying cluster membership, defragmenting the store, and taking snapshots. If an etcd server is launched with the option *--client-cert-auth=true*, the field of Common Name (CN) in the client's TLS certificate will be used as an etcd user. In this case, the common name authenticates the user and the client does not need a password.

2.4.3. Kubernetes

One of the orchestrating solutions that are used is Kubernetes (KUBERNETES, 2017). Kubernetes is a powerful system, developed by Google, for managing containerized applications in a clustered environment. It aims to provide better ways of managing related, distributed components across varied infrastructure. Kubernetes, at its basic level, is a system for managing containerized applications across a cluster of nodes. In many ways, Kubernetes was designed to address the disconnection between the way that modern, clustered infrastructure is designed, and some of the assumptions that most applications and services have about their environments. Most clustering technologies strive to provide a uniform platform for application deployment. The user should not have to care much about where work is scheduled. The unit of work presented to the user is at the "service" level and can be accomplished by any of the member nodes. However, in many cases, it does matter what the underlying infrastructure looks like. When scaling an application, the administrator cares that the various instances of a service are not all being assigned to the same host. On the other hand, many distributed applications built with scaling in mind are comprised of smaller component services. These services must be scheduled on the same host as related components if they are going to be configured in a trivial way. This becomes even more important when they rely on specific networking conditions to communicate appropriately. While it is possible with most clustering software to make these types of scheduling decisions, running at the level of individual services is not perfect. Applications comprised of different services should still be managed as a single application in most cases. Kubernetes provides a layer over the infrastructure to allow for this type of management. Even though Kubernetes provides a lot of functionality, there are always new scenarios that would benefit from new features. Application-specific workflows can be streamlined to accelerate developer velocity. Ad hoc orchestration that is acceptable initially often requires robust automation at scale. This is why Kubernetes was also designed to serve as a platform for building an ecosystem of components and tools to make it easier to deploy, scale, and manage applications. Labels empower users to organize their resources however they please. Annotations enable users to decorate resources with custom information to facilitate their workflows and provide an easy way for management tools to checkpoint state. Additionally, the Kubernetes control plane is built upon the same APIs that are available to developers and users. Users can write their own controllers, such as schedulers, with their own APIs that can be targeted by a general-purpose command-line tool. This design has enabled a number of other systems to build atop Kubernetes (KUBERNETES, 2017).

Main features of Kubernetes are:

- **Self-healing** - Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- **Horizontal scaling** – Scaling an application up and down with a simple command, with a UI, or automatically based on CPU usage. This feature can be further upgraded for automatic scaling based on metrics taken from the network, CPU/Disk utilization and other metrics.
- **Automatic bin-packing** - Automatically places containers based on their resource requirements and other constraints, while not sacrificing availability. This stands for mixing critical and best-effort workloads to drive up usage and save even more resources.
- **Automated rollouts and rollbacks** - Kubernetes progressively rolls out changes to the deployed application or its configuration, while monitoring its health to ensure that it doesn't kill all the other instances at the same time. If unpredictable events take place, Kubernetes will roll back the change automatically.
- **Service discovery and load balancing** – Is a very important feature that annuls the need to modify a running application to use an unfamiliar service discovery mechanism. Kubernetes gives containers their own IP addresses and a single DNS name for a set of containers and can load-balance across them.
- **Secret and configuration management** – An excellent security property that allows deploying and updating secrets and application configuration without rebuilding the image and without exposing secrets in the stack configuration.
- **Storage orchestration** - Automatically mount the storage system by choice, whether from local storage, a public cloud provider such as GCP or AWS, or a network storage system such as NFS, iSCSI, Gluster, Ceph, Cinder, or Flocker.
- **Batch execution** - In addition to services, Kubernetes can manage batch and CI workloads, replacing containers that fail, if desired.

A. Master components:

Infrastructure-level systems like CoreOS strive to create a uniform environment where each host is disposable and interchangeable. Kubernetes, on the other hand, operates with a certain level of host specialization. The controlling services in a Kubernetes cluster are called the master, or control plane, components. These operate as the main management contact points for administrators and provide many cluster-wide systems for the relatively dumb worker nodes. These services can be installed on a single machine or distributed across multiple machines. The servers running these components have a number of unique services that are used to manage the cluster's workload and direct communications across the system (KUBERNETES, 2017).

- **Etcd** - Kubernetes uses etcd to store configuration data that can be used by each of the nodes in the cluster. This can be used for service discovery and represents the state of the cluster that each component can reference to configure or reconfigure themselves. By providing a simple HTTP/JSON API, the interface for setting or retrieving values is very straightforward.
- **API Server** - This is the main management point of the entire cluster, as it allows a user to configure many of Kubernetes' workloads and organizational units. It also is responsible for making sure that the etcd store and the service details of deployed containers are in agreement. It acts as the bridge between various components to maintain cluster health and disseminate information and commands. • Controller Manager Service - The controller manager service is a general service that has many responsibilities. It is responsible for a number of controllers that regulate the state of the cluster and perform routine tasks. For instance, the replication controller ensures that the number of replicas defined for a service matches the number currently deployed on the cluster. The details of these operations are written to etcd, where the controller manager watches for changes through the API server.
- **Scheduler Service** - The process that assigns workloads to specific nodes in the cluster is the scheduler. This is used to read in a service's operating requirements, analyze the current infrastructure environment, and place the work on an acceptable node or nodes. The scheduler is responsible for tracking resource utilization on each host to make sure that workloads are not scheduled in excess of the available resources. The scheduler must know the total resources available on each server, as well as the resources allocated to existing workloads assigned on each server.
- **Kube-controller manager** - Runs controllers, which are the background threads that handle routine tasks in the cluster. Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process. These controllers include:
    - Node Controller: Responsible for noticing and responding when nodes go down.
    - Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
    - Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods).
    - Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces.
- **Cloud-controller manager** - runs controllers that interact with the underlying cloud providers. The cloud-controller-manager binary is an alpha feature introduced in Kubernetes release 1.6. Cloud-controller-manager runs cloud-provider-specific controller loops only. One must disable these controller loops in the kube-controller-manager. It is possible to disable the controller loops by setting the **--cloud-provider** flag to **external** when starting the kube-controller-manager. Cloud-controller-manager allows cloud vendors code and the Kubernetes core to evolve independent of each other. In prior releases, the core Kubernetes code was dependent upon cloud-provider-specific code for functionality. In future releases, code specific to cloud vendors should be maintained by the cloud vendor themselves and linked to cloud-controller-manager while running Kubernetes. The following controllers have cloud provider dependencies:
    - Node Controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
    - Route Controller: For setting up routes in the underlying cloud infrastructure
    - Service Controller: For creating, updating and deleting cloud provider load balancers

  - Volume Controller: For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes
- **Addons** – Represent pods and services that implement cluster features. The pods may be managed by Deployments, ReplicationControllers, and so on. “Namespaced” addon objects are created in the **kube-system** namespace. Addon manager creates and maintains addon resources.
- **DNS -** While the other addons are not strictly required, all Kubernetes clusters should have cluster DNS, as many examples rely on it. Cluster DNS is a DNS server, in addition to the other DNS server(s) in the given environment, which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.
- **Web UI (Dashboard) –** Is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.
- **Container resource monitoring – R**ecords generic time-series metrics about containers in a central database and provides a UI for browsing that data.
- **Cluster-level logging –** A mechanism that handles saving container logs to a central log store with search/browsing interface.

*B. **Node Server Components:***

In Kubernetes, servers that perform work are known as nodes. Node servers have a few requirements that are necessary to communicate with the master components, configure the networking for containers, and run the actual workloads assigned to them (KUBERNETES, 2017).

- **Docker running on a dedicated subnet** - The first requirement of each individual node server is docker. The docker service is used to run encapsulated application containers in a relatively isolated but lightweight operating environment. Each unit of work is, at its basic level, implemented as a series container that must be deployed. One key assumption that Kubernetes makes is that a dedicated subnet is available to each node server. This is not the case with many standard clustered deployments. For instance, with CoreOS, a separate networking fabric called flannel is needed for this purpose. Docker must be configured to use this so that it can expose ports in the correct fashion. Also, Calico can be used in conjunction with Flannel for network overlay, which is supported in Kubernetes as a plugin called “Canal”.
- **Kubelet Service** - The main contact point for each node with the cluster group is through a small service called kubelet. This service is responsible for relaying information to and from the control plane services, as well as interacting with the etcd store to read configuration details or write new values. The kubelet service communicates with the master components to receive commands and work. Work is received in the form of a "manifest" which defines the workload and the operating parameters. The kubelet process then assumes responsibility for maintaining the state of the work on the node server.
- **Proxy Service -** To deal with individual host subnetting and to make services available to external parties, a small proxy service is run on each node server. This process forwards requests to the correct containers, can do primitive load balancing, and is generally responsible for making sure the networking environment is predictable and accessible, but isolated.
- **Rkt** – Is an experimental container platform, supported as an alternative to Docker.
- **Supervisord** – A lightweight process monitor and control system that can be used to keep kubelet and Docker running.
- **Fluentd** – A daemon that helps provide cluster-level logging.

C. *Kubernetes Work Units:*

While containers are the used to deploy applications, the workloads that define each type of work are specific to Kubernetes. We will go over the different types of "work" that can be assigned below (KUBERNETES, 2017):

- **Pods -** A pod is the basic unit that Kubernetes deals with. Containers themselves are not assigned to hosts. Instead, closely related containers are grouped together in a pod. A pod generally represents one or more containers that should be controlled as a single "application". This association leads all the involved containers to be scheduled on the same host. They are managed as a unit and they share an environment. This means that they can share volumes and IP space, and can be deployed and scaled as a single application. One can and should generally think of pods as a single virtual computer to best conceptualize how the resources and scheduling should work. The general design of pods usually consists of the main container that satisfies the general purpose of the pod, and optionally some helper containers that facilitate related tasks. These are programs that benefit from being run and managed in their own container but are heavily tied to the main application. Horizontal scaling is generally discouraged on the pod level because there are other units more suited for the task.
- **Services** - We have been using the term "service" throughout this guide in a very loose fashion, but Kubernetes actually has a very specific definition for the word when describing work units. A service, when described this way, is a unit that acts as a basic load balancer and ambassador for other containers. A service groups together logical collection of pods that perform the same function to present them as a single entity. This allows deployment of a service unit that is aware of all of the backend containers to pass traffic to. External applications only need to worry about a single access point but benefit from a scalable backend or at least a backend that can be swapped out when necessary. A service's IP address remains stable, abstracting any changes to the pod IP addresses that can happen as nodes die or pods are rescheduled. Services are an interface to a group of containers so that consumers do not have to worry about anything beyond a single access location. By deploying a service, one easily gains discover-ability and can simplify the container designs.

D. *Controller units:*

- **Replication Controllers -** A more complex version of a pod is a replicated pod. These are handled by a type of work unit known as a replication controller. A replication controller is a framework for defining pods that are meant to be horizontally scaled. Essentially, the work unit is a nested unit. A template is provided, which is basically a complete pod definition. This is wrapped with additional details about the replication work that should be done. The replication controller is delegated responsibility over maintaining a desired number of copies. This means that if a container temporarily goes down, the replication controller might start up another container. If the first container comes back online, the controller will kill off one of the containers. • Labels - A Kubernetes organizational concept outside of the work-based units is labeling. A label is basically an arbitrary tag that can be placed on the above work units to mark them as a part of a group. These can then be selected for management purposes and action targeting. Labels are fundamental to how both services and replication controllers function. To get a list of backend servers that a service should pass traffic to, it usually selects containers based on label.
- **Replica sets -** ReplicaSet is the next-generation Replication Controller. The only difference between a ReplicaSet and a Replication Controller at this point is the selector support. ReplicaSet supports the new set-based selector requirements, whereas a Replication Controller only supports equality-based selector requirements. A ReplicaSet ensures that a specified number of pod replicas

are running at any given time. However, a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to pods along with a lot of other useful features. Therefore, it is recommended to utilize Deployments instead of directly using ReplicaSets, unless a particular custom update orchestration is required, or no updates are required at all.

- **Deployments -** A Deployment controller provides declarative updates for Pods and ReplicaSets. A desired state is described in a Deployment object, and the Deployment controller changes the actual state to the desired state at a controlled rate. One can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.
- **Stateful sets -** Manages the deployment and scaling of a set of Pods and provides guarantees about the ordering and uniqueness of these Pods. Like a Deployment, a StatefulSet manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods. These pods are created from the same spec but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling. A StatefulSet operates under the same pattern as any other Controller. One defines the desired state in a StatefulSet object, and the StatefulSet controller makes any necessary updates to achieve that from the current state.
- **Daemon sets -** A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created. Some typical uses of a DaemonSet are: running a cluster storage daemon, such as glusterd, ceph, on each node; running a logs collection daemon on every node, such as fluentd or logstash; running a node monitoring daemon on every node, such as Prometheus Node Exporter, collectd, Datadog agent, New Relic agent, or Ganglia gmond. In a simple case, one DaemonSet, covering all nodes, would be used for each type of daemon. A more complex setup might use multiple DaemonSets for a single type of daemon, but with different flags and/or different memory and cpu requests for different hardware types.
- **Garbage collection -** The role of the Kubernetes garbage collector is to delete certain objects that once had an owner, but no longer have an owner. Some Kubernetes objects are owners of other objects. For example, a ReplicaSet is the owner of a set of Pods.
- **Jobs -** A *job* creates one or more pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the *job* tracks the successful completions. When a specified number of successful completions is reached, the job itself is complete. Deleting a Job will clean up the pods it created. A simple case is to create one Job object to reliably run one Pod to completion. The Job object will start a new Pod if the first pod fails or is removed (for example due to a node hardware failure or a node reboot). A Job can also be used to run multiple pods in parallel.
- **Cron jobs -** One Cron job object is like one line of a crontab (cron table) file. It runs a job periodically on a given schedule, written in Cron format.

E. Namespaces:

Kubernetes introduces namespaces as a concept of organizing the components of the management in a single namespace (as in Figure 48). Namespace represent an isolated area that can contain pods, replication controllers and services that are isolated within it. This allow having services and pods with the same names without experiencing conflicts between the namespaces on a same physical Kubernetes cluster (Figure 48) (KUBERNETES, 2017).

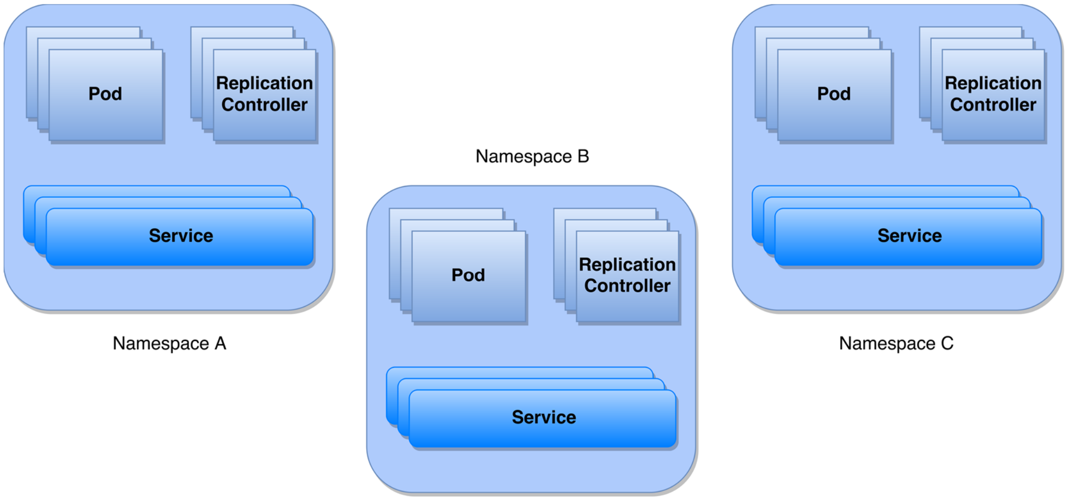


Figure 48. Kubernetes namespaces

2.4.4. Orchestration of Docker containers with Kubernetes

Container Orchestration refers to the automated arrangement, coordination, and management of software containers. When the applications becomes dissected into splits, it encompasses an architecture known as microservices (DIGITALOCEAN, 2018). A single-container deployment refers to deploying an application in one container image. However, if the specific application (or infrastructure) has multiple functional splits, it is possible to dissect the deployment into multiple containers. The formed containers can be easily organized into clusters of containers and deployed then remotely using orchestrator such as Kubernetes (KUBERNETES, 2017).

For example, to deploy an application in a Kubernetes Pod, it is necessary to create a YAML file (see Appendix A). The execution of the file will create a replicated database cluster, based on SQL and in much replicas as designated into the YAML configuration (in this case 3). To create the particular pod, a single command only is required: *kubectl create –f db-pod.yml,* where the *db-pod.yml* is the name of the YAML file. To deploy any other application, the same method can be adopted. The *kubectl* daemon will initialize all the necessary pods and run the specific Docker containers in them. The simplicity of using Kubernetes for cluster management of a microservices deployment is thus unprecedented. With conjunction of few open-source solutions mentioned in the following chapters and subchapters, Kubernetes can solve all the predicaments in terms of service discovery, load balancing, secrets/configuration/storage management, health checks, auto-scaling/restart/healing of containers and nodes and provide the desirable zero-downtime deployments.

2.4.5. Security of application containers, secret storage and managing secrets

It is known that the container technology may suffer from security predicaments as much as any other system. The image-oriented nature of containers represents a conundrum by itself. As it is the case with any software, containerized deployments can suffer of vulnerabilities. For example, a Docker container can have a backdoor installed and if it is available for implementation from a public repository (i.e. Docker Hub), it is very easy to deploy an insecure application without having a notion about it. One solution is image scanning for vulnerabilities, which is usually manual process or an Enterprise feature that is adopted by companies such as Docker, IBM, Google, etc. (TAK, B. et al., 2017).

To resolve the security issue on a network and application level, a multi-layer security approach is adopted (network-level security with Pfsense firewall (PFSENSE, 2018) and Cisco ACL, container-level security with policy-based networking and secret storage, as well as application-level security with provided authentication using SIM cards). Since the containers need to communicate between each other, exchange tokens, security-related data, encrypted passwords or database access; a policy-based approach is defined via SDN (Software-Defined Networking) entities, explained later. Moreover, since the mobile network utilizes the DIAMETER authentication protocol, a secret management system is required. For that purpose, the open-source tool vault is employed, which is a tool for securely accessing secrets. A secret is anything that a specific access control is preferred for, such as API keys, passwords, or certificates. Vault provides a unified interface to any secret, while providing tight access control and recording a detailed audit log. A modern system requires access to a multitude of secrets: database credentials, API keys for external services, credentials for service-oriented architecture communication, etc. Understanding who is accessing what secret is already very difficult and platform-specific. Adding on key rolling, secure storage, and detailed audit logs is almost impossible without a custom solution, which is the actual main function of Vault (HASHICORP, 2018).

The key features of Vault are:

- **Secure Secret Storage**: Arbitrary key/value secrets can be stored in Vault. Vault encrypts these secrets prior to writing them to persistent storage, so gaining access to the raw storage isn't enough to access the given secrets. Vault can write to disk to Consul etc.
- **Dynamic Secrets**: Vault can generate secrets on-demand for some systems, such as AWS or SQL databases. For example, when an application needs to access an S3 bucket, it asks Vault for credentials, and Vault will generate an AWS keypair with valid permissions on demand. After creating these dynamic secrets, Vault will also automatically revoke them after the lease is up.
- **Data Encryption**: Vault can encrypt and decrypt data without storing it. This allows security teams to define encryption parameters and developers to store encrypted data in a location such as SQL without having to design their own encryption methods.
- **Leasing and Renewal**: All secrets in Vault have a *lease* associated with them. At the end of the lease, Vault will automatically revoke that secret. Clients are able to renew leases via built-in renew APIs.
- **Revocation**: Vault has built-in support for secret revocation. Vault can revoke not only single secrets, but a tree of secrets, for example all secrets read by a specific user, or all secrets of a particular type. Revocation assists in key rolling as well as locking down systems in the case of an intrusion.

Vault is not tied to any specific configuration management system. One can read secrets from configuration management, but it is also possible to use the API directly to read secrets from applications. This means that configuration management requires fewer secrets, and in many cases doesn’t ever have to persist the secrets to disk. Vault encrypts the data onto physical storage and requires multiple keys to read it. If an attacker were to gain access to the physical encrypted storage, it could not be read without multiple keys, which are generally distributed to multiple individuals. This is known as *unsealing* and happens once whenever the Vault service starts. For an unsealed Vault, every interaction is logged in via the audit devices. Even erroneous requests (invalid access tokens, for example) are logged. To access any data, an access token is required. This token is usually associated with an identity coming from a system such as GitHub, LDAP, etc. This identity is also written to the audit log. Access tokens can be given fine-grained control over what secrets can be accessed. It is rare to have a single key that can access all secrets. This makes it easier to have fine-grained access for consumers of Vault. In addition to being able to store secrets, Vault can be used to encrypt/decrypt data that is stored elsewhere. The primary use of this is to allow applications to encrypt their data while still storing it in the primary data store (HASHICORP, 2018).

2.4.6. Automation with Puppet and Terraform (Infrastructure-as-a-Code)

In order to establish a platform for automation, it is essential to introduce some state-of-the-art DevOps practices of the modern system administration paradigm. Many open-source technologies exist, that are able to accommodate any kind of infrastructures, while establishing a solid base for automation of deployment and configuration management. Some of the solutions that are represented thus are: Puppet (PUPPET, 2018), Terraform (HASHICORP, 2018) and earlier, Kubernetes (KUBERNETES, 2017) as an orchestrator.

Configuration management tools install and manage software on a machine that already exists. Terraform is not a configuration management tool, and it allows existing tooling to focus on their strengths: bootstrapping and initializing resources. Using provisioners, Terraform enables any configuration management tool (such as Puppet) to be used to setup a resource once it has been created. Terraform focuses on the higher-level abstraction of the datacenter and associated services, without sacrificing the ability to use configuration management tools to do what they do best. It also embraces the same codification that is responsible for the success of those tools, making entire infrastructure deployments easy and reliable (HASHICORP, 2018). Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions. Configuration files describe to Terraform the components needed to run a single application or an entire datacenter. Terraform generates an execution plan describing what it will perform in order to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied. The infrastructure Terraform can manage includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc. Complex changesets can be applied to the infrastructure with minimal human interaction. With the previously mentioned execution plan and resource graph, it is possible to know exactly what Terraform will change and in what order, avoiding many possible human errors.

In the particular research, Software Defined Networking (SDN) is the prevalent area of operation, as it provides more control and allows the network to better support the applications running on top. Most SDN implementations have a control layer and infrastructure layer. Terraform can be used to codify the configuration for software defined networks. This configuration can then be used by Terraform to automatically setup and modify settings by interfacing with the control layer. This allows configuration to be versioned and changes to be automated. As an example, AWS VPC is one of the most commonly used SDN implementations and can be configured by Terraform (HASHICORP, 2018).

2.5. The role of NFV and SDN in the evolution towards 5G

Usually, the concepts of Software-Defined Network (SDN) and virtual Network Function (vNF) are being misunderstood and mistaken for each other. However, the definitions for both terms are acutely different. Software-Defined Networking is *"The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices"* (OPENNETWORKFOUNDATION , 2018), whereas Network functions virtualization (NFV) is *"the concept of replacing dedicated network appliances - such as routers and firewalls - with software running on commercial off-the-shelf (COTS) servers"* (ADVA OPTICAL NETWORKING, 2018). The both paradigms have a similar common goal, which is to transform the way communication service providers (CSPs) architect networks and deliver network services. Network operations are transformed as network function software, that is dynamically instantiated in various locations in the network as needed, without requiring the installation of new equipment.

As stated previously, the Long-Term Evolution (LTE) is constituted of two entities: UMTS-RAN and EPC, namely, the 3G Universal Mobile Telecommunication Service - Radio Access Network and the Evolved Packet Core. The EPC contains the HSS (Home Subscriber Server), which is in fact a database for the users; the PGW (Packet Data Gateway) that routes the data communication via IP network and interfaces SGW (Service Gateway) with the Internet. The SGW is responsible for routing the traffic of calls and other services from the eNB to the PGW, which is dictated by the MME (Mobile Management Entity). The dedicated hardware for the LTE network serves the purpose of the functionality it is dedicated for. Adding additional hardware in the core network adds to the complexity of the system, as well as the cost. When it comes to accommodation of larger number of users, the peak times are handled by duplicating the entities. Such process will lead to incremental complexity of the core network and linear increase of costs for providing the adequate hardware equipment. As the number of users is predicted to increase with the emergence of the next-generation networks, the hardware-based 4G LTE equipment is no longer apposite to correspond to the requirements, because the expenditures will remain constant while the revenues per subscriber will gradually decrease. This introduces the necessity for the Software-Defined Networking (SDN) paradigm and Network Function Virtualization (NFV). The proposed methodologies offer various scenario improvements, while enabling combinations that are usually not feasible with the existing hardware only. In fact, moving the EPC to the cloud helps achieve cost reduction for the benefit of enlarging revenues margin (TAWBEH, Ali et al., 2017).

A crucial requirement for telecom network infrastructure is the compatibility with cloud or computing architecture as a flexible and cost-effective service platform. With the fact that the NFV and SDN are enabling the hardware equipment emulation into the cloud, technical issues from a networking standpoint emerge. The desirable outcome for introducing peculiar 5G network slices, should encompass automated and scalable management of cloud-based NFV infrastructure; as well as the possibility for improvement of the particular performance of the current infrastructure, in terms of latency, throughput and reasonable applicability of the model parallelly to the given scenario.

2.5.1. Mobility meets virtualization

In a 5G world, the capacity and latency are the most critical units that need to be taken into consideration. One of the appliances that the technology will be based on, is the Distributed Cloud. A Distributed Cloud represents arranging a data center at the edge domain, like central office or a base station. All services that are included in this domain are taking a virtual form, which will enable ease of access, reduction of latency, exponential reduction of hardware cost for implementation etc. Taking all these instruments into consideration will contemplate a logical need for orchestration machinery as well as analytics and monitoring solutions that are of frivolous manner. Evidently, this machinery shall be comprised of location and even personality-based AI objects, that will automatically evaluate, monitor, troubleshoot, organize and manage the infrastructure. This network intelligence is a complex assortment of Software Defined Networking (SDN), Network Virtualization Function (NFV), Artificial Intelligence and machine learning combined with immutable infrastructure (UDDENFELDT, Jan, 2017).

To establish these modules, a precise automation method is required. Immutable infrastructure provides stability, efficiency, and fidelity to applications through automation and the use of successful patterns from programming. No rigorous or standardized definition of immutable infrastructure exists yet, but the basic conception is that one creates and operates an infrastructure using the programming concept of immutability: once something is instantiated, it is never changed. Instead, it is replaced with another instance to make changes or ensure proper behavior. Immutable Infrastructure builds on processes from the nature and how it maintains advanced biological systems (STELLA, Josh, 2015). The primary mechanism of fidelity in humans is the constant destruction and replacement of subcomponents. It triggers the immune system, which destroys cells to maintain health and it motivates the growth system, which allows different subsystems to mature over time through destruction and replacement. The individual human being maintains a sense of self and intention, while the underlying components are constantly replaced. Systems managed using II patterns are analogous (BERNSTEIN, Ben, 2015). The reimbursements of immutable infrastructure are manifold if applied appropriately to an application and have effusively automated deployment and recovery methods for any infrastructure:

- **Simplifying operations.** With fully-automated deployment methods, it is possible to replace old components with new versions to guarantee that systems are never far in time from their initial “known-good” state. Maintaining a fleet of instances becomes considerably simpler with II since there is no need to track the changes that occur with mutable maintenance methods.

- **Continuous deployments, fewer failures.** With II, it is known what is running and how it behaves, deploying updates can become mundane and continuous, with fewer failures transpiring in production. All change is tracked by the source control and Continuous Integration/Continuous Deployment processes.

- **Reduces errors and threats.** Services are built atop a complex stack of hardware and software, and events usually take wrong occurrence over time. By automating replacement instead of maintaining instances, instances are regularly and repeatedly regenerated. This reduces configuration drift, vulnerability surface, and level of effort to keep Service Level Agreements. Many of the maintenance fire drills in mutable systems are taken care of naturally.

- **Complete cloud rebooting.** With Immutable Infrastructure, the running components are familiar, and with fully automated recovery methods for the services in place, cloud reboots of the underlying instances should be handled gracefully and with minimal, if any, application interruption.

The concept of immutable infrastructure is an emerging IT strategy enabled using Docker and containers (BRYZEK, Michael, 2014). Docker can empower the 5G networking components to behave just like the organs in the human body, where a single malfunctioning organ can be replaced with adjacent one from a donor that has a corresponding genetic sequence at the 6th chromosome (the Human Leukocyte Antigen). The advanced idea that is researched is beyond the replacement of the organs, where an intelligent system will decide to automatically perform the replacement of the modules, adjusting network performance, create a temporary solution for a peculiar problem. For example, one of the main modules deducted for the successful 5G operation is the Air Interface that is the SDN / NFV module of the 5G network, to facilitate successful service delivery to the end users (INFLUXDATA, 2017). The Air Interface can be easily incorporated into containers, and furthermore in form of a microservice architecture (that opens even additional potentials). From this point, the prospects are interminable. At a soccer match, tens of thousands of viewers can record the event with presumably HD or even 4K imaging devices. Due to the popularity of the social networks, many of the experiences tend to be shared with the acquaintances. The only possible way is using the mobile infrastructure at that point. A current LTE network supports [real] 10s of Mbit/s traffic speed, which will allow the user to send the video content on Facebook or upload on YouTube. Simultaneous uploads from most of the viewers will bottleneck the neighboring base stations with the GBs of content intended for sharing. In a 5G scenario, using the immutable infrastructure, the Air Interface containers can be delivered to a Distributed Cloud in the vicinity of the soccer match. This action can be performed automatically using genetic algorithms for prediction and identification of bigger demands for the network, where the NFV module will receive instructions to integrate itself within the Distributed Cloud and replicate to enable load-balancing and high availability for other services that require resources at the same moment (interconnected train sensors, self-driving car sensors, IoT devices, wearable gadgets etc.). This way, the end-to-end service delivery would be uninterrupted, perfected, while lowest possible latency is ensured, altogether approximately less than 1ms. This can scale up to a situation where surgeons can perform remote surgeries over long distances using automated hardware and robotics, that will get as much resources as required due to priority, all in the same area where the soccer match is taking place.

2.5.2. SDN and NFV solutions, network overlay and underlay

To successfully connect remote workloads, a networking solution that can manage L3 and L2 operations is required, namely routing and switching. The actual devices that can perform these operations are hardware routers and switches. For the particular requirements of the mobile network infrastructure, the hardware usage feasibility is limited for a simple reason, which is virtualization of the networking function. Specifically, a physical router cannot be virtualized without usage of a software-defined network. For the deployment of the mobile network and connecting the eNB base station to the virtualized Evolved Packet Core, the most appropriated SDN options are selected and consequently described. Each of these technologies have their own distinct features that can contribute to establishing a secure, trustable and fault-tolerant next-generation virtualized mobile infrastructure.

A. Calico

Calico provides secure network connectivity for containers and virtual machine workloads. It [calico] creates and manages a flat layer 3 network, assigning each workload a fully routable IP address. Workloads can communicate without IP encapsulation or network address translation for bare metal performance, easier troubleshooting, and better interoperability. In environments that require an overlay, Calico uses IP-in-IP tunneling or can work with other overlay networking such as Flannel. Calico also provides dynamic enforcement of network security rules. Using simple policy language, it is possible to achieve fine-grained control over communications between containers, virtual machine workloads, and bare metal host endpoints. Proven in production at scale, Calico features integrations with Kubernetes, OpenShift, Docker, Mesos, DC/OS, and OpenStack (TIGERA INC., 2017).

The Calico's control plane design is reflected upon the design of the Internet itself, which serves billions of endpoints around the world, and represents the largest network ever built. Scaling the cloud to millions of workloads should be easy, and therefore, Calico borrows proven IP routing technology to connect containers (and VMs) to one another and to underlying infrastructure. Accordingly, security policy rules are distributed with conventionality to cloud techniques pioneered by web-scale operators such as Google. Making use of the same raft consensus algorithm found in systems like Kubernetes, a consistent, fast convergence times are achieved (typically a few milliseconds, even at scale) with high levels of fault tolerance. Sometimes, an overlay network (encapsulating packets inside an extra IP header) is necessary. Often, though, it just adds unnecessary overhead, resulting in multiple layers of nested packets, impacting performance and complicating troubleshooting. It would thus be immensely desirable if the virtual networking solution adapts to the underlying infrastructure, using an overlay only when required. Subsequently, in most environments, Calico simply routes packets from the workload onto the underlying IP network without any extra headers. Where an overlay is needed – for example when crossing availability zone boundaries in public cloud – it can use lightweight encapsulation including IP-in-IP and VxLAN or MACVLAN. Project Calico even supports both IPv4 and IPv6 networks. Moreover, Calico can be integrated with all the major cloud platforms, including OpenStack's Neutron, AWS, GCE, orchestrators like Kubernetes, Mesos, Docker container solution serving as CNI (Container Networking Interface) etc. In terms of reliability, Calico is a widely-deployed SDN solution. For example, Japan empowers Calico with Kubernetes on top of OpenStack to deliver the Yahoo services with enterprise-grade performance and reliability (TIGERA INC., 2017).

Calico security principles

With Calico as a Docker network plugin, Calico uses an identically named profile to represent each Docker network. This profile is applied to each container in that network and the profile is used by Calico to configure access policy for that container. The Calico network plugin will automatically create the associated profile if it does not exist when the container is attached to the network. By default, the profile contains rules that allow full egress traffic but allow ingress traffic only from containers within the same network and no other source. Custom policy for a network can be configured by creating in advance, or editing, the profile associated with the Docker network (TIGERA INC., 2017).

There are two approaches by which the policy that defines the Docker network can be modified:

a) Modify the profile policy rules - This policy is applied directly to each container in the associated Docker network. This approach is simple, but not very flexible, as the profile must describe the full set of rules that apply to the containers in the network.

b) Assign labels to the profile and define global selector-based policy - The (Calico-specific) labels are assigned to containers in the associated Docker network. The globally defined policy uses selectors to determine which subset of the policy is applied to each container based on their labels. This approach provides a powerful way to group together all of the particular network Policy, makes it easy to reuse policy in different networks, and makes it easier to define policy that extends across different orchestration systems that use Calico.

Managing Calico policy for a network

In both cases a Calico-Docker network is created and the *"calicoctl"* tool is used to achieve the required isolation. The following example denotes an isolation between a set of database containers and frontend containers:

- Frontend containers can only access the Database containers over TCP to port 3306. At this point, it is assumed that no other connectivity is allowed to/from the frontend.
- Database containers have no isolation between themselves (to handle synchronization within a cluster). This could be improved by locking down the port ranges and protocols, but for brevity a full access between database containers is allowed.

a) Policy applied directly by the profile

In the following example the policy for containers is applied in both networks with using profiles. Each network has associated an identically named profile that consists of a set of labels and policy rules. The labels and policy rules are set for each of the two network profiles to provide the required isolation. Moreover, Docker networks are created by the following commands:

```
docker network create --driver calico --ipam-driver calico-ipam database
docker network create --driver calico --ipam-driver calico-ipam frontend
```

After the creation of the networks, the profiles are added for each of the networks. The labels are set on each profile indicating the network role, and in this case frontend or database. Each profile also includes a set of ingress and egress rules and actions, where each rule can filter packets based on a variety of source or destination attributes (which includes selector-based filtering using label selection). The labels and rules are applied directly to each container in the corresponding network. The labels themselves are arbitrary key/value pairs, and their current purpose at this point is to use the key role showing the network role and a value of either frontend or database. Correspondingly, a profile will have the following form in YAML structure (see Appendix D) (TIGERA INC., 2017).

The profiles provide the required isolation between the frontend and database containers. This works as follows:

- Containers in the "database" Docker network are assigned the "database" Calico profile.

- Containers in the “frontend” Docker network are assigned the “frontend” Calico profile.
- Each container in the “database” network inherits the label role = database from its profile.
- Each container in the “frontend” network inherits the label role = frontend from its profile.
- The “database” profile applies ingress and egress policy:
    - An ingress rule to allow TCP traffic to port 3306 from endpoints that have the label role = frontend (i.e. from frontend containers since they are the only ones with the label role = frontend)
    - An ingress and egress rule to allow all traffic from and to endpoints that have the label role = database (i.e. from database containers).
- The “frontend” profile applies a single egress rule to allow all TCP traffic to port 3306 on endpoints that have the label role = database (i.e. to database containers)

a) Global policy applied through label selection

The same example can be demonstrated using global selector-based policy. In this case, the network profiles are used to apply labels (as in the previous example), but additionally define a set of global policy resources that use selectors to determine which subset of the policy applies to each container based on the labels applied by the profile. After the creation of the Docker networks, the profiles for each of the networks are created accordingly (see Appendix E) (TIGERA INC., 2017).

The labels on each profile indicate the network role, and in this case frontend or database. The labels are applied directly to each container in the corresponding network. As with the previous example, the key role indicating the network role and a value of either frontend or database is used. Unlike the previous, no policy rules are defined within the profile. To enable the required network isolation, the global policy is created (policy resources are defined globally, and like profile includes a set of ingress and egress rules and actions, where each rule can filter packets based on a variety of source or destination attributes, which includes selector-based filtering using label selection). Each policy resource also has a “main” selector that is used to determine which endpoints the policy is applied to base on the labels applied by the network profiles (refer to Appendix F) (TIGERA INC., 2017).

Implementing Calico as a solution for securing host interfaces

It is feasible to use Calico for securing the host’s network interfaces (as opposed to those of any container/VM workloads that are present on the host). The host endpoints are distinguishable by workload endpoints by the role they play. The former are the physical host network endpoints, while the latter are referring to the ones of the virtual machines and containers. Calico supports the same rich security policy model for host endpoints that it supports for workload endpoints. Host endpoints can have labels, and their labels are in the same “namespace” as those of workload endpoints. This allows security rules for either type of endpoint to refer to the other type (or a mix of the two) using labels and selectors. Calico does not support setting IPs or policing MAC addresses for host interfaces, it assumes that the interfaces are configured by the underlying network fabric (this option is tested further on using VxLAN and MACVLAN for performance improvement and avoiding network overlay when possible). Calico distinguishes workload endpoints from host endpoints by a configurable prefix. Unless there is a host interfaces whose name matches the default for that prefix (cali), changing the same would not be required. In a positive case, it can be configured accordingly. Interfaces that start with a value listed in InterfacePrefix are assumed to be workload interfaces. Others are treated as host interfaces. Calico blocks all traffic to/from workload interfaces by default; allowing traffic only if the interface is known and policy is in place. However, for host endpoints, Calico is more lenient; it only polices traffic to/from interfaces that it’s been explicitly informed about. Traffic to/from other interfaces is neglected. As of Calico v2.1.0, Calico applies host endpoint security policy both to traffic that is terminated locally, and to traffic that is forwarded between host endpoints. Previously, policy was only applied to traffic that was terminated locally. The change allows Calico to be used to secure a NAT gateway or router. Calico supports selector-based policy as normal when running on a gateway or router allowing for rich, dynamic security policy based on the labels attached to the workloads, as represented in Figure 49 (TIGERA INC., 2017).

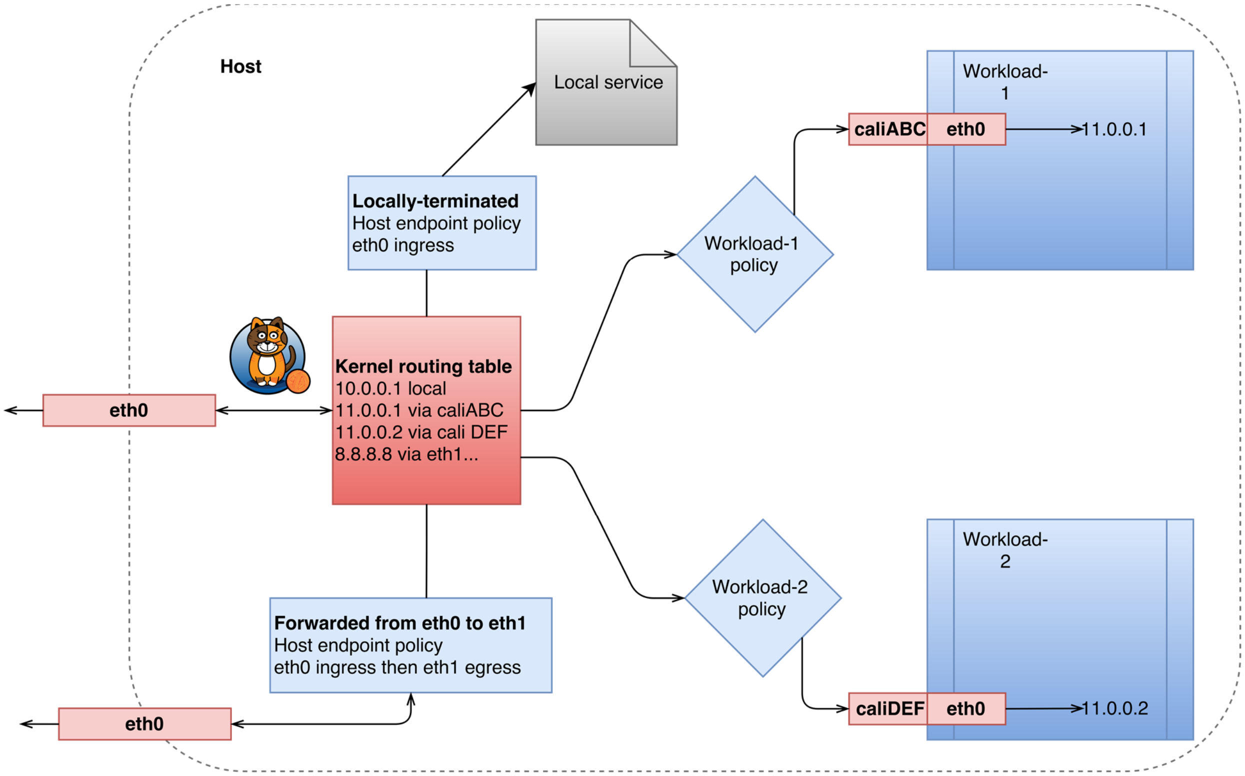


Figure 49. Organization of Calico security endpoints to protect physical hosts

B. Open vSwitch (OvS)

Open vSwitch (Figure 50) is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). In addition, it is designed to support distribution across multiple physical servers similar to VMware's vNetwork distributed vswitch or Cisco's Nexus 1000V. Open vSwitch can operate both as a soft switch running within the hypervisor, and as the control stack for switching silicon. It has been ported to multiple virtualization platforms and switching chipsets. It is the default switch in XenServer 6.0, the Xen Cloud Platform and also supports Xen, KVM, Proxmox VE and VirtualBox. It has also been integrated into many virtual management systems including OpenStack, openQRM, OpenNebula and oVirt. The kernel datapath is distributed with Linux, and packages are available for Ubuntu, Debian, Fedora and openSUSE. Open vSwitch is also supported on FreeBSD and NetBSD (OPENVSWITCH, 2018).

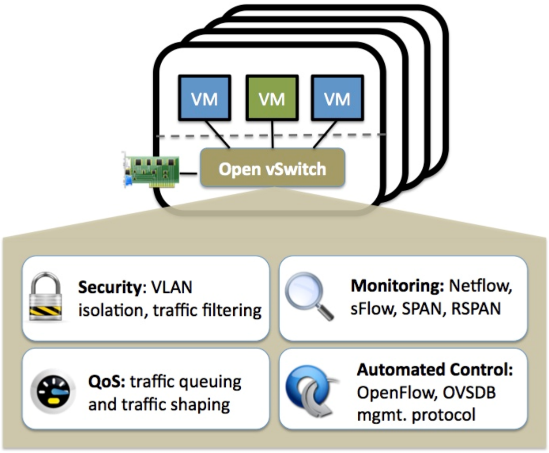

VM
VM
VM
Open vSwitch
Security: VLAN isolation, traffic filtering
Monitoring: Netflow, sFlow, SPAN, RSPAN
QoS: traffic queuing and traffic shaping
Automated Control: OpenFlow, OVSDB mgmt. protocol

Figure 50. Open vSwitch architecture (OPENVSWITCH, 2018)

For this thesis, the Open vSwitch will serve as a translator to the OpenStack cloud, where the GTP-U mobile network protocol should interwork with the IP layer-3 Neutron entity via L2TP tunnels. Consequently, Open vSwitch can enable container-to-container communication, disregarding the host that accommodates the containers. In union with Calico L3 routing with BGP, Open vSwitch will accommodate remote workloads from the EPC to the eNB, with full-duplex communication and very low latency.

2.6. Hardware for establishing a base station (software-defined radio)

The hardware used for the experiments is the Universal Software Radio Peripheral (USRP) that is a range of software-defined radios designed and sold by Ettus Research (ETTUS, 2018) and its parent company, National Instruments. Developed by a team led by Matt Ettus, the USRP product family is envisioned to be a comparatively inexpensive hardware platform for software radio, and is commonly used by research labs, universities, and hobbyists. Most USRPs connect to a host computer through a high-speed link, which the host-based software uses to control the USRP hardware and transmit/receive data. Some USRP models also integrate the general functionality of a host computer with an embedded processor that allows the USRP device to operate in a stand-alone fashion. The USRP family was designed for accessibility, and many of the products are open source hardware. The board schematics for select USRP models are freely available for download; all USRP products are controlled with the open source UHD driver, which is free and open source software. USRPs are commonly used with the GNU Radio software suite to create complex software-defined radio systems (ETTUS, 2018).

The USRP product family includes a variety of models that use a similar architecture. A motherboard provides the following subsystems: clock generation and synchronization, FPGA (Field-Programmable Gate Array), ADCs (Analog-to-Digital Converters), DACs (Digital-to-Analog Converters), host processor interface, and power regulation. These are the basic components that are required for baseband processing of signals. A modular front-end, called a daughterboard, is used for analog operations such as up/down-conversion, filtering, and another signal conditioning. This modularity permits the USRP to serve applications that operate between DC and 6 GHz. In stock configuration, the FPGA performs several DSP operations, which ultimately provide translation from real signals in the analog domain to lower-rate, complex, baseband signals in the digital domain. In most use-cases, these complex samples are transferred to/from applications running on a host processor, which perform DSP operations. The code for the FPGA is open-source and can be modified to allow high-speed, low-latency operations to occur in the FPGA. The USRP software defined radio products are designed for RF applications from DC to 6 GHz, including multiple antenna (MIMO) systems. Example application areas include white spaces, mobile phones, public safety, spectrum monitoring, radio networking, cognitive radio, satellite navigation, and amateur radio (ETTUS, 2018).

2.6.1. USRP N200 – Network series

The USRP N200 series provides high-bandwidth, high-dynamic range processing capability. The product architecture includes a Xilinx® Spartan® 3A-DSP 1800 FPGA, 100 MS/s dual ADC, 400 MS/s dual DAC and Gigabit Ethernet connectivity to stream data to host processors. A modular design allows the USRP N200 to operate from DC to 6 GHz. An expansion port allows multiple USRP N200 series devices to be synchronized and used in a MIMO configuration. An optional GPSDO module can also be used to discipline the USRP N200 reference clock to within 0.01 ppm of the worldwide GPS standard. The USRP N200 can stream up to 50 MS/s to and from host applications, and users can implement custom functions in the FPGA fabric, or in the on-board 32-bit RISC softcore. The FPGA offers the potential to process up to 100 MHz of RF bandwidth in both the transmit and receive directions. The FPGA firmware can be reloaded through the Gigabit Ethernet interface (ETTUS, 2018).

The USRP hardware driver (UHD) is the device driver provided by Ettus Research for use with the USRP product family. It supports Linux, MacOS, and Windows platforms. Several frameworks including GNU Radio, LabVIEW, MATLAB and Simulink use UHD. The functionality provided by UHD can also be accessed directly with the UHD API, which provides native support for C++. Any other language that can import C++ functions can also use UHD. This is accomplished in Python through SWIG, for example. UHD provides portability across the USRP product family. Applications developed for a specific USRP model will support other USRP models if proper consideration is given to sample rates and other parameters (ETTUS, 2018).

Several software frameworks support UHD:

- GNU Radio as a Free/Libre toolkit that can be used to develop software-defined radios. This framework uses a combination of C++ and Python to optimize DSP performance while providing an easy-to-use application programming environment. GNU Radio Companion is a graphical programming environment provided with GNU Radio.
- National Instruments NI USRP 292x series, which is functionally equivalent to the Ettus Research USRP N210. NI also offers LabVIEW support for this device with the NI-USRP Driver (NATIONALINSTRUMENTS, 2018).
- USRP N210 and USRP2 are supported by MATLAB and Simulink. This package includes plug-ins and several examples for use with both the devices.
- Many users develop with their own, custom frameworks. In this case, the USRP device can be accessed with the UHD API. There are also examples provided with UHD that show how to use the API (ETTUS, 2018) [USRP hardware driver and manual].

2.6.2. USRP B200/B210 – Bus series

The USRP B210 provides a fully integrated, single-board, Universal Software Radio Peripheral (USRP™) platform with continuous frequency coverage from 70 MHz – 6 GHz. Designed for low-cost experimentation, it combines the AD9361 RFIC direct-conversion transceiver providing up to 56MHz of real-time bandwidth, an open and reprogrammable Spartan6 FPGA, and fast SuperSpeed USB 3.0 connectivity with convenient bus-power. Full support for the USRP Hardware Driver™ (UHD) software allows developing with GNU Radio, prototype a custom GSM base station with OpenBTS, and seamless transition code from the USRP B210 to higher performance, industry-ready USRP platforms. An enclosure accessory kit is available to users of green PCB devices (revision 6 or later) to assemble a protective steel case. Experiments with the USRP B210 across a wide range of applications include: FM and TV broadcast, cellular, GPS, WiFi, ISM, and more. Users can immediately begin prototyping in GNURadio and participate in the open-source SDR community. Full support by the UHD software allows seamless code reuse from existing designs, compatibility with open-source applications like HDSDR and OpenBTS, and an upgrade path to industry-ready USRP systems to meet application requirements (ETTUS, 2018).

The integrated RF frontend on the USRP B210 is designed with the new Analog Devices AD9361, a single-chip direct-conversion transceiver, capable of streaming up to 56 MHz of real-time RF bandwidth. The B210 uses both signal chains of the AD9361, providing coherent MIMO capability. Onboard signal processing and control of the AD9361 is performed by a Spartan6 XC6SLX150 FPGA connected to a host PC using SuperSpeed USB 3.0. The USRP B210 real time throughput is benchmarked at 61.44MS/s quadrature, providing the full 56 MHz of instantaneous RF bandwidth to the host PC for additional processing using GNU Radio or applications that use the UHD API (ETTUS, 2018).

B200 differs from B210 by only the fact that it doesn't support full-duplex MIMO operation (4 antennas) but only 2, from which one is used for Rx and the other for Tx.

3. DESCRIPTION OF OPEN AIR INTERFACE

OpenAirInterface5G is a project developed by EURECOM, a French graduate school and a research center in communication systems based in the international science park of Sophia Antipolis within the new Campus SophiaTech, which brings together renowned universities such as TELECOM ParisTech and other European Universities such as the Polytechnic University of Turin, Aalto University (formerly Helsinki University of Technology), Munich University of Technology and the Norwegian University of Science and Technology. EURECOM benefits from a strong interaction with the industry through its specific administrative structure: Economic Interest Group (consortium type of structure), which brings together international companies such as: Orange, ST Microelectronics, BMW Group Research & Technology, Symantec, Monaco Telecom, SAP, IABG. The Principality of Monaco is an institutional member which joined the consortium at the beginning of 2013. Thanks to its strong ties set up with the industry, EURECOM was awarded the "Institut Carnot" label jointly with the Institut Telecom in 2006. The Carnot Label was designed to develop and professionalize cooperative research. It encourages the implementation of research projects in public research centers that work together with socioeconomic actors, especially companies (EURECOM, 2017).

3.1. OpenAirInterface5G as EURECOM project and its aims

The OpenAirInterface5G^{TM} Software Alliance (OSA) is a non-profit consortium fostering a community of industrial as well as academic contributors for open source software and hardware development for the core network (EPC), access network and user equipment (EUTRAN) of 3GPP cellular networks. The Alliance sponsors the initial work of EURECOM to create OpenAirInterface5G^{TM} towards development of 5G Cellular Stack on Commercial Off-The-Shelf (COTS) hardware. The current generation of hardware/software for radio access network (RAN) consist of large numbers of proprietary elements that stifle innovation and increase the cost for the operators to deploy new services/application in an ever-changing fast paced cellular network. Open source software running on general purpose processors (such as x86, ARM) can greatly simplify network access, reduce cost, increase flexibility, improve innovation speed and accelerate time-to-market for introduction of new services. There is already a movement going on within the industry on the development of Software Defined Networking (SDN) concepts to open the proprietary interfaces to control the RAN hardware/software. At the same time, open-source has made a very significant impact in the extremities of current networks, namely in the terminals due to the Android ecosystem and in cloud infrastructure due, in part, to the OpenStack ecosystem (OAI, 2017).

An open source implementation of fully real-time stack (eNB, UE and core network) on general purpose processors when combined with SDN, Network Function Virtualization (NFV) and OpenStack and bring significant efficiency in RAN design from both innovation and cost perspective. OSA currently provides a standard-compliant implementation of a subset of Release 10 LTE for UE, eNB, MME, HSS, SGw and PGw on standard Linux-based computing equipment (Intel x86 PC/ARM architectures). The software is freely distributed by the Alliance under the terms stipulated by the OSA license model. It can be used in conjunction with standard RF laboratory equipment available in many labs (i.e. National Instruments/Ettus USRP and PXIe platforms) in addition to custom RF hardware provided by EURECOM to implement these functions to a sufficient degree to allow for real-time interoperation with commercial devices. Some industrial users have already been working on OpenAirInterface5G^{TM} (OAI)-based systems integrated with commercially-deployable remote radio-head equipment and have provided demonstrations at major industrial tradeshows (Mobile World Congress Asia 2014, Mobile World Congress Barcelona in 2013, IMIC 2013). The primary future objective is to provide an open-source reference implementation which follows the 3GPP standardization process starting from Rel-13 and the evolutionary path towards 5G and that is freely-available for experimentation on commodity laboratory equipment (OAI, 2017).

3.2. Architecture of OpenAirInterface5G

OpenAirInterface5G^{TM} (OAI) wireless technology platform is a flexible solution towards an open LTE ecosystem. The peculiar platform offers an open-source software-based implementation of the LTE system spanning the full protocol stack of 3GPP standard both in E-UTRAN and EPC. It can be used to build and customize a LTE base station (OAI eNB), a user equipment (OAI UE) and a core network (OAI EPC) on a PC. The OAI eNB can be connected either to a commercial UEs or OAI UEs to test different configurations and network setups and monitor the network and mobile device in real-time. In addition, OAI UE can be connected to eNB test equipment (CMW500) and some trials have been successively run with commercial eNB in December 2016 (OAI, 2017).

OAI is based on a PC hosted software radio frontend architecture. With OAI, the transceiver functionality is realized via a software radio front end connected to a host computer for processing. OAI is written in standard C for several real-time Linux variants optimized for Intel^{TM} x86 and ARM^{TM} processors and released as free software under the OAI License Model. OAI provides a rich development environment with a range of built-in tools such as highly realistic emulation modes, soft monitoring and debugging tools, protocol analyzer, performance profiler, and configurable logging system for all layers and channels (OAI, 2017).

When the matter of speech is building an open cellular ecosystem for supple and low-cost 4G/5G deployment and researches, OAI objects at the following points (OAI, 2017):

- Open and integrated development environment under the control of the experimenters;
- On the network side: Fully software-based network functions offering flexibility to architect, instantiate, and reconfigure the network components (at the edge, core, or cloud using the same or different addressing space);
- On UE side: Fully software-based UE functions which can be used by modem designers with upgrading and/or developing LTE and 5G advanced features
- Playground for commercial handsets as well as application, service, and content providers;
- Rapid prototyping of 3GPP compliant and non-compliant use-cases as well as new concepts towards 5G systems ranging from M2M/IoT and software-defined networking to cloud-RAN and massive MIMO.

Currently, the OAI platform includes a full software implementation of 4th generation mobile cellular systems compliant with 3GPP LTE standards in C under real-time Linux optimized for x86. At the Physical layer, it provides the following features (OAI, 2017):

- LTE release 8.6 compliant, with a subset of release 10;
- FDD and TDD configurations in 5, 10, and 20 MHz bandwidth;
- Transmission mode: 1 (SISO), and 2, 4, 5, and 6 (MIMO 2×2);
- CQI/PMI reporting;
- All DL channels are supported: PSS, SSS, PBCH, PCFICH, PHICH, PDCCH, PDSCH, PMCH;
- All UL channels are supported: PRACH, PUSCH, PUCCH, SRS, DRS;
- HARQ support (UL and DL);
- Highly optimized base band processing (including turbo decoder). With AVX2 optimization, a full software solution would fit with an average of 1×86 core per eNB instance (64QAM in downlink, 16QAM in uplink, 20MHz, SISO).

For the E-UTRAN protocol stack, it provides (OAI, 2017):

- LTE release 8.6 compliant and a subset of release 10 features;

- Implements the MAC, RLC, PDCP and RRC layers;
- protocol service for all Rel8 Channels and Rel10 eMBMS (MCH, MCCH, MTCH);
- Channel-aware proportional fair scheduling;
- Fully reconfigurable protocol stack;
- Integrity check and encryption using the AES and Sonw3G algorithms;
- Support of RRC measurement with measurement gap;
- Standard S1AP and GTP-U interfaces to the Core Network;
- IPv4 and IPv6 support.

Evolved packet core network features (OAI, 2017):

- MME, SGW, PGW and HSS implementations. OAI reuses standards compliant stacks of GTPv1u and GTPv2c application protocols from the open-source software implementation of EPC called nwEPC;
- NAS integrity and encryption using the AES and Snow3G algorithms;
- UE procedures handling: attach, authentication, service access, radio bearer establishment;
- Transparent access to the IP network (no external Serving Gateway nor PDN Gateway are necessary). Configurable access point name, IP range, DNS and E-RAB QoS;
- IPv4 and IPv6 support.

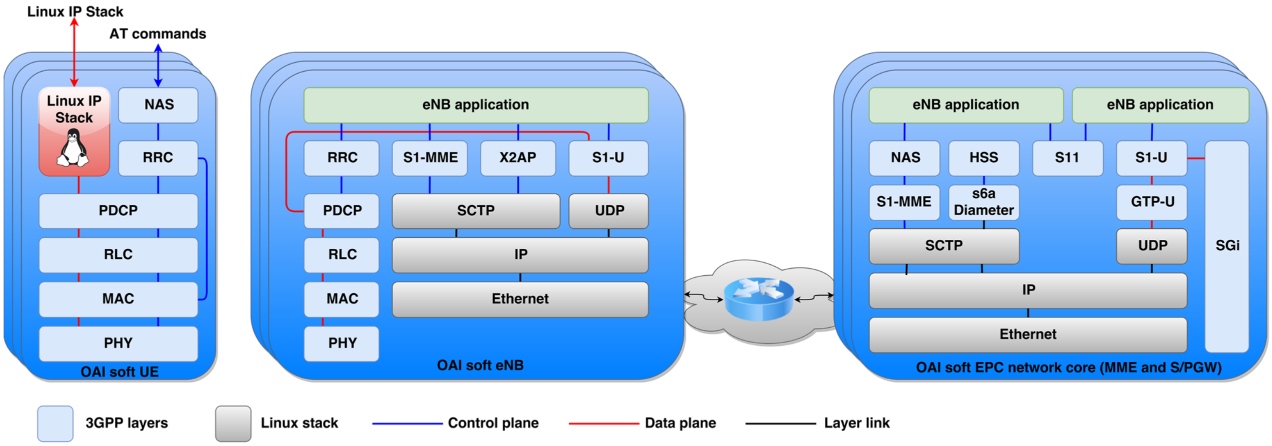


Figure 51. Architecture of OpenAirInterface5G

Figure 51 shows a schematic of the implemented LTE protocol stack in OAI. OAI can be used in the context of a rich software development environment including Aeroflex-Geisler LEON / GRLIB, RTOS either RTAI or RT-PREEMPT, Linux, GNU, Wireshark, control and monitoring tools, message and time analyser, low level logging system, traffic generator, profiling tools and soft scope. It also provides tools for protocol validation, performance evaluation and pre-deployment system test. Several interoperability tests have been successfully performed (OAI, 2017):

- OAI eNB with the commercial LTE-enabled mobile devices, namely Huawei E392, E398u-1, Bandrich 500 as well as with commercial 3rd party EPC prototypes.
- OAI-UE with test equipment CMW500 and commercial enodeB (Ericsson on com4Innov network) with commercial EPC.

OAI platform can be used in several different configurations involving commercial components to varying degrees (OAI, 2017):

- Commercial UE ↔ Commercial eNB + OAI EPC
- Commercial UE ↔ OAI eNB + Commercial EPC
- Commercial UE ↔ OAI eNB + OAI EPC
- OAI UE ↔ OAI eNB + OAI EPC
- OAI UE ↔ OAI eNB + Commercial EPC
- OAI UE ↔ Commercial eNB + Commercial EPC

3.2.1. Built-in emulation platform

Apart from real-time operation of the software modem on the hardware targets described above, the full protocol stack can be run in a controlled laboratory environment for realistic system validation and performance evaluation (see Figure 52). The platform is designed to represent the behavior of the wireless access technology in a real network setting, while obeying the temporal frame parameters of the air-interface. The platform targets large-scale repeatable experimentation in a controlled laboratory environment with various realistic test-cases and can be used for integration, performance evaluation and testing (OAI, 2017).

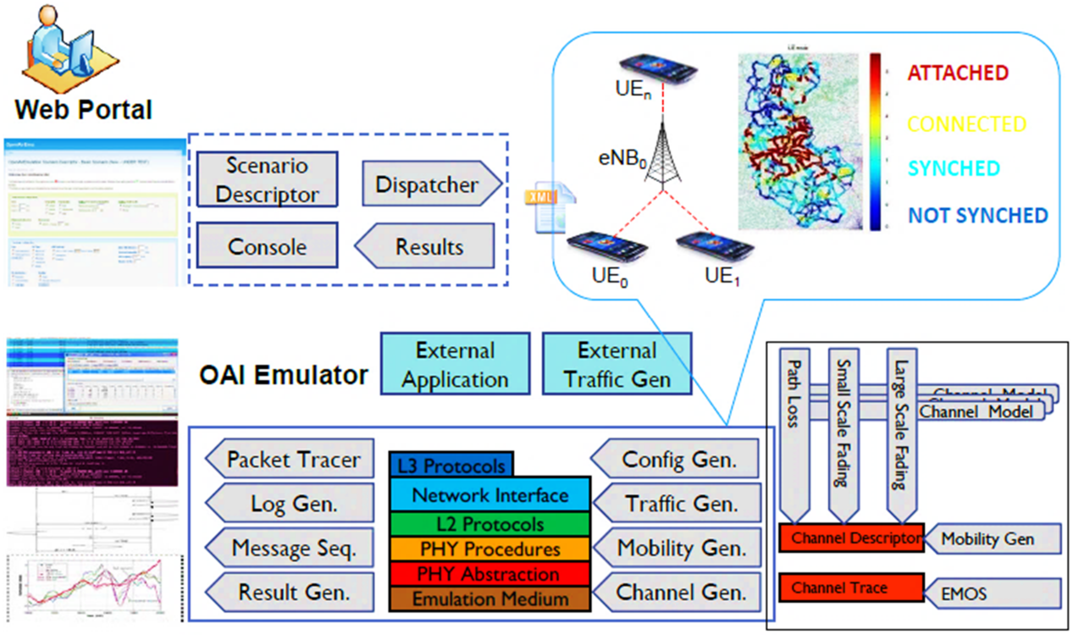


Figure 52. Emulation platform of the OpenAirInterface5G (OAI, 2017)

3.3. OpenAirInterface5G as an open-source solution is a driver towards 5G

OpenAirInterface5G (OAI) Software Alliance broadly focuses on the evolution of 3GPP Cellular stack (eNB + UE + Core Network) on general purpose processor architectures (Intel/ARM) with the goal of establishing generic interfaces with 3rd party RF platforms like EURECOM Express MIMO, National Instruments/Ettus Research USRP, Nuand BladeRF, SoDeRa Lime SDR platforms. The alliance also ensures that several projects conducted within the framework of the alliance are capable of running on Commercial-Off-The-Shelf (COTS) hardware platforms, for example Intel x86 and ARM. The Alliance engages itself in projects that enhances the core software (eNB/UE and Core Network) with the goal of running it across several platforms, while at the same time evolving towards future 3GPP standards. Since, the evolution of 5G is still under discussion within academia/industry, we plan to identify different areas in which different members of the alliance create projects. The projects are created with the goal of furthering these strategic areas within the alliance and making sure the output of the project is merged back with the main repository at some point (OPENAIRINTERFACE, 2018).

4. DESCRIPTION OF THE ESTABLISHMENT OF THE MOBILE NETWORK

The mobile network resides in the domain of a distinct Cisco 2800 router inside the OsloMet network 158.36.118.0/23 (see Appendix B), which can be accessed through SSH (a FQDN: *brunos-gw.hioa.no*). The particular firmware version of the router only allows usage of Diffie-Hellman key exchange procedure, and thus the SSH command has to include *-oKexAlgorithms=+diffie-hellman-group1-sha1 bruno@158.36.118.16* flag, in order to successfully log in to the router. To access the other OsloMet network 128.39.120.0/23, the datacenters are connected through direct fiber link. The OpenStack cloud of the OsloMet University is located at the address 128.39.120.13, where the master node can be accessed for the management of the OpenStack cloud named "Alto", and therefore, it is reachable from the Internet (*cloud.cs.hioa.no*).

The router redistributes existing routes from the 158.36.118.0/23 network into private segments of 192.168.10.0/24 for VLAN10 and 192.168.20.0/24 for VLAN20, respectively. The OSPF routing protocol is used and the FastEthernet0/0 interface is initialized as a relaying point to the external network. Since the option *"redistribute connected subnets"* and *"redistribute static subnets"* is being used, the Cisco router assigns itself as a distinct AS (Autonomous System) with its own routing area, where the existing subnetted OsloMet subnets are redistributed in the inside NAT, beyond the router. At the FastEthernet0/0 interface on the router, the option "IP Virtual Reassembly" (VFR) is disabled. This feature adds additional security layer for the mobile network that is situated in the NAT domain. Accordingly, the VFR option requires modification of the IP packets, and therefore, later the value of the MTU is modified at each physical and virtual interface, which can interfere with the operation of the mobile network. Later, the VFR security layer will be implemented to protect the mobile networks, after the adequate TCP/SCTP packet size is defined. VFR is responsible for detecting and preventing the following types of fragment attacks:

- *Tiny Fragment Attack* - In this type of attack, the attacker makes the fragment size small enough to force Layer 4 (TCP and User Datagram Protocol (UDP)) header fields into the second fragment. Thus, the ACL rules that have been configured for those fields will not match. VFR drops all tiny fragments, and an alert message such as follows is logged to the syslog server: "VFR-3-TINY_FRAGMENTS."
- *Overlapping Fragment Attack* - In this type of attack, the attacker can overwrite the fragment offset in the noninitial IP fragment packets. When the firewall reassembles the IP fragments, it might create wrong IP packets, causing the memory to overflow or the system to crash. VFR drops all fragments within a fragment chain if an overlap fragment is detected, and an alert message such as follows is logged to the syslog server: "VFR-3-OVERLAP_FRAGMENT."
- *Buffer Overflow Attack* - In this type of denial-of-service (DoS) attack, the attacker can continuously send a large number of incomplete IP fragments, causing the firewall to lose time and memory while trying to reassemble the fake packets. To avoid buffer overflow and control memory usage, a maximum threshold for the number of IP datagrams that are being reassembled and the number of fragments per datagram is configured. (Both of these parameters can be specified via the IP virtual-reassembly command.) When the maximum number of datagrams that can be reassembled at any given time is reached, all subsequent fragments are dropped, and an alert message such as the following is logged to the syslog server: "VFR-4_FRAG_TABLE_OVERFLOW." When the maximum number of fragments per datagram is reached, subsequent fragments will be dropped, and an alert message such as the following is logged to the syslog server: "VFR-4_TOO_MANY_FRAGMENTS." In addition to configuring the maximum threshold values, each IP datagram is associated with a managed timer. If the IP

datagram does not receive all of the fragments within the specified time, the timer will expire and the IP datagram (and all of its fragments) will be dropped.

As shown in Figure 53, the router defines two dot1Q encapsulation VLAN segments at two different virtual interfaces. The first interface is FastEthernet0/1.10 and the second FastEthernet0/1.20, for VLAN10 and VLAN20, respectively. The addressing space is assigned appropriately for the both interfaces, namely: 192.168.10.0/24 and 192.168.20.0/24, accordingly. The access to the subnets is regulated with access-list command that allows connection to the particular subnets.

From this point, the router is connected to a Cisco 2960 switch, that defines the VLANs and ports for the particular hosts that are situated in the local networks (see Appendix C). The switch defines two VLANs, namely 10 and 20. The VLAN10 is subnetted for the usage of the OpenAirInterface5G project and hosts a networked USRP N210 device with IP address 192.168.10.2 on the GigabitEthernet0/1 port, under the virtual interface FastEthernet0/1.10 from the router, as a default gateway to the Virtual LAN. Analogously, for the VLAN20, the separate USRP N210 device is connected to the GigabitEthernet0/2 port and shares the address space of the other hosts that reside in the 192.168.20.0/24 subnet.

The first port from the switch serves as a trunk port, that carries all the traffic from both VLANs to the router, has an IP address 192.168.0.2, and belongs to the routing domain of the router's physical interface 192.168.0.1. The same port from the switch serves as a management port and can be accessed from the LAN through telnet. The initial Fast-Ethernet switch will be replaced with HP ProCurve 2910al-24G switch (HEWLETT-PACKARD, 2018), which is a switch that is comprised of 24 gigabit ports and 4 SFP fiber ports for increased data rates.

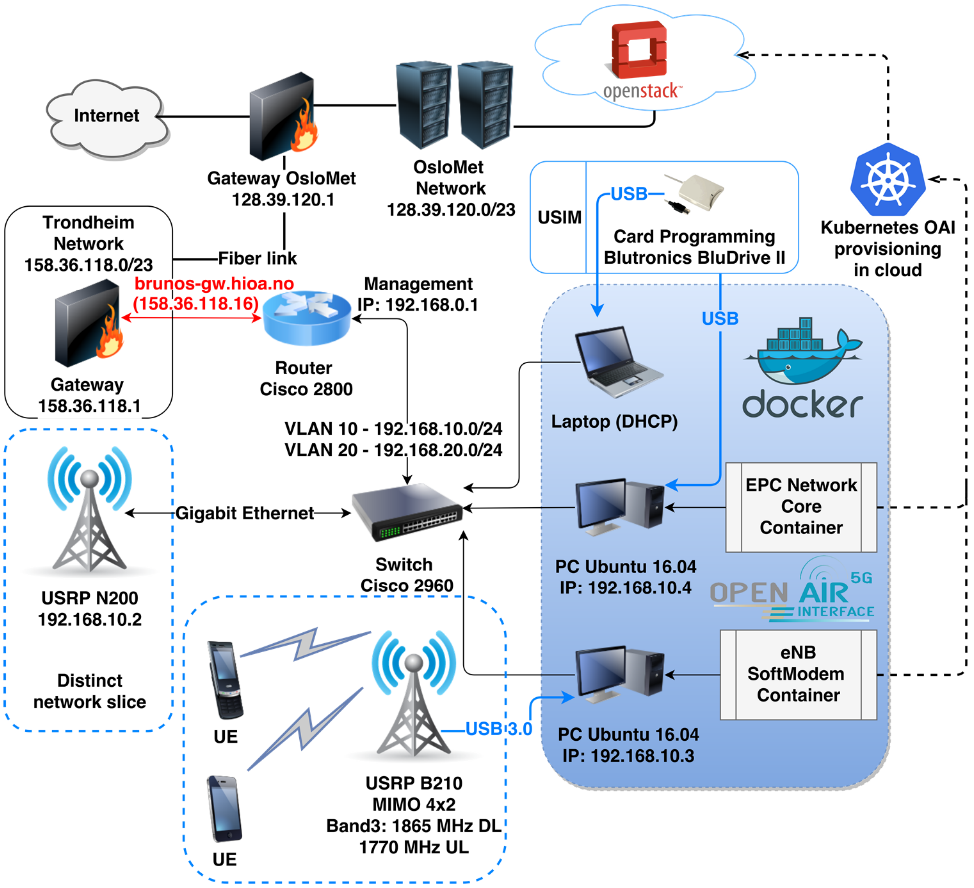


Figure 53. Architecture of the OpenAirInterface5G core network and eNB base station

As indicated in Figure 53, the local network hosts 3 machines that are physical computers. The first host is the 192.168.10.4 PC that runs the EPC network core from OpenAirInterface5G, which is accessible from the base station PC that is 192.168.10.3, and runs the LTE eNB SoftModem from OpenAirInterface5G. The PC is Intel NUC5i5MYHE portable box personal computer with 16GB DDR3 1600MHz RAM, Intel Core i5-5300U CPU with 3M cache and up to 2.9GHz frequency, running Linux Ubuntu 16.04 operating system on a M.2 SSD disk for a swift access to the HSS database and avoidance of possible bottlenecks.

The eNB is the current USRP B210, which connects through USB3.0 connection and allows mobile devices to access the EPC network core through the routing domain of the LAN 192.168.10.0/24. The eNB PC hardware is based on HP architecture with an Intel Core i7-4770 CPU 3.4GHz Haswell architecture (4-core, 8 threads), 16GB DDR3 RAM, and a normal HDD disk. Each mobile device is equipped with programmed SIM card that is performed on the Laptop (Windows) or on the PC 192.168.10.4, that runs Linux Ubuntu 16.04 operating system. With the appropriate credentials set in the EPC, the UE can access the database and successfully authenticate in the mobile network.

It is of utmost importance to note that the eNB and EPC run in Docker containers, allowing unprecedented flexibility. Further on, this type of sub-virtualized infrastructure will enable remote and automated provisioning into the OpenStack cloud, using an orchestrator such as Kubernetes or using OpenStack Heat template. Both PCs are running Calico SDN networking segment, with a custom policy defined for a granular access to the EPC elements, especially the database of the HSS module (see Appendix D, Appendix E and Appendix F). Calico is integrated with the Docker networking portion, and interworks with Open vSwitch to create tunnel for routing GTP-U traffic remotely. Containerized deployment allows immutability, which is a very useful feature if one prefers to maintain the form of an unfixed infrastructure. It allows ease of scaling and most importantly, enables rolling updates of the deployment without downtime.

4.1. Containerizing the infrastructure modules in Docker containers

In order to proceed with the packaging of the infrastructure into Docker containers (Dockerizing), the host machines need to meet several requirements. Particularly, since the mobile network processing is of real-time nature, it is of utmost importance to implement a Linux low-latency kernel. Thus, the low-latency 4.8 kernel version is adopted on the Ubuntu 16.04 operating system:

```
   wget http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.8/linux-headers-4.8.0-040800-
lowlatency_4.8.0-040800.201610022031_amd64.deb

   wget http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.8/linux-image-4.8.0-040800-
lowlatency_4.8.0-040800.201610022031_amd64.deb

   sudo dpkg -i linux-headers-4.8.0-040800-lowlatency_4.8.0-
040800.201610022031_amd64.deb

   sudo dpkg -i linux-image-4.8.0-040800-lowlatency_4.8.0-
040800.201610022031_amd64.deb
```

Moreover, the hardware-level power management features can impede with the real-time operation, which thus need to be disabled; explicitly, all the possible properties related to power management in BIOS are disabled: sleep states, C-states, P-states and CPU frequency scaling, which is the Intel SpeedStep. The hyperthreading should be disabled as well, but it may not interfere with some operations, since it is also required for running full-duplex MIMO eNB process. To test the stability of the CPU frequency fluctuations, the command: `watch |grep \"cpu MHz\" /proc/cpuinfo` gives details about the EPC host. In the Linux boot options, particularly in the */etc/default/grub*, the lines *intel_pstate=disable* and *GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_pstate=disable processor.max_cstate=1 intel_idle.max_cstate=0 idle=poll"* are added. The grub boot loader is updated with *update-grub*. In the end of the */etc/modprobe.d/blacklist.conf* the power-clamping is blacklisted (with the addition of a single line *blacklist intel_powerclamp*), since it can cause CPU frequency discrepancies. To check the CPU frequency stability, the i7z application is used (see Appendix G).

To facilitate maximum performance from the PCs, it is required to install *cpufrequtils,* which needs to have a line added in */etc/default/cpufrequitls* as follows: *GOVERNOR=”performance”.* The on-demand daemon needs to be thus disabled, for the configuration to take effect after restart: *update-rc.d ondemand disable* and */etc/init.d/cpufrequtils restart.* Moreover, all the wireless or Bluetooth interfaces that are integrated in the machines need to be completely disabled. This will ensure that CPU scaling will be disabled, since most CPUs are mapping computation flags to the kernel on-demand. In this case, everything will be diverted towards maximum performance and the CPU utilization will be 100%.

As a precondition, the both Linux Ubuntu 16.04 machines need to run Docker. The Docker daemon is installed via the official Docker website. To build the image with Dockerfile, the application has to be containerized by packaging it from the appropriate Git repository into the Docker 14.04 image. This way, a cross-OS operation is demonstrated, since the host OS is Ubuntu 16.04 and the Docker base image is Ubuntu 14.04.

4.1.1. Containerizing the EPC elements for the network core (HSS, MME, S/PGW)

The host that hosts the EPC container is named *networkcore* with an IP address 192.168.10.4. The realm that’s required for the OpenAirInterface5G network is referred to as *openair4g.eur* and thus he FQDN name for the EPC should be *epc.openair4g.eur* that resolves to the IP address of the host: 192.168.10.4, or the Docker container that will run in the 172.19.0.0/24 network (172.19.0.2).

The HSS and the MME certificates are arranged to last until 2018 August, and can be renewed using the HSS script certificate-generator, found in */openair-cn/scripts* in the container. The network elements are configurable just like when installing the EPC in the operating system directly. The Docker bridge, as well as the other interfaces should have increased MTU (from 1500 to at least 1648) to support successful connection via SCTP and TCP/UDP between the modules. Since the EPC container runs on a different host from the eNB container, a L3 routing is necessary that is configured with the Calico SDN networking with a configured BGP router for reducing complexity and easier network troubleshooting and scaling. Since mobile networks use RIP routing protocol, the BGP is based on path-vector protocol that is a version of the Bellman-Ford algorithm also used in RIP. Another advantage is the support for MPLS termination and support of various traffic that can be routed to remote workloads in the cloud.

According to those traits, a script is built to manually initialize the Docker network named *“oainet”* and add virtual OvS interfaces with virtual bridges using the Linux *bridge-utils*, while assigning custom MTU values and specific IP addresses and networks (see Appendix H). After the initialization of the OvS and creating a tunnel to a remote IP address, for running the Evolved Packet Core container, the following command should be issued:

```
   docker run -d -it --net=oainet --name=oai_epc --restart=unless-stopped --expose=1-
9000 -h=epc --privileged=true --cap-add=ALL -v /dev:/dev -v /lib/modules:/lib/modules
brunodzogovic/oai_epc
```

Accordingly, all the FQDN names need to be set in order for the modules to communicate between each other on the localhost inside the container:

```
127.0.0.1 localhost
127.0.1.1 mme.openair4g.eur       mme
127.0.1.1 hss.openair4g.eur       hss
127.0.1.1 spgw.openair4g.eur      spgw
127.0.0.1 epc.openair4g.eur       epc
```

The Dockerfile that is created is comprised of procedures for cloning the OpenAirInterface5G Git repository and installing the necessary components. The creation of the Docker image can be also performed manually, and for the convenience of elucidation, the steps are described subsequently.

As an initial phase, the installation of SSL certificates is initiated via automated scripts:

```
cd ~/openair-cn/SCRIPTS
./check_hss_s6a_certificate /usr/local/etc/oai/freeDiameter/ hss.openair4G.eur
./check_mme_s6a_certificate /usr/local/etc/oai/freeDiameter/ epc.openair4G.eur
```

The scripts are checking for the validity or existence of already-installed SSL certificates, and proceeds with generating new ones using OpenSSL.

With the certificates installed, the setup of the HSS, MME and S/PGW follows:

```
cd ~/openair-cn
cd SCRIPTS
./build_hss –c
./run_hss -i ~/openair-cn/SRC/OAI_HSS/db/oai_db.sql #Run only once to install database
cd ~/openair-cn/SCRIPTS
./build_mme -c
cd ~/openair-cn
cd SCRIPTS
./build_spgw –c
```

With the EPC constituents built inside the container, a directory is created in */usr/local/etc/oai/freeDiameter* and the configuration files moved accordingly:

```
mkdir -p /usr/local/etc/oai/freeDiameter
cp ~/openair-cn/ETC/mme.conf /usr/local/etc/oai
cp ~/openair-cn/ETC/hss.conf /usr/local/etc/oai
cp ~/openair-cn/ETC/spgw.conf /usr/local/etc/oai
cp ~/openair-cn/ETC/acl.conf /usr/local/etc/oai/freeDiameter
cp ~/openair-cn/ETC/mme_fd.conf /usr/local/etc/oai/freeDiameter
cp ~/openair-cn/ETC/hss_fd.conf /usr/local/etc/oai/freeDiameter
```

The HSS, MME and S/PGW configuration files are adjusted to correspond to the particular deployment and network parameters (see Appendix I, Appendix J and Appendix K, correspondingly). In the */usr/local/etc/oai/freeDiameter/hss_fd.conf* configuration files for the Diameter authentication server, it is essential to set up the proper FQDN identity and the realm that is previously assigned:

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

The equivalent action applies for the *mme_fd.conf* configuration file in the same directory, since MME is responsible for the Diameter authentication procedure establishment:

```
Identity = "epc.openair4G.eur";
Realm = "openair4G.eur";
ConnectPeer= "hss.openair4G.eur" { ConnectTo = "127.0.0.1"; No_SCTP ; No_IPv6;
Prefer_TCP; No_TLS; port = 3868;  realm = "openair4G.eur";};
```

Executable scripts are located at /openair-cn/SCRIPTS, only start the necessary services (apache and MySQL database):

```
/start_service.sh
```

and afterwards, the EPC can be run accordingly:

```
./run_hss
```

```
./run_mme -i
```

```
./run_spgw -i
```

With this, the EPC is functional and running. The last phase is to commit the changes and finalize the Docker image, so it can be tagged accordingly and pushed to the public/private repository:

```
docker commit –a "Bruno Dzogovic" –m "latest EPC version committed"
brunodzogovic/oai_epc:latest
docker push brunodzogovic/oai_epc:latest
```

This action will push the EPC image to the *https://hub.docker.com/r/brunodzogovic/oai_epc/* repository.

Accessing the HSS database is available on the localhost or the container's IP address: http://172.19.0.2/phpmyadmin in which the required user parameters are being stored (phone number, security settings etc). In the database, there are few tables: the PDN information, Users table, APN and PGW tables (refer to the examples in Appendix M and Appendix N). PGW and APN tables contain the address of the Packet gateway location that is 172.19.0.1 (the container network gateway). The APN is the name of the network realm, which is *oaiipv4* (designating that IPv4 is used for routing). This information is used in the UE to assign a Packet Data Network name which the mobile data will use to access the 4G network. In order to access the network, the focus is moved to the Docker networking portion. As explained before, Docker offers several options for networking from which the automatic one is the usage of the Docker Bridge. The Calico network driver offers two modes of operation: integration as a driver in the Docker networking, or using as a standalone SDN. In this case, both options are employed, since Calico acts not only as an inter-container communication solution, but also as a backbone networking, offering BGP connection as an ASBR (Autonomous System Boundary Router); with the support of MPLS, Calico can serve massively-scaled workloads on the network Layer 3. Another option is to use the Docker's MACVLAN driver, which will enable L2 connection and more reliable connection between the EPC and eNB+RRH using MAC addresses instead of IP (especially if the containers are running in clustered mode with Kubernetes or Swarm). If VxLAN is employed, then the SCTP+UDP are going to be encapsulated in L2TP, creating a network overlay and add a slight but unnoticeable overhead. On the other side, that will provide additional layer of security between the network core and the eNB.

4.1.2. Containerizing the eNB base-station and regulating the wireless radio propagation parameters

Similarly to the procedure explained in the previous subchapter about containerizing the EPC modules, the eNB is built in a Docker container image based on Linux Ubuntu 14.04, while running on a host operating system Ubuntu 16.04. The FQDN for the eNB PC is designated as *192.168.10.5 bs.openair4g.eur*. In a Dockerfile or after running the eNB container, it is necessary to install required software and clone the OpenAirInterface5G Git repository for the eNB:

```
apt-get update
apt-get install software-properties-common git wget psmisc –y
GIT_SSL_NO_VERIFY=true git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git
cd openairinterface5g
git checkout develop
git pull
source oaienv
```

Following the cloning of the Git repository, the eNB is built using the USRP option, since the given hardware is the USRP B210:

```
cd cmake_targets
./build_oai -w USRP –I –x –c --eNB
```

```
#verify that USRP is working
uhd_usrp_probe
```

To run the eNB with specifically designated parameters, the configuration file that is located in *$OPENAIR_DIR/targets/PROJECTS/GENERIC-LTE-EPC/CONFenb.band3.tm1.usrpb210.conf* is set for the Band 3. Since computational resources are limited on a PC, the higher frequencies require much agile CPU, and therefore, instead of Band 7, the Band 3 is chosen (see Appendix L).

```
cd cmake_targets/lte_build_oai/build
sudo -E ./lte-softmodem -O $OPENAIR_DIR/targets/PROJECTS/GENERIC-LTE-
EPC/CONF/enb.band3.tm1.usrpb210.conf -d
sudo -E ./lte-softmodem -h #(to see help options)
```

With Docker, the changes are committed, the image tagged as a latest version and pushed to the *https://hub.docker.com/r/brunodzogovic/oai5g_enb/* repository:

```
docker commit -a “Bruno Dzogovic” -m “latest eNB version committed”
brunodzogovic/oai5g_enb:latest
docker push brunodzogovic/oai5g_enb:latest
```

The radio configuration adjustment in this stage is a tedious task because the eNB operation is delicate and the UE attachment procedure depends on numerous parameters. The frequency selection, according to the Band 3, refers to the 1.8 GHz spectrum that is the 1865 MHz for the downlink channel and 1770 MHz for the uplink channel, with 95 MHz duplex spacing. The frame type chosen is FDD and the channel bandwidth is set to 5 MHz, since increment of the bandwidth demands a superior CPU and will cause increased error rate. There are two practicable preferences of operation, specifically: SISO (using one Tx transmission and one Rx receiving antenna), as well as full-duplex MIMO using 2 antennas for Rx and 2 antennas for Tx. Given the implementation of MIMO, the Tx and Rx gains are calculated to 90 dBm and 105 dBm, respectively. Any altered levels may instigate discrepancies at the uplink channel and decreased system stability, which will cause the UE to detach or the MME to crash.

Most important factor to be taken into consideration is the RACH (Random Access Channel) structure. In order to stabilize the function of a UE through the network, it is essential to institute a ground base substances for power control of the random access channels (PRACH). Initially, since the base station is transmitting in a limited manner (inside a single room), the distance between the UE and the eNB is varying between 1m and 5m. Consequently, as an initial starting point, the average free-space path loss is calculated (FSPL). For the downlink channel, the values are taken for transmission at 1m average, since the obstruction of transmission is zero (direct line of sight to the eNB):

$$FSPL_{(DL)} = 10\log_{10}\left(\frac{4\pi df}{c}\right)^2 = 20\log_{10}\left(\frac{4\pi df}{c}\right) = 20\log_{10}\left(\frac{4\pi\cdot 0.001\cdot(1.865\cdot 10^6)}{299792}\right) = 20\log_{10}0.07817513847 = -22{,}13dBm,$$

For 1m distance from the eNB and where:

$d = 0.001km$

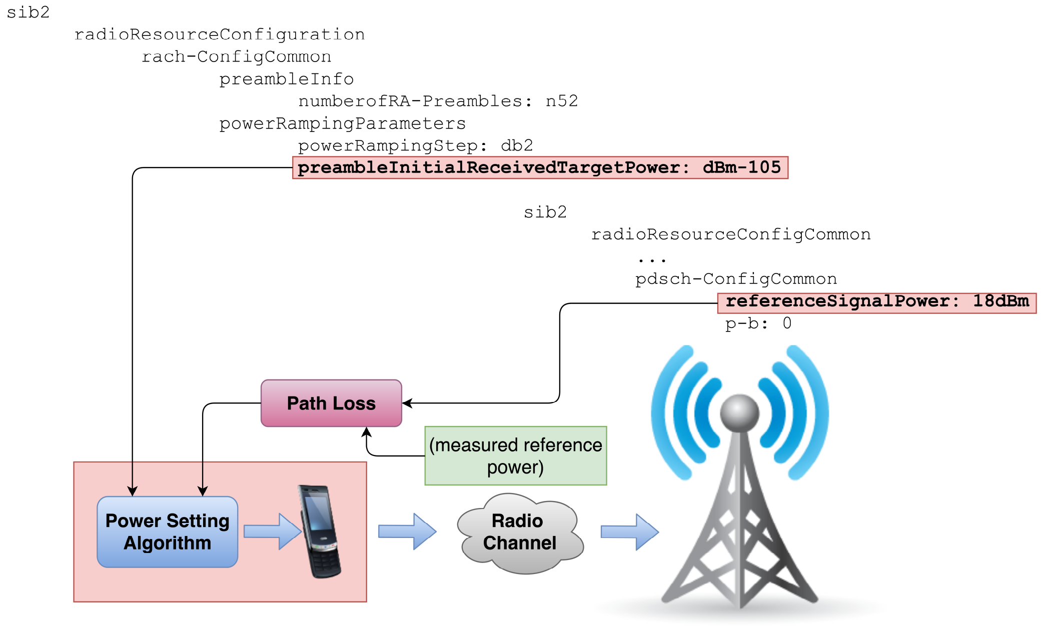
$f = 1865MHz = 1.865x10^6Hz$

$c = speed\ of\ light = 299792km/s$

Analogously, the calculations are applied for the Uplink channel at 1770 MHz. This step will ensure that the radio propagation does not extend out of the building premises, since the transmission at band 3 has a greater range than transmission at higher frequencies. This is very important because the radio propagation should not interfere with the commercial infrastructure due to strict law restrictions for transmission at commercial radio bands.

Following the calculation of free-space path loss of the radio waves, it is thus necessary to compute the optimal channel configuration. For a comparative reference, in wired communications the amount of power being sent from the transmitter reaches the receiver without ample variation. However, in wireless communications, the energy drop at the receiving point may be immense, which requires higher power amplifiers for increasing the power of transmission. The solution might seem simple but it doesn't come without repercussions. If the distance between the UE and the eNB is reasonable, then it is reasonable to increase the power of the signal transmission. However, if the distance is very close then the receiver will be oversaturated. Handling this situation can be simple in fixed conditions. Specifically, if the distance between the transmitter and receiver does not change and the channel conditions (air humidity, precipitation, obstructions) remain fixed, a manual setup would work. On the contrary, the distance between the receiver and transmitter and channel conditions are subjects to a frequent variation, which requires for a dynamic power control mechanism usually referred to as Closed Loop Power Control. The dynamic approach takes few stages into consideration; namely, after initial transmission of the signal towards the receiver, it [the receiver] measures the power of the received signal. Afterwards, if the measured signal power is too low, the receiver sends an "increase the power" command to the transmitter. Also, if the received signal power measured is too strong, it would send a "decrease the power" command (3GPP, 2009) [3GPP Specification TS 36.213].

In reality, the resolution of this case is much more complex. For example, many times the transmitter and the receiver are not in such state. When the UE is being turned on, it has to send an initial message in the first signal. This moment is very important, since the signal transmitted might have too low power and the base station would not detect it. If the transmitted signal power from the UE is very high, then it can interfere with other UE units. To address this issue, a more granular approach is being utilized. Given the fact that the eNB is transmitting a certain reference signal with fixed power value, the information about that value is being advertised through the network. Besides that, the network also advertises the information about the maximum allowed power that a single UE can transmit. When the UE receives this information, it decodes the reference signal coming from the eNB and measures the power. Accordingly, UE measures the FSPL and correspondingly to the allowed power, it can fathom with how much power it should categorically transmit. This model of power determination process is referred to as Open Loop Power Control. To initiate the Open Loop Power Control, the PRACH power should be determined, which is being carried by SIB messages (see Figure 54) (3GPP, 2009) [3GPP Specification TS 36.213].



```
sib2
      radioResourceConfiguration
            rach-ConfigCommon
                   preambleInfo
                          numberofRA-Preambles: n52
                   powerRampingParameters
                          powerRampingStep: db2
                          preambleInitialReceivedTargetPower: dBm-105
```

```
sib2
      radioResourceConfigCommon
          ...
          pdsch-ConfigCommon
                  referenceSignalPower: 18dBm
                  p-b: 0
```

Figure 54. PRACH Open Loop Power Control

Furthermore, there are several entities that need to be calculated in order to establish efficient power control (3GPP, 2009) [3GPP Specification TS 36.213]. Particularly:

$$\boldsymbol{P_{PRACH}} = min\{P_{CMAX}, PREAMBLE\_RECEIVED\_TARGET\_POWER + PL\}\ [dBm]$$

$$\boldsymbol{P_{SRS}(i)} = min\left\{P_{CMAX}, P_{SRS\_OFFSET} + 10\log_{10}(M_{SRS}) + P_{0_{PUSCH}}(j) + \alpha(j)\cdot PL + f(i)\right\}\ [dBm]$$

$$\boldsymbol{P_{PUCCH}(i)} = min\left\{P_{CMAX}, P_{0\_PUCCH} + PL + h\left(n_{CQI}, n_{HARQ}\right) + \Delta_{F_{PUCCH}}(F) + g(i)\right\}\ [dBm]$$

$$\boldsymbol{P_{PUSCH}(i)} = min\left\{P_{CMAX}, 10\log_{10}(M_{PUSCH}(i)) + P_{0_{PUSCH}}(j) + \alpha(j)\cdot PL + \Delta_{TF}(i) + f(i)\right\}\ [dBm]$$

$$\boldsymbol{PH(i)} = P_{CMAX} - \left\{10\log_{10}\left(M_{PUSCH}(i)\right) + P_{0_{PUSCH}}(j) + \alpha(j)\cdot PL + \Delta_{TF}(i) + f(i)\right\}\ [dB]$$

The power of the channel per subframe - *i* suggests that the channel powers are premeditated and set for each subframe. According to the first formula, the scheming takes the minimum value (smaller value) between the P_CMAX and the subsequent calculation, meaning that if the formula gives the value smaller than P_CMAX; the channel power for a subframe - *i* will become the value given by the formula. On the other hand, if the formula gives the value greater than the P_CMAX, the power of the channel becomes the actual P_CMAX value. In any case, the power of the channel cannot exceed the value of P_CMAX. According to these calculations, the manufacturers of the UE need to ensure that the devices do not transmit any power greater than these particular power values (3GPP, 2009) [3GPP Specification TS 36.213].

The PUSCH power control has a subframe number *i* and a variable *j* that can be either 1 or 0. The $M_{PUSCH}(i)$ represents the number of resource blocks allocated for the UE. The $P_{0\_PUSCH}(j)$ is the P_{0_PUSCH} nominal uplink power plus the $P_{0\_UE\_PUSCH}$ power, where the two for *j*=0, 1 come from higher layer (the SIB2 messaging). The P_{0_PUSCH} nominal power comes from *p0-NominalPUSCH* header in the SIB2 message and the $P_{0\_UE\_PUSCH}$ comes from the *p0-UE-PUSCH* header in the SIB2 message as an initiator for RRC connection setup and reconfiguration. The *alpha(j),* for *j*= 0, 1 can has values of {0, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}. The PL variable refers to the calculated Path Loss, which also comes from higher layer filtered

RSRP (Reference Signal Power) in the configuration file, and is defined in the SIB2 message (3GPP, 2009) [3GPP Specification TS 36.213].

The PUCCH power control also has the *i* – subframe number and the *j* variable that can be 0 or 1. The P_{0_PUCCH} is the P0 PUCCH nominal power plus the P0 UE PUCCH power, where they are set from higher layer, or specifically the SIB2 message for RRC connection setup and/or reconfiguration. The delta format defines the modulation scheme at the PUCCH control, which can be *1* – no modulation scheme; *1a* – BPSK modulation with 1 bit per subframe; *1b* – QPSK modulation with 2 bits per subframe; *2* – QPSK with 20 bits per subframe; *2a* – QPSK+BPSK modulation with 21 bits per subframe; *2b* – QPSK+QPSK with 22 bits per subframe; *3* – QPSK modulation with 48 bits per subframe and delta 4 and delta 5 (3GPP, 2009) [3GPP Specification TS 36.213].

To determine the RACH preamble, the initial message established from the UE to the eNB when the device turns on is accordingly expounded. The Random Access Channel (RACH) assignment procedure is the most important entity, which can conclude the formation of the configuration file used in the operation of the eNB (refer to Appendix O for detailed insight into the eNB configuration). The RACH process achieves UL synchronization between the UE and eNB and obtains resources for Message 3 (i.e. RRC Connection Request), therein the importance for the Random Access Channel because synchronization between transmitter and receiver is the ultimate objective. The synchronization is achieved via special synchronization channel that broadcasts to all UEs and transmits synchronization messages at a certain interval. According to that, the synchronization process happens only when there is an instant necessity and should be dedicated to only a specific UE. The RACH procedure happens in case of an initial access from RRC_IDLE or with RRC connection re-establishment procedure. Moreover, it is initialized in case of handover or DL/UL data arrival during RRC_CONNECTED when the synchronization status is "non-synchronized". The RACH procedure is also used for positioning purposes when timing advance is needed for UE positioning. There are two types of RACH procedures: Contention-based and Contention-free RACH. When a UE transmits a RACH preamble, it can give out a particular pattern that is called 'signature'. In each LTE cell, there are 64 possible preamble signatures from which the UE selects randomly. This alludes on the possibility of selecting two identical signatures simultaneously, which can induce collisions. In the Contention-based RACH procedure, multiple UEs may select the same signature that are being resolved later by the network in a contention resolution step. The other possibility is restricting the selection of a signature in a time domain, which means that the UE gets information from the network at which time which particular preamble signature to use. The network knows when the UE will send the RACH even before the UE sends it, because the network informs the UE about the time when it is supposed to transmit the RACH. The selection of a PRACH preamble is denoted in the Table 10, which contains the pre-calculated values for a given configuration index and corresponding preamble format and subframe number (3GPP, 2009) [3GPP Specification TS 36.213]:

Table 10. PRACH configuration index (3GPP, 2009) [3GPP Specification TS 36.213]

PRACH configuration index	Preamble format	System frame number	Subframe number	PRACH configuration index	Preamble format	System frame number	Subframe number
0	0	Even	1	32	2	Even	1
1	0	Even	4	33	2	Even	4
2	0	Even	7	34	2	Even	7
3	0	Any	1	35	2	Any	1
4	0	Any	4	36	2	Any	4
5	0	Any	7	37	2	Any	7
6	0	Any	1, 6	38	2	Any	1, 6
7	0	Any	2, 7	39	2	Any	2, 7
8	0	Any	3, 8	40	2	Any	3, 8

9	0	Any	1, 4, 7	41	2	Any	1, 4, 7
10	0	Any	2, 5, 8	42	2	Any	2, 5, 8
11	0	Any	3, 6, 9	43	2	Any	3, 6, 9
12	0	Any	0, 2, 4, 6, 8	44	2	Any	0, 2, 4, 6, 8
13	0	Any	1, 3, 5, 7, 9	45	2	Any	1, 3, 5, 7, 9
14	0	Any	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	46	N/A	N/A	N/A
15	0	Even	9	47	2	Even	9
16	1	Even	1	48	3	Even	1
17	1	Even	4	49	3	Even	4
18	1	Even	7	50	3	Even	7
19	1	Any	1	51	3	Any	1
20	1	Any	4	52	3	Any	4
21	1	Any	7	53	3	Any	7
22	1	Any	1, 6	54	3	Any	1, 6
23	1	Any	2, 7	55	3	Any	2, 7
24	1	Any	3, 8	56	3	Any	3, 8
25	1	Any	1, 4, 7	57	3	Any	1, 4, 7
26	1	Any	2, 5, 8	58	3	Any	2, 5, 8
27	1	Any	3, 6, 9	59	3	Any	3, 6, 9
28	1	Any	0, 2, 4, 6, 8	60	N/A	N/A	N/A
29	1	Any	1, 3, 5, 7, 9	61	N/A	N/A	N/A
30	N/A	N/A	N/A	62	N/A	N/A	N/A
31	1	Even	9	63	3	Even	9

4.2.Connecting the eNB with the EPC through container network

After completion of the radio configuration, the remaining step is to interconnect the two remote workloads: the eNB container and EPC container. As previously stated, they belong to a same Docker network named as *"oainet"* and take the address space of the 172.19.0.0/24 network (see Figure 55). To separate the workloads on Layer-2, a virtual switch with OvS is initialized. The virtual bridges that are created are used as reference points for assigning an autonomous system with Calico, instead of using the eth0 interface. This allows for further expansion of access control and policy-based networking. Because Kubernetes is being used for provisioning the infrastructure in microservices architecture, the default Flannel SDN is replaced by Calico. An environment variable is set in a .cfg file and Calico is initialized via Kubernetes addon:

```
NETWORKING_OPTION=Calico
kubeadm init
Flannelkubectl create - f http://docs.projectcalico.org/v2.0/getting-
started/kubernetes/installation/hosted/kubeadm/calico.yaml
```

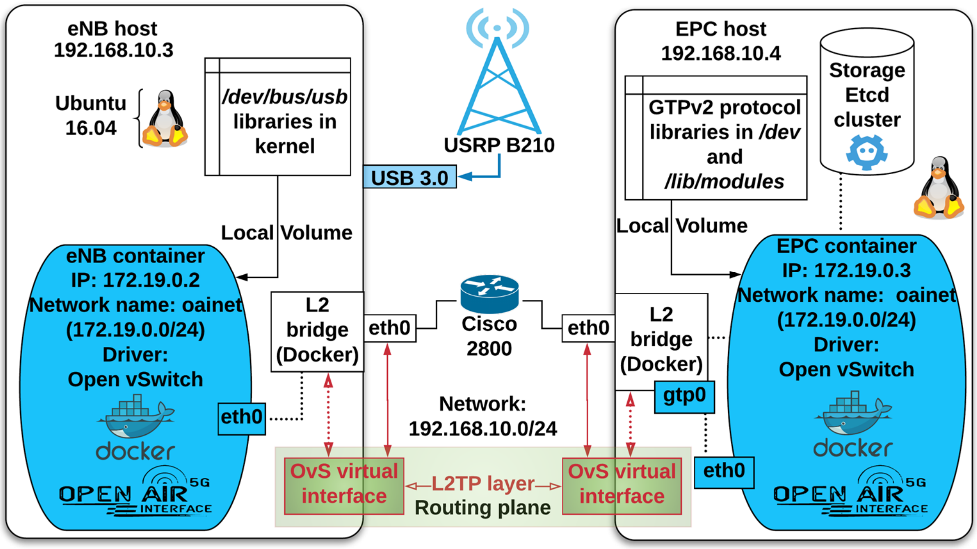


Figure 55. Direct container-to-container communication using OvS

Calico allows isolation of a Kubernetes namespace. Since the EPC can run in multiple replicas in a same or different namespace, Calico can ensure that a specific namespace policy will isolate some constituents from remote access and guarantee that only the eNB container can access the EPC container, given that their IP addresses and MAC addresses are fixed:

```
kubectl create ns policy-oai
kubectl run --namespace=policy-oai oai_epc --replicas=2 --image=oai_epc
kubectl expose --namespace=policy-oai deployment oai_epc --ports=1-9000
```

By enabling isolation for the current policy of the namespace, Calico will prevent connections to the pods in that particular namespace:

```
kubectl create -f - <<EOF
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
  namespace: policy-oai
spec:
  podSelector:
    matchLabels: {}
EOF
```

When allowing access to the oai_epc service using a network policy, incoming connections from our access Pod will be allowed only, but not from anywhere else:

```
kubectl create -f - <<EOF
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-oai_epc
  namespace: policy-oai
spec:
  podSelector:
    matchLabels:
      run: oai_epc
  ingress:
    - from:
      - podSelector:
          matchLabels:
            run: access
EOF
```

At this point, the container environment is secured with a simple Calico policy. To proceed further, the networking model is premeditated. Since the physical router uses the OSPF protocol, Calico will need to redistribute its physical routes into the SDN BGP autonomous system. For this purpose, there are several factors that need to be taken into consideration. First and foremost, the AS (Autonomous System) has a number which is used by the BGP agent on a Calico node when it has not been explicitly specified. In practice, all the Calico nodes use the same AS number for the matter of simplicity. Moreover, the node-to-node mesh is enabled by default, which provides a mechanism for automatic configuring peering between all Calico nodes. This is useful for smaller deployments, but when the infrastructure is scaled, then the full node-to-node mesh is disabled and explicit BGP peers are configured for the Calico nodes. With a simple command, the status of the peerings on Calico nodes can be checked:

```
sudo calicoctl node status

Calico process is running.

IPv4 BGP status
+--------------+-------------------+-------+----------+-------------+
| PEER ADDRESS |     PEER TYPE     | STATE |  SINCE   |    INFO     |
+--------------+-------------------+-------+----------+-------------+
| 172.19.0.2   | node-to-node mesh | up    | 23:30:04 | Established |
| 172.19.0.3   | node-to-node mesh | up    | 23:30:27 | Established |
| 10.20.30.40  |       global      | start | 10:16:13 |   Connect   |
| 192.10.0.0   |   node specific   | start | 10:28:46 |   Connect   |
+--------------+-------------------+-------+----------+-------------+

IPv6 BGP status
+--------------+-------------------+-------+----------+-------------+
| PEER ADDRESS |     PEER TYPE     | STATE |  SINCE   |    INFO     |
+--------------+-------------------+-------+----------+-------------+
| aa:bb::ff    | node-to-node mesh | up    | 23:17:26 | Established |
+--------------+-------------------+-------+----------+-------------+
```

The IP pools need an external connectivity for OSPF route redistribution, and therefore, the NAT option is turned on:

```
calicoctl get ipPool

cat << EOF | calicoctl apply -f -
- apiVersion: projectcalico.org/v3
  kind: IPPool
  metadata:
    name: ippool-ext-1
  spec:
    cidr: 172.19.0.0/24
    natOutgoing: true
EOF
```

The last thing remaining is setting the MTU to 1648, which is added to the CNI configuration file of the Calico control daemon:

```
{
    "name": "any_name",
    "cniVersion": "0.1.0",
    "type": "calico",
    "mtu": 1648,
    "ipam": {
        "type": "calico-ipam"
    }
}
```

Since the argument *–ip-autodetection-method* is enabled in Calico, the configuration of the network is automatic. However, it is possible also to manually set the subnets and all IP addresses for every individual container network.

5. SECURITY AND AUTHENTICATION OF THE MOBILE NETWORK

When a UE attempts to attach to the mobile network, it checks whether valid credentials are present in the HSS database. Through previously-explained procedures, the MME initiates attachment procedures of the UE to the HSS via the Diameter protocol. If the credentials do not match, then the device will be denied access. In order to successfully authenticate, the UE devices need a specifically programmed SIM card that supports MILENAGE algorithm.

The UE devices used are Huawei P9 Lite phones (HUAWEI, 2018) VNS-L22 series. The phone supports 4G network operation on multiple bands, namely: FDD band 1, 3, 5, 7, 8, 19, 28 and TDD band 40; 3G operation on bands 1, 5, 6, 8 and 19; as well as, 2G operation on GSM bands 850 MHz, 900 MHz, 1800 MHz and 1900 MHz. For the purpose of successful establishment of a reliable connection, blank SIM cards are programmed.

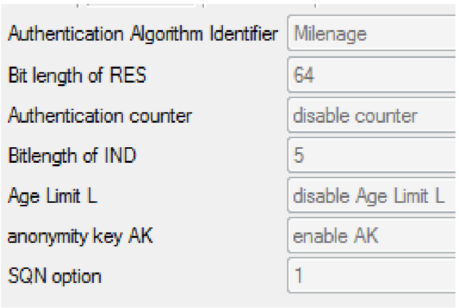
5.1. Building USIM cards with MILENAGE encryption for authentication

When embedding the information to the SIM cards, it is essential to take into account few parameters. Most important is the operator key (OP), which is used by the HSS for derivation of other keys and is kept as a secret by the service provider (operator). The encryption keys and parameters are represented in Table 11.

Table 11. SIM card programming values and keys

SIM cards	Algorithm	Ki	OP	SPN	MCC	MNC	IMSI	SQN
SIM 1	Milenage	8BAF473F2F8FD09487CCCBD7097C6862	11111111111111111111111111111111	OpenAirInterface	208	93	208930000000003	000000000004
SIM 2	Milenage	8BAF473F2F8FD09487CCCBD7097C6862	11111111111111111111111111111111	OpenAirInterface	208	93	208930000000004	000000000005
SIM 3	Milenage	8BAF473F2F8FD09487CCCBD7097C6862	11111111111111111111111111111111	OpenAirInterface	208	93	208930000000005	000000000006
SIM 4	Milenage	8BAF473F2F8FD09487CCCBD7097C6862	11111111111111111111111111111111	OpenAirInterface	208	93	208930000000006	000000000007

As seen in Figure 56, within the choice for authentication algorithm, the Milenage option is selected.



Authentication Algorithm Identifier	Milenage
Bit length of RES	64
Authentication counter	disable counter
Bitlength of IND	5
Age Limit L	disable Age Limit L
anonymity key AK	enable AK
SQN option	1

Figure 56. SIM programming parameters

After this process, the UE can securely authenticate and connect to the mobile network. For the purpose of programming a SIM card, a specific programming hardware is required. The procedure is detailed in the following subsection.

5.1.1. Programming a USIM card for the OpenAirInterface5G network

For establishing a secure and trustable communication between the Mobile/IoT platforms to the Internet, USIM blank cards are obtained and programmed for authentication of the devices to the core network. This subsection elucidates the appliance of a smart card reader with an appropriate kernel driver, a middle-ware for detection and usage of the cards and testing the diligence. The device used is a Blutronics BluDrive II model. The installation of the Chip/Smart Card Interface Devices (CCID) software driver in Linux is straightforward, including *pcsclite* and additional *pcsc* tools as middleware for testing, with which it is feasible to write on different types of smart cards. The procedure is explicated in Appendix P in detail.

5.2. Tunneling SCTP protocol into L2TP/VPN and advanced security control

The L2TP tunneling protocol can be used to carry higher-level protocol encrypted traffic (such as VPN), but does not offer encryption itself. Since the mobile communication security architecture is based on Diameter authentication, the security vectors are being carried via UDP and SCTP, which also carry the GTP-U protocol tunnel instructions (see Figure 57). Paired with IPSec, L2TP can offer substantial security enhancement. However, this measure will introduce a slight overhead in the mobile traffic, but that can be considered as a desirable tradeoff for enhancing the security of critical and exposed devices.

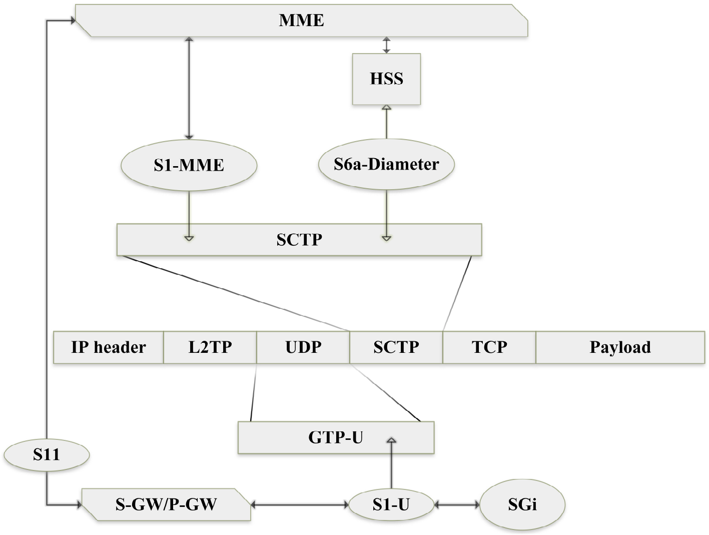


Figure 57. Tunneling encapsulation and packet modification

When a tunnel is established, the traffic is bidirectional. All the other higher-level protocols can be carried via the fixed tunnel. As a session is instituted, the default header size is adapted and requires alteration of the MTU. In this case, the MTU is set to 1648 bits, which is adequate for the current case. However, the MTU is also modified to up to 9000 bits for testing purposes, especially if IPSec is used and the traffic is encrypted or if additional tunnels are required. One disadvantage of a very large MTU value is the susceptibility to fragmentation attacks, and therefore it requires supplementary configuration for VRF on the Cisco router. It should also be noted that for proper communication, the MTU is adjusted properly at all interfaces: the *eth0* physical network interface, Docker virtual bridge, OvS bridges as well as Calico and *veth/gre* interfaces.

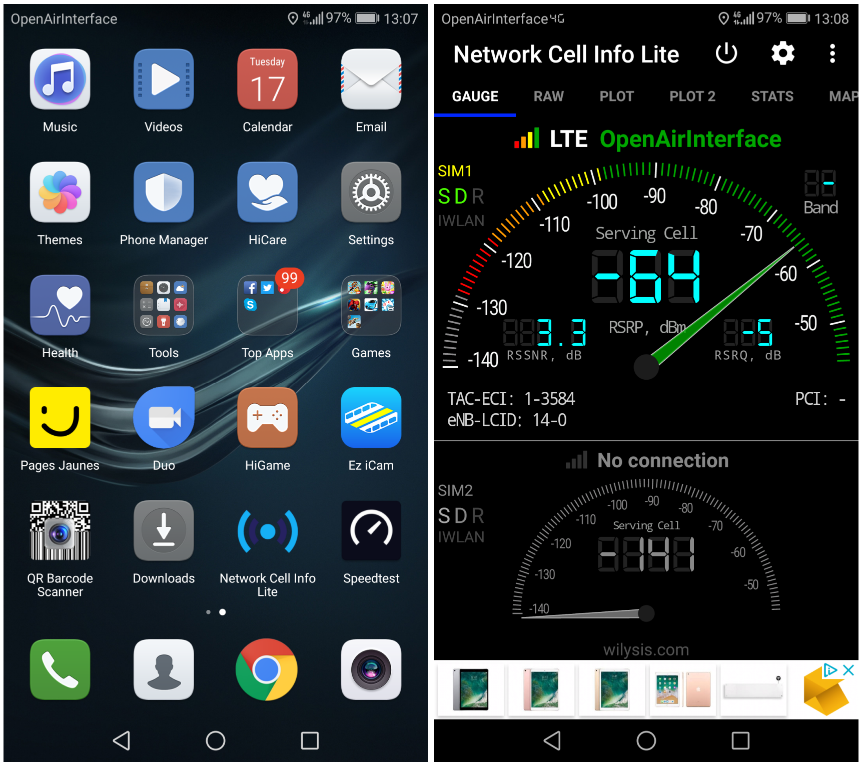
6. EVALUATION

The deployment of the mobile network requires only stability testing, which is represented in the following chapter. Since the aim of this research is to provide a suitable way of establishing a virtualized mobile network, the performance tests and optimization include several layers of analysis without delving in detailed tests or comparative methods. The tests are conducted consequently in accordance to the deployment progress of the infrastructure. The initial test phase incorporates defining the requirements for the hardware platform and its evaluation. After establishing the hardware-level procedurals, the higher-level operations testing is taking place; namely, defining virtualized environment and assessing its performance scaling options. When the virtualized environment is set, and the underlying deployment is primed, the succeeding procedures that require attention are the connectivity and networking. In particular, the SDN models are proposed, and the remote communication between the eNB and EPC is defined together with layer-2 and layer-3 network exactitudes. Hereafter, it can be asserted that the mobile network is successfully deployed, and all the subsequent testing procedures will then coalesce approaches such as wireless radio-link optimizations, network-level tuning and improving the deployment stability for long-term jobs; conclusively, the overall throughput and latency are assessed.

6.1. Testing the Access Stratum (AS) and the Non-Access Stratum (NAS)

The tests of the mobile infrastructure can be divided into examinations of the AS (Access Stratum) and assessments of the NAS (Non-Access Stratum). The AS tests include checking the radio access and ensuring that the adjusted parameters are adequate for the particular scenario. The signal analyzer run at the eNB shows that when the UE connects to the Band3 FDD, a 16-QAM modulation is instigated. Depending on the requirements, the option for 64-QAM is also enabled. In this case, if the traffic increases and the radio access doesn't provide the sufficient resources, the signal will be 64-QAM modulated to increase the throughput. When the uplink signal is saturated, the eNB will try to stabilize the broadcasted gain in order to alleviate the connection. This can be clearly observed from the two figures in Appendix Q, subsequently; the constellation of the 16-QAM becomes stabilized over time.

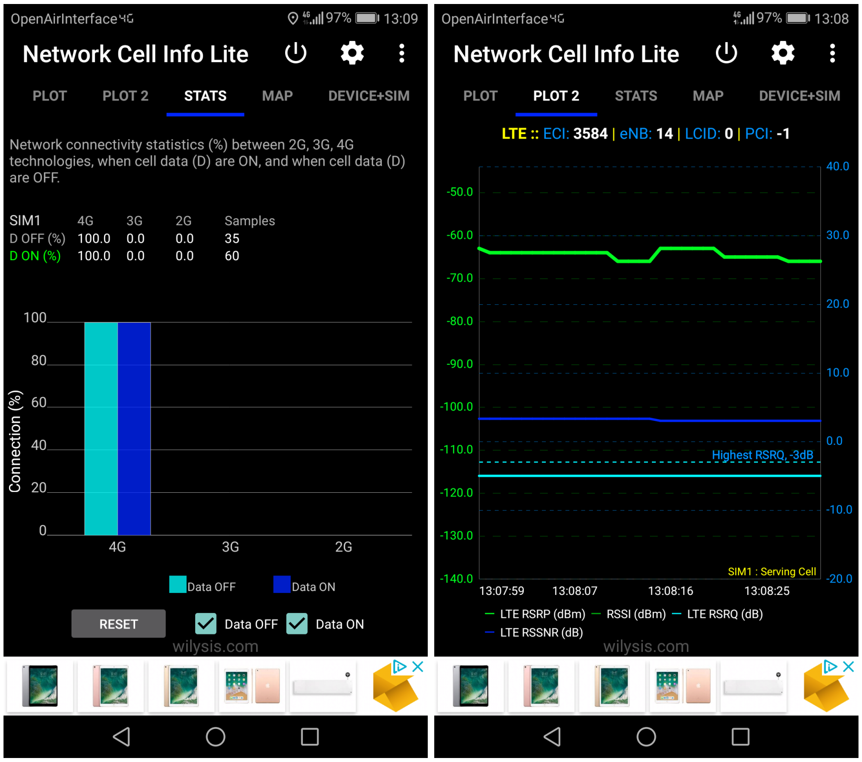
As previously stated, when the UE selects the frequency of the eNB and requests an attachment procedure initiation, it asks the MME for channel assignment, authentication commencement, obtaining an IP address from the S/PGW, bearer designation, etc. This is denoted in the Appendix R. When the HSS daemon is started, the MME connects through the S/PGW and to the HSS. At this point, the mobile network is in idle state, waiting for potential UE devices to authenticate to the EPC. In the UE, the APN (Access Point Name) needs to be set analogously to the one registered in the HSS, specifically: *oai.ipv4,* where the Mobile Country Code and the Mobile Network Code should correspond to the one embedded in the SIM card (208 and 93, respectively). Also, the SPN (Service Provider Name) is set to as *OpenAirInterface*, since that is the name registered in the HSS database and programmed into the SIM cards. When the SIM card is inserted into the UE device, the phone authenticates automatically on the mobile network. Using the open-source software *Network Cell Info Lite* (WILYSIS, 2018), the connection to the eNB can be metered (see Figure 58).



TAC-ECI: 1-3584
eNB-LCID: 14-0
PCI: -

Figure 58. Huawei P9 Lite connected to the OpenAirInterface5G network

The observation indicates clearly that the signal from the UE to the eNB has an exceptional quality with Reference Signal-to-Signal Noise Ratio (RSSNR) as minimal as 3.3dB and Reference Signal Received Quality (RSRQ) of -5dB. The overall Reference Signal Received Power (RSRP) is -64dBM in room conditions. According to this signal strength, the channel estimation can be calculated and thus adjusted accordingly for the particular conditions. In Figure 59, the normal function of a UE attached to the eNB can be ascertained. In this case, the serving cell maintains a stable connection without any mobility of the UE and thus, the RSRP signal power is situated around -65dBm. For eliminating biased results, the cell is set to only support 4G LTE connections, without the possibility to connect to 3G UMTS.

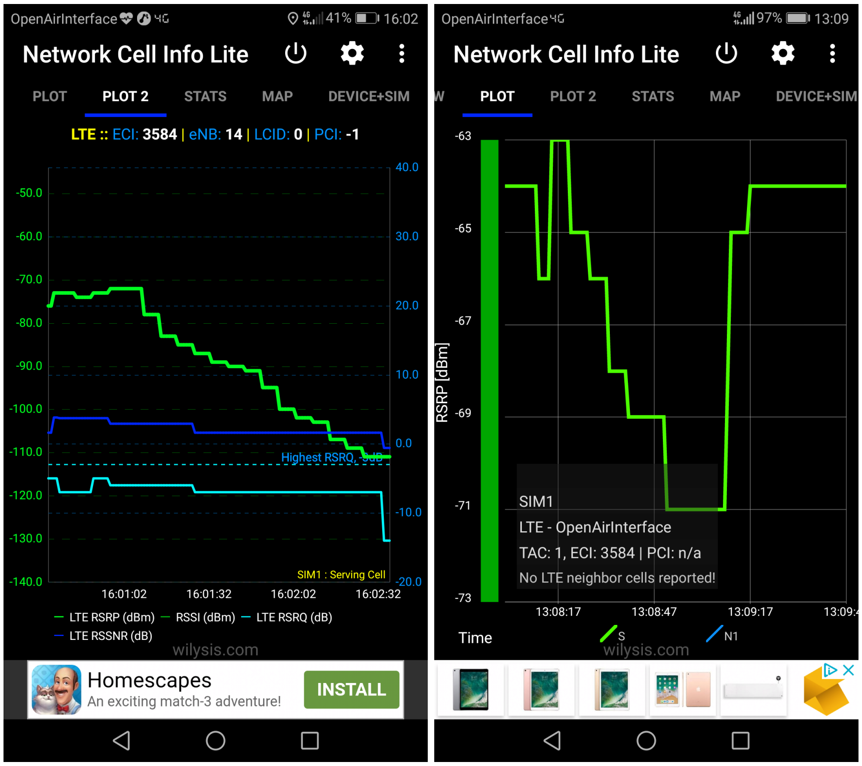


Network connectivity statistics (%) between 2G, 3G, 4G technologies, when cell data (D) are ON, and when cell data (D) are OFF.

SIM1	4G	3G	2G	Samples
D OFF (%)	100.0	0.0	0.0	35
D ON (%)	100.0	0.0	0.0	60

Figure 59. Normal operation of the UE while connected on the OpenAirInterface5G network

However, in case of mobility (linear distancing from the eNB); as depicted in Figure 60, the UE registers signal power drop in time domain. Alternatively, in the case of severe obstruction, the signal drops exponentially in time domain and then again rises exponentially fast when the obstruction of the line-of-sight (LoS) is resolved. It is also crucial to note that the signal power drop is much higher when the UE increases the distance from the eNB rather than as in the obstruction case. As shown in the Figure 60 on the left, with the higher distances from the eNB, the signal to the UE reaches unacceptable levels of low RSRQ quality. The signal strength drops drastically from around -70dBm to less than 100dBm. Consequently, the eNB will try to address the situation and push the minimal reference signal received power threshold, so it can broadcast with slightly higher power to reach the UE, but when the threshold reaches 0, then the UE is too far from any possibility of being reached from the eNB. Situationally, the eNB is forced to drop the connection and de-attach the UE. As long as the signal power is at least marginally above the acceptable RSRQ quality levels, the connection will be maintained; in other words, the connection is preserved as long as the link to the base station is at “one line” of the phone’s signal meter. In the second case, the eNB will continue broadcasting normally when obstruction is faced, but the received signal power will drop slightly (in the current case, from -63dBm to around -71dBm), which doesn’t affect the connection and the phone’s signal power meter will still indicate full signal strength link. Contextually, the eNB will not then re-adjust the power thresholds, nor will it upsurge the radiating power.

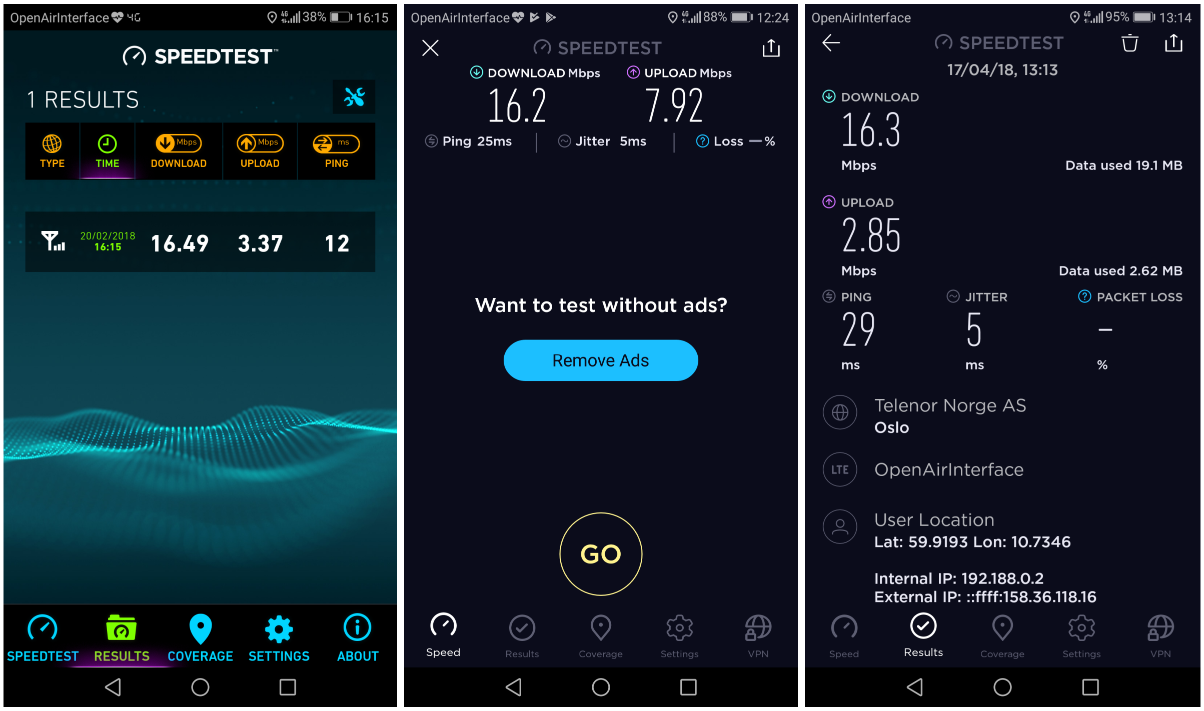


SIM1
LTE - OpenAirInterface
TAC: 1, ECI: 3584 | PCI: n/a
No LTE neighbor cells reported!

Figure 60. Signal power drop due to linear increasing of distance from the eNB and due to obstruction of line-of-sight, correspondingly

In addition to the AS evaluation, testing the NAS can take various dimensions and forms. Most of the possible tests in the EPC can be of IP character, testing the network traffic for particular traits (congestion, jitter, latency, bandwidth, routing) etc. The most important traits of the Non-Access Stratum is the ability to serve substantial amount of users through the eNB and offer suitable QoS, while providing good user experience. For the users, the most important factors are usually the bandwidth and the latency. When video calls are established it is important that the video quality is adequate so the communicating sides can enjoy good conversation, and the latency should be as minimal as possible for the voice exchange to be satisfactory and clear without delay. To test the throughput and the latency/jitter, two methods are applied: testing via external servers by using *speedtest.net* (SPEEDTEST, 2018) and testing using the open-source *iperf3* tool (IPERF, 2018). The SpeedTest results are showing an overall communication capabilities of the network with the outside world and accessing locations externally.

In the Figure 61, it can be observed that the tests yield a network throughput of approximately 16 Mbps at the DL and 3 – 8 Mbps UL. The latency is varying from 12 to 25ms with jitter of ~5ms. The latency deviation is due to different geographical test locations and server distances, which indicates that the communication with these values can be flawless in video calls, browsing the internet or even online gaming. It can be noted that the IP address of the UE is 192.188.0.2. This address belongs to the network previously-assigned by the S/PGW (192.188.0.0/24). The S/PGW has an IP address 192.188.0.1 and it can be used as a point to set up a server for exclusive testing of the mobile network performance, without any impact on the calculations from the outside world and the Internet.



OpenAirInterface 4G 38% 16:15

SPEEDTEST

1 RESULTS

TYPE	TIME	DOWNLOAD Mbps	UPLOAD Mbps	PING ms
	20/02/2018 16:15	16.49	3.37	12

OpenAirInterface 88% 12:24

SPEEDTEST

DOWNLOAD Mbps 16.2

UPLOAD Mbps 7.92

Ping 25ms | Jitter 5ms | Loss — %

Want to test without ads?

Remove Ads

GO

OpenAirInterface 95% 13:14

SPEEDTEST

17/04/18, 13:13

DOWNLOAD 16.3 Mbps Data used 19.1 MB

UPLOAD 2.85 Mbps Data used 2.62 MB

PING 29 ms

JITTER 5 ms

PACKET LOSS – %

Telenor Norge AS
Oslo

LTE OpenAirInterface

User Location
Lat: 59.9193 Lon: 10.7346

Internal IP: 192.188.0.2
External IP: ::ffff:158.36.118.16

Figure 61. SpeedTest results for the OpenAirInterface5G network, issued from the Huawei P9 Lite UE

For testing of the mobile network locally, the tool *iperf3* is used. One instance is initiated at the UE, and the server is running at the EPC listening on the S/PGW interface *gtp0* with IP 192.188.0.1 (see Appendix S). Iperf3 measures the throughput in one direction, from the client requests towards the server responses at the UE and from the server responses to the client requests at the EPC. The packets are sent via TCP/SCTP protocols and the measurements give the results depicted in Appendix T for the UE and in Appendix U for the EPC server side. After running iPerf for 10 minutes, with packet transmission interval of 1 second, the results are similar to the ones obtained from the SpeedTest observations (see Figure 62).

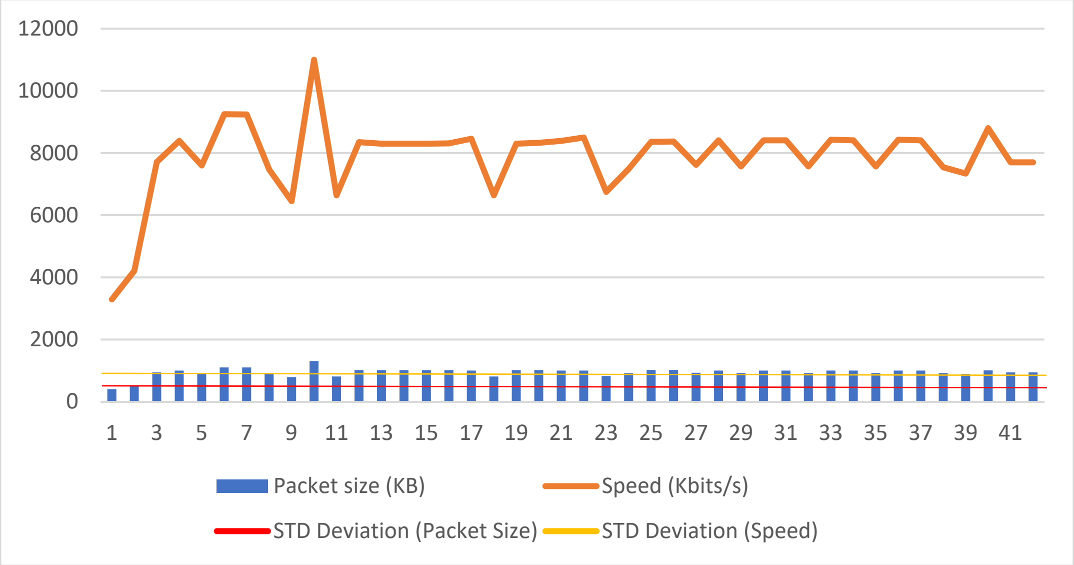


Figure 62. Diagram from iPerf3 observations

According to the diagram, the average packet size transmitted through the network is 950.26 KB, or 0.95 MB. The average speed is 7874.28 Kbit/s, which in both directions is 15.7 Mbit/s (similar to the SpeedTest results). The standard deviation in packet size is 141.9 KB and the standard deviation in speed is proportional and correlated to the packet size, or specifically 1225 Kbit/s. By comparing the two results, it can be stated that the performance of the network are not only conditioned by the OpenAirInterface5G system itself, but also many other factors are impacting the performance. The biggest constraint is the underlying hardware on which the mobile network is situated. If the CPU unit is stronger and has more cores available, then the frequency of transmission can be scaled to higher frequency bands, the sampling rate increased and with higher channel bandwidth of i.e. 25 MHz; as a result, the performance would be much higher. This suggests that the future virtualized mobile 5G networks will be directly related to the performance of the datacenters and the underlying hardware, as well as the availability of the resources and the simplicity of the deployment that can facilitate scaling.

6.2. EURECOM MIMO OpenAir Sounder (EMOS) for testing MIMO propagation

To test the MIMO propagation, the Eurecom MIMO Openair Sounder (EMOS) is utilized. As a part of the OpenAirInterface5G platform which allows real-time MIMO channel measurements synchronously over multiple users moving at vehicular speed, EMOS consists of a base station (BS) that continuously sends out a signaling frame, and one or more users that receive the frames to estimate the channel. The EMOS is using an OFDM modulated sounding sequence. The duration of one transmit frame is 2.667ms and it consists of a synchronization symbol (SCH), a broadcast data channel (BCH) comprising 7 OFDM symbols, a guard interval, and 48 pilot symbols used for channel estimation (see Figure 63). The pilot symbols are taken from a pseudo-random QPSK sequence defined in the frequency domain. The subcarriers of the pilot symbols are multiplexed over the four transmit antennas to ensure orthogonality in the spatial domain. The BCH contains the frame number of the transmitted frame that is used for synchronization among the UEs (EURECOM, 2018).

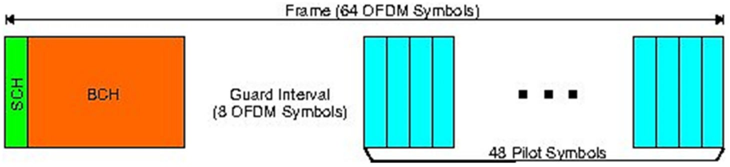


Figure 63. Frame structure of the OFDM sounding sequence (EURECOM, 2018)

Each UE first synchronizes to the BS using the SCH. It then tries to decode the data in the BCH. If the BCH can be decoded successfully, the channel estimation procedure is started. The channel estimation procedure consists of two steps. Firstly, the pilot symbols are de-rotated with respect to the first pilot symbol to reduce the phase-shift noise generated by the dual-RF CardBus/PCMCIA card. Secondly, the pilot symbols are averaged to increase the measurement SNR. The estimated MIMO channel is finally stored to disk. In order to conduct multi-user measurements, all the UEs need to be frame-synchronized to the BS. This is achieved by storing the frame number encoded in the BCH along with the measured channel at the UEs. This way, the measured channels can be aligned for later evaluations. The frame number is also used to synchronize the data acquisition between UEs. One measurement run (file) starts every 22.500 frames (60 sec) and is exactly 18.750 frames (50 sec) long (EURECOM, 2018).

To initialize the EMOS sounder, the lte-softmodem is compiled with the option *EMOS=1* and the real-time measurements are stored to a file (both for the UE and the eNB). The channel estimates are also deposited if the softmodem operation is compiled with the flag *–DEMOS_CHANNEL=1*. With the real-time data, it is feasible to apply machine learning and A.I techniques in order to obtain most optimal channel estimation model.

6.3. ITTI analyzer

OpenAirInterface5G supports implementation of MAC interface for Wireshark using UDP sockets, which allows Wireshark to separate MAC, RLC, PDCP and RRC packets. For the S1AP and GTP packets, there is no particular requirement for configuration, however the initiation of PCAP files is accordingly feasible. To properly set up the Wireshark, it is needed to adjust the heuristics in the preferences menu to correspond to UDP protocol, MAC-LTE, RLC-LTE and PDCP-LTE. The interfaces are then captured by the filters: *s1ap or lte_rrc or mac-lte or rlc-lte or pdcp-lte*. The L2 PDUs are transmitted to the local interface and a LTE packet dissection is saved as *"oai_l2l3.pcap"* file locally or in VCD format as *"oai_l2l3.vcd"*.

The *itti_analyzer* tool is used to analyze the exchanges between RRC<->S1AP, RRC<->PDCP, PDCP<->S1 in case there are problems within the domain of the mobile network and a surgical packet analysis is required. This is enabled when compiling the *lte-softmodem* using the flag ITTI_ENABLED (EURECOM, 2017).

7. VIRTUALIZATION AND DEPLOYMENT IN CLOUD

Hitherto, the deployment of the virtualized constituents of the OpenAirInterface5G mobile infrastructure are explicated in terms of deployment on top of OpenStack cloud. For full cloud integration, the OpenAirInterface5G needs to have its functionality translated so that OpenStack can understand the communication machineries. Since OpenAirInterface5G utilizes tunneling protocols as well as encapsulation from various protocols into TCP/SCTP and UDP, the communication can experience downgrade when the EPC is deployed on top of the cloud. Additionally, earlier versions of OpenStack do not support SCTP traffic to the deployed instances, which necessitates tunneling by default. Tunneling can increase the overall latency in the network traffic and can sometimes be an unreliable method of networking. By avoiding tunneling, the OpenAirInterface5G infrastructure can remove a layer of complexity and also a point of failure; this can be performed by integration of the virtual network function of OpenAirInterface5G into OpenStack using Heat orchestration (OPENSTACK, 2018) [OpenStack Heat Orchestration].

7.1. Deploying OpenAirInterface5G EPC core in OpenStack using Heat templates

Heat is the main project in the OpenStack Orchestration program. It implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code. A native Heat template format is evolving, but Heat also endeavors to provide compatibility with the AWS CloudFormation template format, so that many existing CloudFormation templates can be launched on OpenStack. Heat provides both an OpenStack-native ReST API and a CloudFormation-compatible Query API. A Heat template describes the infrastructure for a cloud application in a text file that is readable and writable by humans, and can be checked into version control, diffed, &c. Infrastructure resources that can be described include: servers, floating IPs, volumes, security groups, users, etc. Heat also provides an autoscaling service that integrates with the Telemetry service of OpenStack, so a scaling group can be included as a resource in a template. Templates can also specify the relationships between resources. This enables Heat to call out to the OpenStack APIs to create all of the desired infrastructure in the correct order to completely launch the application. Heat manages the whole lifecycle of the application - when a change to the infrastructure is desired, the template can be modified and used to update the existing stack. Heat knows how to make the necessary changes. It will delete all of the resources when the application is removed. Heat primarily manages infrastructure, but the templates integrate also with software configuration management tools such as Puppet and Chef (OPENSTACK, 2018) [OpenStack Heat orchestration] in order to enable automation.

For this purpose, an OpenStack Heat template is built (included in the Appendix V), in conjunction with the work of Swisscom Innovations and the OpenAirInterface5G open-source community (SWISSCOM, 2018). The template creates a router, a floating IP, an internal network, volumes, security groups, ports and instances in OpenStack. The EPC can be placed everywhere in the Internet even behind NATs, and there is no requirement for a second external network in OpenStack, so in principle any public OpenStack cloud can be used to deploy the core. In addition, S1 traffic between eNB and MME/SPGW is encrypted and eNBs are authenticated (OPENAIRINTERFACE, 2018).

Before deploying the template with Heat, a custom Ubuntu image is built. Since the heat template uses *OS::Heat::SoftwareDeployment*, the base image for the instances needs to have *os-collect-config/heat-config/os-apply-config* tools installed:

```
sudo apt-get install python-pip
sudo pip install virtualenv
virtualenv ~/dib-virtualenv
. ~/dib-virtualenv/bin/activate
pip install diskimage-builder
```

```
mkdir custom-image
cd custom-image
git clone https://git.openstack.org/openstack/tripleo-image-elements.git
git clone https://git.openstack.org/openstack/heat-templates.git
export ELEMENTS_PATH=tripleo-image-elements/elements:heat-templates/hot/software-
config/elements
diskimage-builder/bin/disk-image-create vm ubuntu os-collect-config os-refresh-config
os-apply-config heat-config heat-config-script -o ubuntu-xenial-os-config.qcow2
```

The QCOW2 image is then implemented in the OpenStack Alto cloud using the Glance module:

```
glance image-create --disk-format qcow2 --file ubuntu-$DIB_RELEASE-os-config.qcow2
ubuntu-$DIB_RELEASE-os-config
```

After the template initiates the network core, the MME, S/PGW and HSS need to be started manually in order to have the core network running and ready for the eNB to attach to it.

7.2. Using Kubernetes for orchestration of the container resources remotely in the cloud

The OpenAirInterface5G components can be separated and deployed in a form of microservices architecture, in a cluster. To achieve that, the previously clarified method of compiling the OpenAirInterface5G EPC modules need to be separately executed in distinct containers. Rationally, two to three containers can be built with two variations: MME+S/PGW in one container and the HSS in a separate container; MME, S/PGW and HSS all in separate containers. The constituents do not necessarily need to be in the same container network, but for the matter of expediency and simplicity, they are set as services and deployed in a cluster mode, which is the crucial for the microservices architecture. With Kubernetes, the container network is served using the Flannel SDN unit. Flannel performs an IP-in-IP connectivity and creates a network overlay, which as previously shown, adds a slight overhead in the networking latency and computational resource demand. To address this concern, the Kubernetes daemon has the Flannel SDN replaced with Calico. Consequently, the container network has a full Layer-3 networking enabled and can communicate to the outside world without any overhead caused by additional network overlays.

In addition to Kubenetes, Calico's driver is also integrated with the OpenStack's Neutron networking module. The Calico plugin is certified by Mirantis (which is the same version of OpenStack installed), it is available for implementation from the Fuel Plugins Catalog. Installation of the driver is straightforward:

```
fuel plugins --install calico-fuel-plugin-<version>.noarch.rpm
```

Subsequently, the OpenStack environment is re-deployed for the settings to take effect. At the Fuel master node, in the Neutron networking tab the setting *"Use Calico Virtual Networking"* is selected. All the other configuration settings are preserved, since the OpenStack is the functional cloud infrastructure at the OsloMet University and the VLAN/IP settings must remain unchanged. In Figure 64, the process of communication between the master node and the deployed remote workloads is performed with Calico through OvS virtual bridges. The *eth0* interface of the physical network card at the master node has a virtual bridge which establishes direct Layer-2 communication. The OvS bridge can then be used for relaying the routed Layer-3 traffic or as a reference point for L2TP tunneling to the OpenStack cloud.

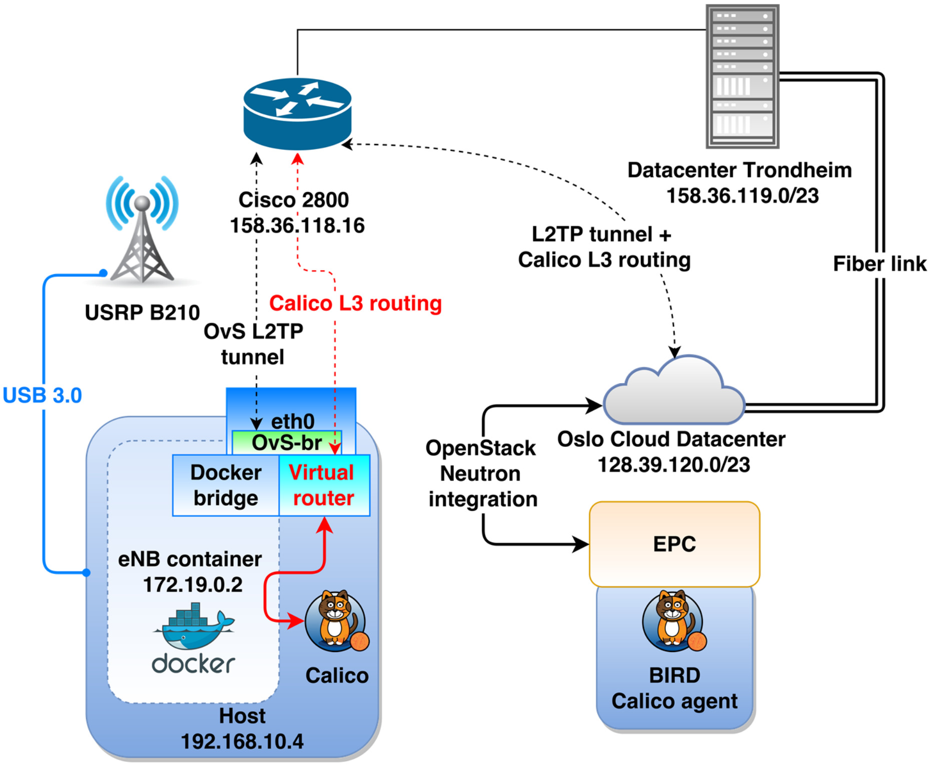


Figure 64. Using the Calico networking driver for OpenStack and integration with Neutron

It should be noted that in order to have fully-operation cluster network, all the Calico functions should work properly. On the OpenStack controller nodes, the BGP BIRD route reflector establishes sessions to all the compute nodes and the Neutron service is running and has initialized the Calico ML2 mechanism driver for inter-VM connectivity. On each compute node, the Calico Felix agent is correctly configured and running, which can enable further establishing of BGP sessions to the route reflector on the controller nodes. In this case, the Open vSwitch driver works in coordination with the Calico driver, and doesn't have to be removed. Calico can also route IPv6 traffic, and since the datacenter is connected also via IPv6, an IP address range is deducted to the OpenStack network for IPv6 networking. In conclusion, the remote workloads of the eNB constituents (a single eNB container or C-RAN splits into RRU and BBU) can communicate with the EPC in the cloud using the BGP protocol, securely. The MME can only be accessed by the eNB and the container mesh in the EPC cluster, but nothing else. This applies also to the HSS and S/PGW. With such isolation, the security of the whole infrastructure introduces additional layers of abstraction and is drastically increased.

To install the aforementioned elements, a YAML file *oai.yml* is formed. Each module belongs to a different Pod, which is part of a distinct Namespace. All the namespaces form a single Daemon set, in the case if the MME, HSS and S/PGW are compiled and run in daemon mode. The process is exactly analogous to the one described previously, about deploying Kubernetes microservices application where each EPC component is implemented in a single container instead in separate. In this case, the control over the comprising elements is more detailed, where each daemon set has a detailed control over the deployed Pods. The EPC thus is instantiated in microservices mode on the instance running on OvS router 128.39.121.19 in the OpenStack cloud, getting an IP address 128.39.121.27 and a private IP 10.0.0.27 (see Figure 65).

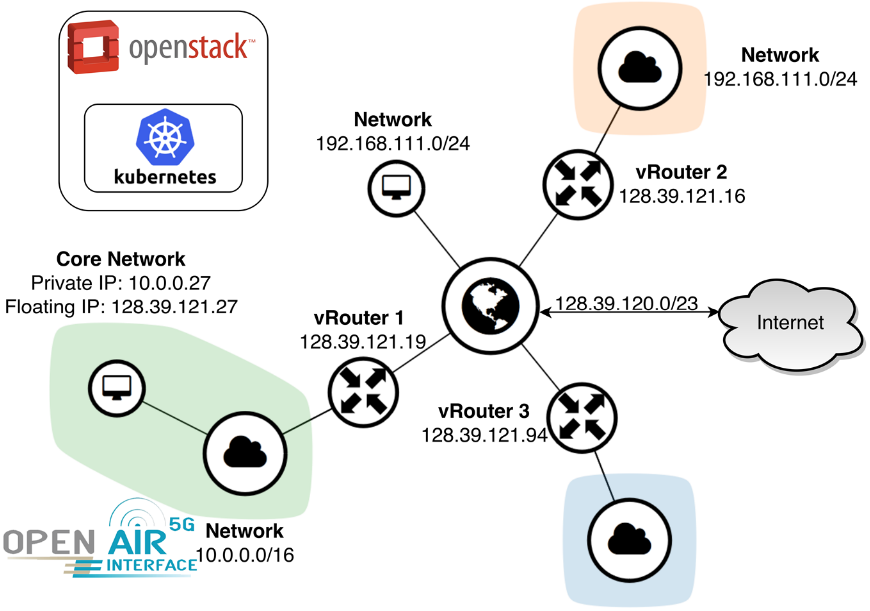


Figure 65. Deployed Kubernetes EPC cluster on top of OpenStack cloud

8. DISCUSSION

The main goal every operator targets to achieve is user satisfaction and good technology experience. With the purpose of providing the declared qualities, the service provider must ensure that the deployed infrastructure is robust and reliable, with minimal to no downtime although offering high level of performance. To facilitate these excellences, every observable impediment needs to be addressed and resolved. In this case, several repercussions surfaced and required special attention.

Unambiguously, one of the biggest downside of the OpenAirInterface5G network are the special requirements for real-time operation. Any processing fluctuations at the host in terms of CPU vacillations can be detrimental to the stability of the network. The eNB runs very sensitive processes and therefore, any discrepancy in the CPU frequency can lead to misapprehending behavior. As pointed out, the CPU scaling and power control are disabled for the unit to operate at maximum performance. However, some recent events in the cyber-security domain have led Intel to produce patches for vulnerabilities that encompassed all current CPU architectures. In particular words, the patches for the Spectre/Meltdown vulnerabilities (COLDEWEY, Devin, 2018) have a substantially negative impact on the performance of the CPU units, and generally thus it drastically affects the function of the currently installed OpenAirInterface5G network. The patches decrease the CPU processing efficiency up to 30% if the units are older than 2 years. For addressing this issue, all recent Linux updates regarding security patching are removed and earlier version of Ubuntu 16 is retained. Also, the eNB operation requires very strong CPU units with as more cores as possible. Testing a full MIMO propagation with multiple eNBs and users can be difficult if the underlying host is not rich in resources and the technology is not innovative.

Furthermore, the OpenAirInterface5G mobile network requires mapping of GTP-U modules from the Linux kernel and indeed requires the presence of such modules. Deployment of the core network in the cloud can be challenging if the modules are missing. Most often, the OpenStack Linux images for the VM instances are being created without the GTP modules, and thus the running of the core network is impossible. To tackle this issue, custom Linux images are produced and the GTP modules included within the scope. The same issues may persist even with the usage of containers, because the containers are mapping modules from the Linux kernel when running an application. Without the appropriate kernel modules and libraries, the application will be unable to run. Given that the host operating system is resolute, the upside of the containerized environment opens endless possibilities for manipulation of the mobile network constituents in terms of scaling, resource localization, upgrades and continuous integration approaches. With fully-functional splits, a clustered environment can be easily scaled; i.e., Kubernetes can assign as many replicas as desired for the particular components. For example, the HSS database can be scaled up to massive clusters in case where the mobile network becomes oversaturated with UE requests. The scaled cluster can then load-balance between requests and increase the database replicas in order to serve the growing requests coming from the users. This minor action can have a substantial impact on the performance of the virtualized mobile network. In addition to this, if the HSS database resides on a SSD or NVMe disk; with their very high access and writing speeds, the mobile network performance can avoid bottlenecks in terms of I/O tailbacks that can impact the performance severely in case if the database is situated on a standard HDD disk. When the container clusters are auto-scaled, usually the low performance hard disks can be the culprit for downtime and degraded operation due to very low read/write speeds. For comparison, a standard 7200rpm HDD disk has a read/write speed of ~80-150 MB/s, which is a theoretical value. In practice, those speeds are much slower especially when i.e. 4K video is being recorded or streamed. With speeds of ~0.8-0.9 MB/s for read/write operations of 4K video, the bottlenecks are too obvious to be ignored. Disregarding the fact that the network speed can go up to 20 MB/s, the disk performance will not be able to keep up with the data transfer through the network. The database access can then be severely restricted and the users can experience very bad performance over the mobile network, i.e. when playing VR games online. On the other hand, if the database is placed on a SSD disk, the read/write speeds that can be achieved in that case can vary from ~150-250 MB/s. This exceeds the overall requirement for serving users a 4K video streamline for any purpose and the end user experience will be greatly enhanced.

According to the results from the tests, the operation of the network is also bottlenecked by the USB connection of the USRP eNB, coupled with the local 100Mbps Ethernet link provided to the EPC (which is not fiber), as well as firewalls, external gateways and redistributing routes through various routing protocols. A resolution for this issue would encompass utilization of a better software-defined radio peripheral such as the USRP X310 (ETTUS RESEARCH, 2018) [USRP X310 High-performance Software-Defined Radio (SDR)], which is the PCI-Express version of the hardware. Through the PCI-e lane, the speeds of accessing the hardware would be much higher in comparison to the USB bus. Given this headroom, the USRP X310 implements two daughterboards with 160 MHz channel bandwidth each, allowing much higher sampling rates and higher speeds.

One factor that plays a crucial role in the impediment of the experimentation is the environment in which the network is tested; the experiments are being subjected to interference from the commercial mobile operators, and without a proper isolated Faraday room, the UL (Uplink) channel varies vigorously, giving test results always differently (Figure 61). This is due to the usage of band 3 without lawful regulation, which commercial operators use for existing networks and mobile access. If an isolated environment is provided, then the test results can be more reliable and straightforward, without any disturbances and external influences. However, besides these obstacles, the mobile network is established successfully and operates in a stable manner.

9. CONCLUSION

In this thesis the concepts of virtualization of a mobile network are introduced and providing a network slice is hence attained; vast amount of devices and users can be served more efficiently by scaling the network functions thus. Many of the emerging devices on the Internet will require specific sets of parameters and network resources, which can be divided fairly and according to the regulations of the service provider. Expected fields that are interested in the current developments are: the commercial users, industry, medical sphere, Internet of Things realm, self-driving cars, law enforcement and military, etc. With the exponential rise of the interest in connectivity, the represented concept of virtualization can prominently contend with the inward challenges; explicitly, by utilizing the potential of Calico's policy-based networking, establishing container isolation and thus network slices on-demand is absolutely achievable while the focus on the security is being maintained. Indeed, the establishment of a portable softwareized mobile infrastructure is achieved and the opportunities it invites are discussed accordingly.

To also conclude the practicability of deployment of a portable virtualized mobile infrastructure, initially the shortcomings of such exertion are highlighted. Not only that the particular desirability is achieved, but additional alleviating contrivances are instituted. Without orchestration and automation, the utilization of such setup can be exceptionally challenging and complex in vast environs; by using Kubernetes or OpenStack Heat, the difficulty of realizing the coveted action is substantially reduced. This refers to the repeatability of deployment, which by utilizing the power of containerization and immutable distributions, enables the user to straightforwardly re-deploy the virtualized groundwork in the same state as when it was created initially. Despite the inadequacies, the vast benefit offered with utilization of the power of open-source software and its availability empowers the accomplishment of the initial stage of 5G virtualization; presenting the opportunity of paving a new way for implementing the succeeding technologies. Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are the key driving concepts of establishing a successful base and amalgamation of the virtualized environments. Using proper orchestration tools, the control and monitoring over the softwareized infrastructure can be unsmilingly alleviated, countenancing automation mechanisms to take control over the life of the setup. Crucially, the benefit the virtualization offers has multiple dimensions, as in terms of value for the society, but also pruning to gains for the whole humanity as a civilization. With reduction of the physical hardware and virtualization of the constituents, the overall power consumption of the service providers will be diminished, which can help tackling the problem with the climate changes caused by the humans and specifically data-center operations. The impact can be reduced further with adequate optimization of the Software-defined networks and how much resources the future 5G infrastructure will use.

9.1.Future work

Adaptable Network Slicing is a concept that follows after the accomplishment of this work. With successful isolation of different network modules, it is then feasible to establish various services for numerous verticals. For example, critical infrastructure can be completely isolated from the IoT platforms, and then merged again when the requirements are demanded. The adaptability refers to the flexible mode of assignment of network slices using orchestration. With this, the future of the 5G networking relies on self-organizing entities and artificial intelligence for prediction of such necessities. In order for this to be achieved, the OpenAirInterface5G community works intensively on creating an environment that can support splitting of the workloads into few entities. Namely, the concept of *Centralized/Cloud – Radio Access Network (C-RAN) splits* indicates the necessity to split the eNB operation into Baseband Units (BBU) and multiple RRU (Remote Radio Units). With this mode of operation, a single eNB base station can then serve multiple verticals and provide various slices with a single hardware entity. In particular, the central processing of the eNB will be able to accommodate multiple RRUs, which are dedicated for processing of the distributed antennas. The antennas will then be the only elements accessible by the UEs, while the eNB resides in the cloud close to the EPC. Logically, the only delay constraint that can be confronted in this situation is the one incurred by the radio propagation, because the eNB communicates with the EPC with a direct link in a same datacenter or a distributed cloud sited in close proximity to the datacenter where the EPC is residing.

In the future, the scaling of the 5G infrastructure can become arduous and yield massive dimensions due to the requirement of the ever-growing demand for resources and devices that will inhabit the ecosphere of the Internet connectivity. The emergence of IoT (Internet of Things) devices, mMTC (Massive Machine Type Communication), interworking with sensor networks, satellite systems, Wi-Fi and more, will induce huge amounts of traffic that converges ultimately fast in time domain. Information exchange in that situation is transpiring very fast and the requirement for scaling up the infrastructure has to be automated. The requisites for this solution will impose necessity for smart monitoring systems of the infrastructure, which can collect data for the particular requirements for scaling. Accordingly, the self-organizing system powered by artificial intelligence can then vigorously scale the network, or migrate the resources to the position where they are mostly required in a particular moment. In any case, the impending operations in that domain will require more than human-only intervention, which primes to further prospects for development and research of cutting-edge technologies and methods.

REFERENCES

3GPP. 2007-2012. *Specification 36.302: Services provided by the physical layer, v8.0.0*. Valbonne: 3rd Generation Partnership Project.

3GPP. 2009. *Specificaiton TS 36.213: E-UTRA Physical Layer Procedures*. [online]. [Accessed 2017]. Available from World Wide Web: <http://www.etsi.org/deliver/etsi_ts/136200_136299/136213/08.08.00_60/ts_136213v080800p.pdf>

3GPP. 2010. *Specification 23.402: Architecture enhancements for non-3GPP accesses (Release 9)*. [online].

3GPP. 2010. *Specification 29.061: Group Core Network and Terminals; Interworking Between the Public Land Mobile Network (PLMN) supporting packet based services and Packet Data Networks (PDN); Release 9; v9.3.0*. [online].

3GPP. 2010. *Specification 29.303 v9.1.0: 3GPP; Technical Specification Group Core Network and Terminals; Domain Name System Procedures; Stage 3, Release 9*. [online].

3GPP. 2011. *Specification 36.912: LTE-Advanced; 3GPP Release 10*. [online].

3GPP. 2013. *LTE-Advanced*. [online]. [Accessed March 2018]. Available from World Wide Web: <http://www.3gpp.org/technologies/keywords-acronyms/97-lte-advanced>

3GPP. 2015. *Specification 24.301: Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3*. Valbonne: 3rd Generation Partnership Project.

3GPP. 2015. *Specification 29.272: Evolved Packet System (EPS); Mobility Management Entity (MME) and Serving GPRS Support Node (SGSN) related interfaces based on Diameter protocol*. [online].

3GPP. 2015. *Specification 29.274: 3GPP Evolved Packet System (EPS); Evolved General Packet Radio Service (GPRS) Tunnelling Protocol for Control plane (GTPv2-C); Stage 3*. [online].

3GPP. 2015. *Specification 33.401: 3GPP System Architecture Evolution (SAE); Security architecture*. [online].

3GPP. 2017. *3GPP*. [online]. [Accessed 24 November 2017]. Available from World Wide Web: <http://www.3gpp.org>

3GPP. 2017. *3rd Generation Partnership Project: Release 16*. [online]. [Accessed December 2017]. Available from World Wide Web: <http://www.3gpp.org/release-16>

3GPP. 2017. *Specification 23.110: Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; Universal Mobile Telecommunications System (UMTS) access stratum; Services and functions (3GPP TS 23.110 version 14)*. Valbonne: 3rd Generation Partnership Project.

3GPP. 2017. *Specification 23.501: System architecture for the 5G system; v1.3.0*. Valbonne: 3rd Generation Partnership Project.

3GPP. 2017. *Specification 25.892: Feasibility study for Orthogonal Frequency Division Multiplexing (OFDM) for UTRAN enhancement*. Valbonne: 3rd Generation Partnership Project.

3GPP. 2017. *Specification 36.201: Evolved Universal Radio Access (E-UTRA); LTE physical layer; General description*. Valbona: 3rd Generation Partnership Project.

5GPPP. 2016. *View on 5G Architecture*. [online]. Available from World Wide Web: <https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-5G-Architecture-WP-For-public-consultation.pdf>

ADRIO COMMUNICATIONS LTD. 2017. *Radio-Electronics (LTE OFDM, OFDMA, SC-FDMA and Modulation)*. [online]. [Accessed February 2018]. Available from World Wide Web: <http://www.radio-electronics.com/info/cellulartelecomms/lte-long-term-evolution/lte-ofdm-ofdma-scfdma.php>

ADRIO COMMUNICATIONS LTD. 2017. *Radio-Electronics (Quadrature Amplitude Modulation)*. [online]. [Accessed February 2018]. Available from World Wide Web: <http://www.radio-electronics.com/info/rf-technology-design/quadrature-amplitude-modulation-qam/what-is-qam-tutorial.php>

ADVA OPTICAL NETWORKING. 2018. *Network Function Virtualization*. [online]. [Accessed February 2018]. Available from World Wide Web: <https://www.advaoptical.com/en/products/technology/what-is-nfv>

ALI-YAHIYA, Tara. 2011. *Understanding LTE and its Performance*. New York: Springer-Verlag.

ANDREW, A. 2015. *Revolution Wi-Fi*. [online]. [Accessed March 2018]. Available from World Wide Web: <http://www.revolutionwifi.net/revolutionwifi/2015/3/how-ofdm-subcarriers-work>

ANKIT, D. P. 2013. Multiple Antenna & Diversity: Smart Antennas. *International Journal of Scientific and Research Publications*. **3**(4).

ARAÚJO, D. C., T. MAKSYMYK, A. L. F. DE ALMEIDA et al. 2016. Massive MIMO: Survey and Future Research Topics. *IET Communications*. **10**(15), pp.1938-1946.

AVIAT NETWORKS. 2017. *Diversity Scheme*. [online]. [Accessed February 2018]. Available from World Wide Web: <http://aviatnetworks.com/tag/diversity-scheme/>

AZIZ, A. M. 2009. A multiple-antenna diversity scheme for reception of fading signals in noise. New Cairo: IEEE National Radio Science Conference, pp.1-10.

BERNSTEIN, Ben. 2015. *Twistlock*. [online]. [Accessed 22 May 2017]. Available from World Wide Web: <https://www.twistlock.com/2015/08/06/immutable-infrastructure-containers-and-security/>

BIEHLE, G. 2016. *The MCAT Physics Book*. West Hollywood: Nova Press.

BOUDRIGA, N. 2010. *Security of Mobile Communicaitons*. Boca Raton, FL: CRC Press.

BRYZEK, Michael. 2014. *Immutable Infrastructure with Docker and EC2*. [online]. [Accessed 22 May 2017]. Available from World Wide Web: <https://www.youtube.com/watch?v=GaHzdqFithc>

C. S. ARENAS, John, Torsten DUDDA, and Laetitia FALCONETTI. 2017. Ultra-low Latency in Next Generation LTE Radio Access. *In*: *SCC 2017; 11th International ITG Conference on Systems, Communications and Coding; Proceedings of*. Hamburg: VDE.

CHANG, Zheng, Zhenyu ZHOU, Sheng ZHOU et al. 2016. *Towards Service-oriented 5G: Virtualizing the Networks for Everything-as-a-Service*. [online]. [Accessed May 2017]. Available from World Wide Web: <https://arxiv.org/pdf/1604.01739.pdf>

CHRYSSOMALLIS, M. 2000. Smart Antennas. *IEEE Antennas and Propagation Magazine*. **42**(3), pp.129-136.

CISCO. 2008. *Multipath and Diversity*. [online]. [Accessed February 2017]. Available from World Wide Web: <https://www.cisco.com/c/en/us/support/docs/wireless-mobility/wireless-lan-wlan/27147-multipath.html>

CISCO. 2017. *Configuraiton of Virtual Router Redundancy Protocl (VRRP) on ISA500 Series Integrate Security Applications*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://supportforums.cisco.com/t5/small-business-support-documents/configuration-of-virtual-router-redundancy-protocol-vrrp-on/ta-p/3171571>

COCHRAN, W.T., J.W. COOLEY, D.L. FAVIN et al. 1967. What is Fast Fourier Transform. *In*: *Proceedings of the IEEE*. IEEE.

COLDEWEY, Devin. 2018. *Tech Crunch*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://techcrunch.com/2018/03/15/intel-announces-hardware-fixes-for-spectre-and-meltdown-on-upcoming-chips/>

COREOS. 2018. *CoreOS - Etcd*. [online]. [Accessed February 2018]. Available from World Wide Web: <https://coreos.com/etcd/docs/latest/getting-started-with-etcd.html>

COX, C. 2014. *An Introduction to LTE - LTE, LTE-Advanced, SAE, VoLTE and 4G Mobile Communications*. United Kingdom: John Wiley & Sons Ltd.

DIGHE, P. A., R. K. MALLIK, and S. S. JAMUAR. 2003. Analysis of Transmit-Receive Diversity in Rayleigh Fading. *IEEE Transactions on Communications*. **51**(4), pp.694-703.

DIGITALOCEAN. 2018. *Deploying and Scaling Microservices in Kubernetes*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://www.digitalocean.com/community/tutorials/webinar-series-deploying-and-scaling-microservices-in-kubernetes>

DOBKIN, D. M. 2011. *RF Engineering for Wireless Networks: Hardware, Antennas and Propagation*. San Diego, California: Elsevier.

DOCKER. 2018. *Docker Hub: Repository - Bruno Dzogovic*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://hub.docker.com/u/brunodzogovic/>

DOCKER, INC. 2017. *Docker*. [online]. [Accessed 20 October 2017]. Available from World Wide Web: <http://www.docker.com/>

DOCKER, INC. 2018. *Docker Cloud*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://docs.docker.com/docker-cloud/>

DOCKER, INC. 2018. *Docker Hub*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://docs.docker.com/docker-hub/>

DOCKER, INC. 2018. *Docker Storage*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://docs.docker.com/storage/>

DOCKER, INC. 2018. *Overview of Docker Compose*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://docs.docker.com/compose/overview/>

ELKHODR, M., Q. F. HASSAN, and S. SHAHRESTANI. 2017. *Networks of the Future: Architectures, Technologies, and Implementations*. USA: Chapman & Hall/CRC Press.

ELSALLAL, M. W., I. HOOD AND MCMICAHEL, and T. BUSBEE. 2016. 3D Printed Material Characterization for Complex Phased Arrays and Metamaterials. *Microwave Journal*. **59**(10).

ETSI. 2017. *The European Regulatory Environment for Radio Equipment and Spectrum, a brochure*. [online].

ETTUS. 2018. *Ettus Research*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://www.ettus.com/>

ETTUS. 2018. *USRP Hardware Driver and USRP Manual*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://files.ettus.com/manual/page_coding.html>

ETTUS RESEARCH. 2018. *USRP X310 High-performance Software-Defined Radio (SDR)*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://www.ettus.com/product/details/X310-KIT>

EURECOM. 2017. *EURECOM*. [online]. [Accessed October 2017]. Available from World Wide Web: <www.eurecom.fr/en>

EURECOM. 2017. *Gitlab*. [online]. [Accessed February 2018]. Available from World Wide Web: <https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/IttiAnalyzer>

EURECOM. 2018. *Eurecom: MIMO Channel Sounder (EMOS)*. [online]. [Accessed March 2018]. Available from World Wide Web: <http://openairinterface.eurecom.fr/mimo-channel-sounder-emos>

FAJARDO, V., J. ARKKO, J. LOUGHNEY, and G. ZORN. 2012. *RFC6733 - Diameter Base Protocol*. Internet Engineering Task Force.

FARRIS, I., T. TALEB, H. FLINCK, and A. IERA. 2017. Providing ultra-short latency to user-centric 5G applications at the mobile network edge. *Transactions on Emerging Telecommunications Technologies (ETT)*. **e3169**(Special Issue Article), pp.1-13.

FISCHER, P., S. YI, S. CHUN, and Y. LEE. 2011. LTE User Plane Protocols. *In*: S. SESIA, I. TOUFIK, and M. BAKER, (eds). *LTE - The UMTS Long-Term Evolution: From Theory to Practice*, Chichester, UK: John WIley & Sons, Ltd., pp.87-120.

FORENZA, A., A. PANDHARIPANDE, H. KIM, and R. W. HEATH. 2005. Adaptive MIMO transmission scheme: exploiting the spatial selectivity of wireless channels. 2005 IEEE 61st Vehicular Technology Conference, pp.3188-3192.

FOSBERG, D., G. HORN, W-D. MOELLER, and V. NIEMI. 2013. *LTE Security*. Chichester, UK: John WIley & Sons, Ltd.

GAMAGE, H., N. RAJATHEVA, and M. LATVA-AHO. 2016. High PAPR Sequence Scrambling for Reducing OFDM Peak-to-Average Power Ratio. *In*: *European Wireless 2016; 22th European Wireless Conference; Proceedings of*. Oulu, Finland: VDE.

GARCIA, M. and C. OBERLI. 2009. Intercarrier Interference in OFDM: A General Model for Transmissions in Mobile Environments with Imperfect Synchronization. *EURASIP Journal on Wireless Communications and Networking*.

GHOSH, A., J. G. ANDREWS, R. MUHAMED, and J. ZHANG. 2011. *Fundamentals of LTE*. Prentice Hall Press Upper Saddle River, NJ, USA.

GOLBON-HAGHIGHI, M. H. 2016. Beamforming in Wireless Networks. *In*: H. K. BIZAKI, (ed). *Towards 5G Wireless Networks - A Physical Layer Perspective*, London: InTech Open Ltd., pp.163-199.

GOLDBLATT, Robert. 1987. *Orthogonality and Spacetime Geometry*. New York: Springer-Verlag.

HALE, J. S., L. LI, C. N. RICHARDSON, and G. N. WELLS. 2017. Containers for Portable, Productive, and Performant Scientific Computing. *Computing in Science & Engineering*. **19**(6), pp.40-50.

HANLEN, L. and M. FU. 2006. Wireless communication systems with-spatial diversity: a volumetric model. *IEEE Transactions on Wireless Communications*. **5**(1), pp.133-142.

HARTMANN, C. 2017. *Radio Resource Management in Cellular F/TDMA Smart Antenna Systems*. Munich, Germany : Herbert Utz Verlag Gmbh.

HASHICORP. 2018. *Terraform*. [online]. [Accessed February 2018]. Available from World Wide Web: <https://www.terraform.io/intro/vs/chef-puppet.html>

HASHICORP. 2018. *Vault*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://www.vaultproject.io>

HEWLETT-PACKARD. 2018. *HP ProCurve Enterprise 2900 Switch Series*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-c01840758>

HINDLE, P. 2018. *Microwave Journal*. [online]. [Accessed 25 February 2018]. Available from World Wide Web: <http://www.microwavejournal.com/articles/29572-antenna-technologies-for-the-future?page=1>

HONG, W., Z. H. JIANG, C. YU et al. 2017. Multibeam Antenna Technologies for 5G Wireless Communications. *IEEE Transactions on Antennas and Propagation*. **65**(12), pp.6231-6249.

HO, M-J., G. L. STUBER, and M. D. AUSTIN. 1998. Performance of Switcherd-Beam Smart Antennas for Cellular Radio Systems. *IEEE transactions on Vehicular Technology*. **47**(1), pp.10-19.

HUAWEI. 2018. *Huawei P9 Lite Phone*. [online]. [Accessed April 2018]. Available from World Wide Web: <https://consumer.huawei.com/en/phones/p9-lite/>

HUNT, A., A. DEGABRIELE, A. ROTH et al. 2016. Effects of channel environment on timing advance for mobile device positioning in long-term evolution networks. Pacific Grove, CA: 50th Asilomar Conference on Signals, Systems and Computers, pp.643-647.

IETF. 2003. *Diameter Base Protocol standard*. [online].

IETF. 2007. *SCTP - Stream Control Transmission Protocol*. [online].

IETF. 2011. *Standard RFC6071: IP Security (IPSec) and Internet Key Exchange (IKE) document roadmap*. [online].

INFLUXDATA. 2017. *InfluxData*. [online]. [Accessed 22 May 2017]. Available from World Wide Web: <https://www.influxdata.com/project/5g-test-network-and-influxdb/>

IPERF. 2018. *iPerf3*. [online]. [Accessed April 2018]. Available from World Wide Web: <https://iperf.fr/>

KADIR, M. F. A., M. K. SUAIDI, M. Z. A. ABD AZIZ et al. 2008. Polarization Diversity in Wireless MIMO Systems. Putrajaya: 2008 6th National Conference on Telecommunication Technologies and 2008 2nd Malaysia Conference on Photonics, pp.128-131.

KEITHLEY INSTRUMENTS. 2008. *The Multi-Path Problem*. [online]. [Accessed February 2017]. Available from World Wide Web: <www.keithley.com>

KUBERNETES. 2017. *Kubernetes*. [online]. [Accessed 20 October 2017]. Available from World Wide Web: <https://kubernetes.io>

KUMAR, S. 2017. *3G4G Wireless Resource Center*. [online]. [Accessed February 2017]. Available from World Wide Web: <http://www.3g4g.co.uk/Tutorial/SK/sk_rach_procedure.html>

LARSSON, E. G. and L. VAN DER PERRE. 2017. Massive MIMO for 5G. *IEEE 5G Tech Focus*. **1**(1), pp.(online: https://5g.ieee.org/tech-focus/march-2017/massive-mimo-for-5g).

LAURIDSEN, Mads, Lucas CHAVARRIA GIMENEZ, Ignacio RODRIGUEZ et al. 2017. From LTE to 5G for Connected Mobility. *IEEE Communications Magazine*. **55**(3), pp.156 - 162.

LI, Y. and N. R. SOLLENBERGER. 1999. Adaptive Antenna Array Systems for OFDM Systems with Cochannel Interference. *IEEE Transactions on Communications*. **47**(2), pp.217-229.

LOZANO, A. and N. JINDAL. 2010. Transmit Diversity vs. Spatial Multiplexing in Modern MIMO Systems. *IEEE Transactions of Wireless Communications*. **9**(1), pp.186-197.

MIETZNER, J., R. SCHOBER, L. LAMPE et al. 2009. Multiple-Antenna Techniques for Wireless for Wireless Communications - A Comprehensive Literature Survey. *IEEE Communications Surveys and Tutorials*. **11**(2).

MOLISCH, A. F. 2011. *Wireless Communications*. United Kingdom: John Wiley & Sons Ltd.

NATIONALINSTRUMENTS. 2018. *National Instruments*. [online]. [Accessed March 2018]. Available from World Wide Web: <http://www.ni.com/en-no.html>

OAI. 2017. *OpenAirInterface*. [online]. [Accessed October 2017]. Available from World Wide Web: <www.openairinterface.org>

ONGARO, D. and J. OUSTERHOUT. 2014. In Search of an Understandable Consesus Algorithm. Philadelphia, USA: Proceedings of USENIX ATC '14.

OPENAIRINTERFACE. 2018. *OpenAirInterface: 5G Software Alliance for Democartising Wireless Innovation*. [online]. [Accessed March 2018]. Available from World Wide Web: <http://www.openairinterface.org/>

OPENNETWORKFOUNDATION. 2018. *Software-Defined Networking*. [online]. [Accessed February 2018]. Available from World Wide Web: <https://www.opennetworking.org/sdn-definition/>

OPENSTACK. 2017. *OpenStack*. [online]. [Accessed October 2017]. Available from World Wide Web: <https://www.openstack.org/>

OPENSTACK. 2018. *OpenStack Heat*. [online]. [Accessed April 2018]. Available from World Wide Web: <https://wiki.openstack.org/wiki/Heat>

OPENVSWITCH. 2018. *Open vSwith*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://www.openvswitch.org/>

PALAT, S. and P. GODIN. 2011. Network Architecture. *In*: S. SESIA, I. TOUFIK, and M. BAKER, (eds). *LTE - The UMTS Long-Term Evolution: From Theory to Practice*, Chichester, UK: John Wiley & Sons, Ltd., pp.25-55.

PARVEZ, I., A. RAHMATI, I. GUVENC et al. 2017. *Arxiv.org: A Survery on Low Latency Towards 5G: RAN, Core Network and Cachin Solutions*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://arxiv.org/pdf/1708.02562.pdf>

PFSENSE. 2018. *Pfsense Open-Source Security*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://www.pfsense.org/>

POOLE, I. 2017. *Radio Electronics*. [online]. [Accessed February 2017]. Available from World Wide Web: <http://www.radio-electronics.com/info/cellulartelecomms/lte-long-term-evolution/physical-logical-transport-channels.php>

PUPPET. 2018. *Puppet*. [online]. [Accessed February 2018]. Available from World Wide Web: <https://puppet.com/>

RF WIRELESS WORLD. 2012. *Physical Control Format Indicator Channel (PCFICH)*. [online]. [Accessed February 2017]. Available from World Wide Web: <http://www.rfwireless-world.com/Terminology/LTE-PCFICH-Physical-Control-Format-Indicator-Channel.html>

RF WIRELESS WORLD. 2012. *Physical Random Access Channel (PRACH)*. [online]. [Accessed February 2017]. Available from World Wide Web: <http://www.rfwireless-world.com/Terminology/LTE-PRACH-Physical-Random-Access-Channel.html>

RF WIRELESS WORLD. 2012. *Physical Uplink Control Channel (PUCCH)*. [online]. [Accessed February 2017]. Available from World Wide Web: <http://www.rfwireless-world.com/Terminology/LTE-PUCCH-Physical-Uplink-Control-Channel.html>

RF WIRELESS WORLD. 2012. *Physical Uplink Shared Channel (PUSCH)*. [online]. [Accessed February 2017]. Available from World Wide Web: <http://www.rfwireless-world.com/Terminology/LTE-PUSCH-Physical-Uplink-Shared-Channel.html>

RF WIRELESS WORLD. 2012. *Physucal Multicast Channel (PMCH)*. [online]. [Accessed Februrary 2017]. Available from World Wide Web: <http://www.rfwireless-world.com/Terminology/LTE-PMCH-Physical-Multicast-Channel.html>

ROHLING, Hermann. 2011. *OFDM - Concepts for Future Communication Systems*. Berlin: Springer-Verlag Berlin Heidelberg.

RUSSELL, Travis. 2016. *LTE Signaling with DIAMETER*. United States: McGraw-Hill Education.

SÄLZER, T., D. GESBERT, C. VAN RENSBURG et al. 2011. Multiple Antenna Techniques. *In*: S. SESIA, I. TOUFIK, and M. BAKER, (eds). *LTE - The UMTS Lont-Term Evolution: From Theory to Practice*, Chichester, UK: John Wiley & Sons, Ltd., pp.249-277.

SAUTER, Martin. 2014. *From GSM to LTE-Advanced - An Introduction to Mobile Networks and Mobile Broadband*. Chichester: John Wiley & Sons, Ltd.

SHAFIK, R. A., S. RAHMAN, R. AHM. ISLAM, and N. S. ASHRAF. 2006. On the Error Vector Magnitude as a Performance Metric and Comparative Analysis. Peshawar, Pakistan: 2nd International Conference on Emerging Technologies, IEEE-ICET.

SPEEDTEST. 2018. *Speed Test*. [online]. [Accessed April 2018]. Available from World Wide Web: <http://www.speedtest.net/>

SRAR, J. A., K. S. CHUNG, and A. MANSOUR. 2010. Adaptive array beamforming using a combined LMS-LMS algorithm. Big Sky, MT: 2010 IEEE Aerospace Conference.

STELLA, Josh. 2015. *O'Riley.com*. [online]. [Accessed 22 May 2017]. Available from World Wide Web: <https://www.oreilly.com/ideas/an-introduction-to-immutable-infrastructure>

STÜBER, G. L. 1996. *Principles of Mobile Communications*. New York, USA: Springer-Science & Business Media.

SUYAMA, S., T. OKUYAMA, Y. INOUE, and Y. KISHIYAMA. 2016. 5G Multi-antenna Technology. *NTT DOCOMO Technical Journal*. **17**(4), pp.29-39.

SWISSCOM. 2018. *Swisscom Innovation Lab*. [online]. [Accessed April 2018]. Available from World Wide Web: <https://www.swisscom.ch/en/business/enterprise/themen/digital-business/smart-enterprise/innovation.html>

TAHA, A-E. M., N. A. ALI, and H. S. HASSANEIN. 2012. *LTE, LTE-Advanced and WiMAX*. Chichester: John Wiley & Sons, Ltd.

TAK, B., C. ISCI, S. DURI et al. 2017. Understanding Security Implicaitons of Using Containers in the Cloud. Santa Clara, CA, USA: USENIX Annual Technical COnference (USENIX ATC '17).

TATARINSKIY, S. N., M. V. KAVUN, and D. N. TREMBACH. 2006. Diversity Reception System. Sevastopol, Crimea: 16th International Crimean Microwave and Telecommunication Technology, pp.1014-1014.

TAWBEH, Ali, Haidar SAFA, and Ahmad, R. AND DHAINI. 2017. A Hybrid SDN/NFV Architecture for Future LTE Networks. Paris: IEEE International Conference on Communications (ICC).

TELECOMPAPER. 2013. *Telecompaper: Samsung develops core 5G technology*. [online]. [Accessed March 2018]. Available from World Wide Web: <https://www.telecompaper.com/news/samsung-develops-core-5g-technology--942840>

TIGERA INC. 2017. *Calico*. [online]. [Accessed October 2017]. Available from World Wide Web: <https://projectcalico.org/>

TOKGOZ, K. K., S. MAKI, J. PANG et al. 2018. A 120Gb/s 16QAM CMOS millimeter-wave wireless transceiver. *In*: *Solid - State Circuits Conference (ISSCC), 2018 IEEE International*. San Francisco, USA: IEEE, pp.168-170.

UDDENFELDT, Jan. 2017. *Keynote: Five Trends Enabled by 5G That Will Change Networking Forever*. [online]. [Accessed 22 May 2017]. Available from World Wide Web: <https://youtu.be/ljQgxQzZPZ0>

VAEZI, Mojtaba and Ying ZHANG. 2017. Cloud Mobile Networks. *In*: *From RAN to EPC*, Cham, Switzerland: Springer International Publishing AG, pp.11-31.

VELDE, Himke van der. 2011. Control Plane Protocols. *In*: S. SESIA, I. TOUFIK, and M. BAKER, (eds). *LTE - The UMTS Long-Term Evolution: From theory to practice*, Chichester, UK: John Wiley & Sons Ltd., pp.57-86.

WEI, H., C-K. C. T. TZUANG, and Y. FAN. 2014. New antenna technology for 5G wireless communications. *IEEE: China Communications*. **11**(11).

WILYSIS. 2018. *Network Cell Info Lite*. [online]. [Accessed April 2018]. Available from World Wide Web: <http://wilysis.com/networkcellinfo>

YANG, Y. Z. a. X. 2016. A Novel Adaptive Beamforming Algorithm for Smart Antenna System. Wuxi: 12th International Conference on Computational Intelligence and Security (CIS), pp.522-525.

YANG, S. L. S., K. M. LUK, H. W. LAI et al. 2008. A dual-polarized antenna with pattern diversity. *IEEE Antennas and Propagation Magazine*. **50**(6), pp.71-79.

YEO, Y., H. LEE, and Y. YOU. 2018. *Distributed Antenna System*. US9859982B2.

YONG-JIANG, S., Q. DONG-DONG, R. JIA-REN et al. 2012. Research on adaptive beamforming algorithm. Hangzhou: International Conference on Image Analysis and Signal Processing, pp.1-3.

APPENDIX

Appendix A. YAML configuration file for deploying a Kubernetes pod

```
apiVersion: v1
kind: Pod
metadata:
  name: db
  labels:
    name: sql
    app: hsscluster

spec:
      containers:
      - image: [the desired container]
        name: database
     replicas: 3
        ports:
        - name: sql
          containerPort: 27017

        volumeMounts:
          - name: sql-storage
            mountPath: /data/db

      volumes:
          - name: sql-storage
            hostPath:
              path: /data/db
```

Appendix B. Router Cisco 2800 configuration

```
brunos-gw>en
Password:
brunos-gw#sh run
Building configuration...

Current configuration : 3257 bytes
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
```

```
!
hostname brunos-gw
!
boot-start-marker
boot-end-marker
!
enable secret 5 $1$4iRB$vYhCHqJ555OHSP3lDvQWG/
!
aaa new-model
!
!
!
aaa session-id common
!
resource policy
!
ip cef
!
!
no ip dhcp use vrf connected
!
ip dhcp pool bruno
   import all
   network 192.168.10.0 255.255.255.0
   default-router 192.168.10.1
   dns-server 158.36.161.20 158.36.161.21
!
ip dhcp pool sidd
   import all
   network 192.168.20.0 255.255.255.0
   default-router 192.168.20.1
   dns-server 158.36.161.20 158.36.161.21
!
!
ip domain name hioa.no
ip name-server 158.36.161.20
ip name-server 158.36.161.21
ip ssh version 2
!
!
!
voice-card 0
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
username bruno privilege 15 secret 5 $1$q29P$9u4CcYzM5yutCq55bpWZN1
!
!
!
```

```
!
!
!
!
interface FastEthernet0/0
 ip address 158.36.118.16 255.255.254.0
 ip nat outside
 no ip virtual-reassembly
 duplex auto
 speed auto
 ipv6 address 2001:700:700:6::16/23
 ipv6 enable
 ipv6 nd ra suppress
!
interface FastEthernet0/1
 ip address 192.168.0.1 255.255.255.0
 ip nat inside
 no ip virtual-reassembly
 duplex auto
 speed auto
!
interface FastEthernet0/1.10
 encapsulation dot1Q 10
 ip address 192.168.10.1 255.255.255.0
 ip nat inside
 no ip virtual-reassembly
!
interface FastEthernet0/1.20
 encapsulation dot1Q 20
 ip address 192.168.20.1 255.255.255.0
 ip nat inside
 no ip virtual-reassembly
!
interface Serial0/2/0
 no ip address
 shutdown
 no fair-queue
 clock rate 125000
!
interface Serial0/2/1
 no ip address
 shutdown
 clock rate 125000
!
router ospf 1
 log-adjacency-changes
 redistribute connected subnets
 redistribute static subnets
 network 158.36.0.0 0.0.255.255 area 0
 network 192.168.0.0 0.0.0.255 area 0
 network 192.168.10.0 0.0.0.255 area 0
 network 192.168.20.0 0.0.0.255 area 0
 default-information originate
!
ip route 0.0.0.0 0.0.0.0 FastEthernet0/0
!
!
ip http server
no ip http secure-server
ip nat inside source list 101 interface FastEthernet0/0 overload
ip nat inside source static tcp 192.168.10.4 2020 interface FastEthernet0/0 2020
ip nat inside source static tcp 192.168.10.4 80 interface FastEthernet0/0 80
ip nat inside source static tcp 192.168.10.3 2021 interface FastEthernet0/0 2021
ip nat inside source static tcp 192.168.10.3 15900 interface FastEthernet0/0 15900
ip nat inside source static udp 192.168.10.3 15900 interface FastEthernet0/0 15900
ip nat inside source static tcp 192.168.10.4 25900 interface FastEthernet0/0 25900
ip nat inside source static tcp 192.168.20.4 2222 interface FastEthernet0/0 2222
ip nat inside source static udp 192.169.10.4 25900 interface FastEthernet0/0 25900
ip nat inside source static tcp 192.168.10.5 2022 interface FastEthernet0/0 2022
ip nat inside source static tcp 192.168.10.5 35900 interface FastEthernet0/0 35900
ip nat inside source static udp 192.168.10.5 35900 interface FastEthernet0/0 35900
!
access-list 101 permit ip 192.168.0.0 0.0.0.255 any
access-list 101 permit ip 192.168.10.0 0.0.0.255 any
access-list 101 permit ip 192.168.20.0 0.0.0.255 any
!
!
!
!
!
!
control-plane
!
!
!
!
!
!
!
!
!
!
line con 0
line aux 0
line vty 0 4
 transport input ssh
!
scheduler allocate 20000 1000
end
```

Appendix C. Cisco Switch 2960 configuration

```
Switch#
Switch#
Switch#sh run
Building configuration...

Current configuration : 2712 bytes
!
version 12.2
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Switch
!
enable password dukenukem89
!
no aaa new-model
system mtu routing 1500
ip subnet-zero
```

```
!
!
!
!
no file verify auto
spanning-tree mode pvst
spanning-tree extend system-id
!
vlan internal allocation policy ascending
!
interface FastEthernet0/1
 switchport access vlan 10
 switchport trunk allowed vlan 1-99
 switchport mode trunk
!
interface FastEthernet0/2
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/3
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/4
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/5
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/6
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/7
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/8
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/9
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/10
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/11
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/12
 switchport access vlan 10
 switchport mode access
!
interface FastEthernet0/13
 switchport access vlan 20
 switchport mode access
!
```

```
interface FastEthernet0/14
 switchport access vlan 20
 switchport mode access
!
interface FastEthernet0/15
 switchport access vlan 20
 switchport mode access
!
interface FastEthernet0/16
 switchport access vlan 20
 switchport mode access
!
interface FastEthernet0/17
 switchport access vlan 20
 switchport mode access
!
interface FastEthernet0/18
 switchport access vlan 20
 switchport mode access
!
interface FastEthernet0/19
 switchport access vlan 20
 switchport mode access
!
interface FastEthernet0/20
 switchport access vlan 20
 switchport mode access
!
interface FastEthernet0/21
 switchport access vlan 20
 switchport mode access
!
interface FastEthernet0/22
 switchport access vlan 20
 switchport mode access
!
interface FastEthernet0/23
 switchport access vlan 20
 switchport mode access
!
interface FastEthernet0/24
 switchport access vlan 20
 switchport mode access
!
interface GigabitEthernet0/1
 switchport access vlan 10
 switchport mode access
!
interface GigabitEthernet0/2
 switchport access vlan 20
 switchport mode access
!
interface Vlan1
 ip address 192.168.0.2 255.255.255.0
 no ip route-cache
!
ip default-gateway 192.168.0.1
ip http server
!
control-plane
!
!
line con 0
line vty 0 4
 password !Zzgn5J8rtyiG*J$VHh@5@#
 login
line vty 5 15
 password !Zzgn5J8rtyiG*J$VHh@5@#
 login
!
end
```

Appendix D. Calico configuration for database access policy

```
cat << EOF | calicoctl apply -f -
- apiVersion: v1
  kind: profile
  metadata:
    name: database
    labels:
      role: database
  spec:
    ingress:
    - action: allow
      protocol: tcp
      source:
        selector: role == 'frontend'
      destination:
        ports:
        -  3306
    - action: allow
      source:
        selector: role == 'database'
    egress:
    - action: allow
      destination:
        selector: role == 'database'
- apiVersion: v1
  kind: profile
  metadata:
    name: frontend
    labels:
      role: frontend
  spec:
    egress:
    - action: allow
      protocol: tcp
      destination:
        selector: role == 'database'
        ports:
        -  3306
EOF
```

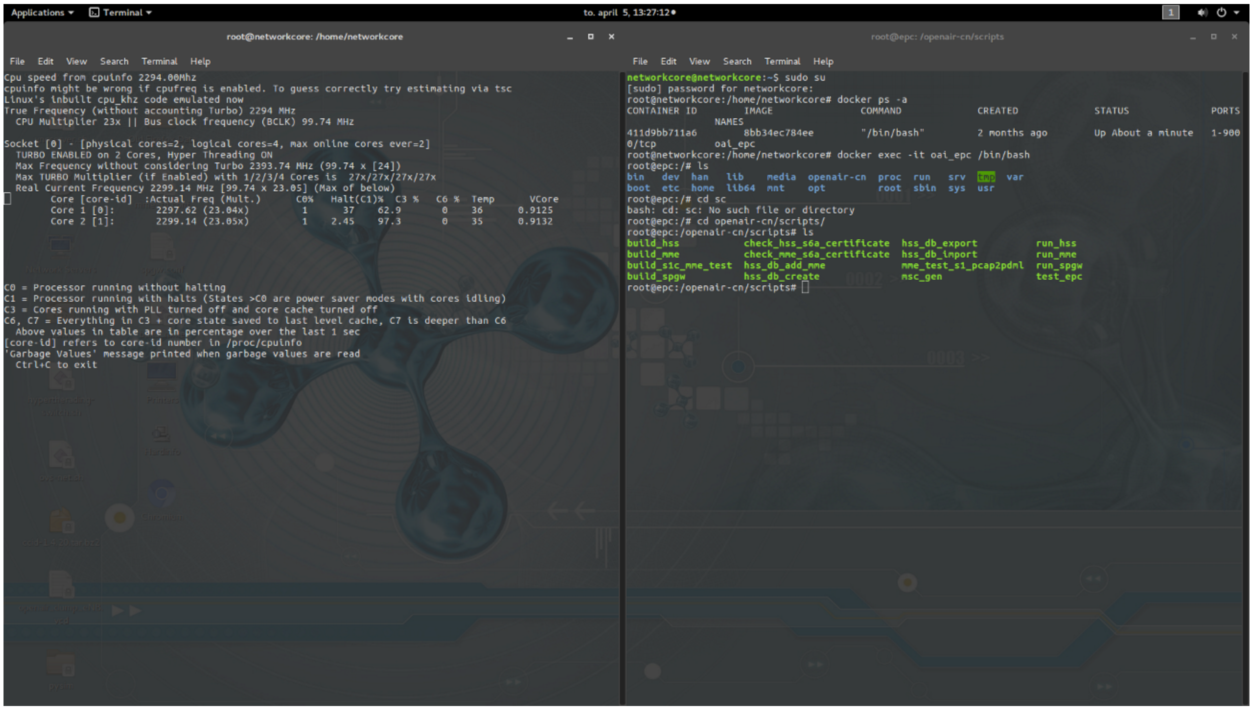
Appendix E. Calico global policy through label selection

```
cat << EOF | calicoctl apply -f -
- apiVersion: v1
  kind: profile
  metadata:
    name: database
    labels:
      role: database
- apiVersion: v1
  kind: profile
  metadata:
    name: frontend
    labels:
      role: frontend
EOF
```

Appendix F. Calico global policy network isolation with ingress and egress rules

```
cat << EOF | calicoctl create -f -
- apiVersion: v1
  kind: policy
  metadata:
    name: database
  spec:
    order: 0
    selector: role == 'database'
    ingress:
    - action: allow
      protocol: tcp
      source:
        selector: role == 'frontend'
      destination:
        ports:
        -  3306
    - action: allow
      source:
        selector: role == 'database'
    egress:
    - action: allow
      destination:
        selector: role == 'database'
- apiVersion: v1
  kind: policy
  metadata:
    name: frontend
  spec:
    order: 0
    selector: role == 'frontend'
    egress:
    - action: allow
      protocol: tcp
      destination:
        selector: role == 'database'
        ports:
        -  3306
EOF
```

Appendix G. i7z CPU state capture


Applications ▾
Terminal ▾
to. april 5, 13:27:12 ●
root@networkcore: /home/networkcore
File Edit View Search Terminal Help
root@epc: /openair-cn/scripts
File Edit View Search Terminal Help

```
Cpu speed from cpuinfo 2294.00Mhz
cpuinfo might be wrong if cpufreq is enabled. To guess correctly try estimating via tsc
Linux's inbuilt cpu_khz code emulated now
True Frequency (without accounting Turbo) 2294 MHz
  CPU Multiplier 23x || Bus clock frequency (BCLK) 99.74 MHz

Socket [0] - [physical cores=2, logical cores=4, max online cores ever=2]
  TURBO ENABLED on 2 Cores, Hyper Threading ON
  Max Frequency without considering Turbo 2393.74 MHz (99.74 x [24])
  Max TURBO Multiplier (if Enabled) with 1/2/3/4 Cores is  27x/27x/27x/27x
  Real Current Frequency 2299.14 MHz [99.74 x 23.05] (Max of below)
        Core [core-id]  :Actual Freq (Mult.)      C0%   Halt(C1)%  C3 %   C6 %  Temp      VCore
        Core 1 [0]:       2297.62 (23.04x)         1      37    62.9       0    36      0.9125
        Core 2 [1]:       2299.14 (23.05x)         1     2.45   97.3       0    35      0.9132



C0 = Processor running without halting
C1 = Processor running with halts (States >C0 are power saver modes with cores idling)
C3 = Cores running with PLL turned off and core cache turned off
C6, C7 = Everything in C3 + core state saved to last level cache, C7 is deeper than C6
  Above values in table are in percentage over the last 1 sec
[core-id] refers to core-id number in /proc/cpuinfo
'Garbage Values' message printed when garbage values are read
  Ctrl+C to exit
```

```
networkcore@networkcore:~$ sudo su
[sudo] password for networkcore:
root@networkcore:/home/networkcore# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
                NAMES
411d9bb711a6        8bb34ec784ee        "/bin/bash"         2 months ago        Up About a minute   1-900
0/tcp           oai_epc
root@networkcore:/home/networkcore# docker exec -it oai_epc /bin/bash
root@epc:/# ls
bin   dev  han   lib    media  openair-cn  proc  run   srv  tmp  var
boot  etc  home  lib64  mnt    opt         root  sbin  sys  usr
root@epc:/# cd sc
bash: cd: sc: No such file or directory
root@epc:/# cd openair-cn/scripts/
root@epc:/openair-cn/scripts# ls
build_hss          check_hss_s6a_certificate  hss_db_export          run_hss
build_mme          check_mme_s6a_certificate  hss_db_import          run_mme
build_s1c_mme_test  hss_db_add_mme            mme_test_s1_pcap2pdml  run_spgw
build_spgw         hss_db_create              msc_gen                test_epc
root@epc:/openair-cn/scripts# 
```

Appendix H. Script for building OvS virtual bridges, Docker container network and auto-configuration of interfaces

```
#!/bin/bash
#
# This script will install OVS, purge existing configuration (if any), install all
# necessary applications, set up bridges, tunnels and link the docker bridge that's
created
# automatically to the OVS bridge that is set up afterwards. This will enable
communication
# between containers on different hosts using OVS overlay.
#
# First, set up the remote IP of the host you want to create a tunnel to:

REMOTE_IP=192.168.10.5

# Install bridgeutils and OVS:
  echo "Installing necessary applications and OpenVSwitch..."
apt-get update && apt-get install bridge-utils libvirt-bin openvswitch-switch -y
  sleep 3
  echo "Clearing existing configuration..."

# First, clear all possible existing configuration (virtual interfaces, bridges etc.)
ovs-vsctl del-br br-int
ip link delete ifconfig |grep veth-* |cut -d " " -f1
ip link delete veth0
ip link delete veth1
ip link delete virbr0

# Create OpenVSwitch tunnel between two hosts, for containers to communicate over
# different physical subnets. The following variable is the docker bridge, which can
be
```

```
# also another bridge you create manually. Therefore, it should be changed adequately
to
# correspond to the bridge you like OVS to create a tunnel for.
  echo "Creating Docker network "oainet" on the subnet 172.19.0.0/24..."
  sleep 4
docker network create -d bridge --attachable --subnet 172.19.0.0/24 --gateway
172.19.0.1 oainet
  sleep 3
DOCKER="$(ifconfig |grep br-* |cut -d " " -f1)"
  echo "Initializing new bridge 'br-int'..."
  sleep 3
ovs-vsctl add-br br-int
ip link add veth0 type veth peer name veth1
ovs-vsctl add-port br-int veth1
  echo "Linking the port of the new Docker bridge with the OVS bridge..."
  sleep 4
brctl addif ${DOCKER} veth0
  echo "Setting up the virtual ethernet interfaces..."
ip link set veth1 up
ip link set veth0 up
  sleep 2
# Then add the virtual tunnel bridge:
  echo "Adding virtual tunnel bridge gre0 and connecting to remote gre on IP
$REMOTE_IP"
ovs-vsctl add-port br-int gre0 -- \
  set interface gre0 type=gre options:remote_ip=$REMOTE_IP
  sleep 3
  echo "Setting virtual ethernet interfaces MTU 1648..."
   ip link set veth0 mtu 1648 up
   ip link set veth1 mtu 1648 up
   ip link set ${DOCKER} mtu 1648 up
  sleep 2
  echo "Done!"
  echo "Try to ping other containers on the other host through the same 172.19.0.0/24
subnet."

# The 192.168.10.5 is the remote IP of the host where the other OVS
# virtual network is located. For example, at the 192.168.10.5 host,
# you add the same commands, with the difference of the remote ip, that
# will be the address of this current host.
```

Appendix I. HSS configuration file

```
################################################################################
# Licensed to the OpenAirInterface (OAI) Software Alliance under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The OpenAirInterface Software Alliance licenses this file to You under
# the Apache License, Version 2.0  (the "License"); you may not use this file
# except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------------------------
# For more information about the OpenAirInterface (OAI) Software Alliance:
```

```
#      contact@openairinterface.org
################################################################################
HSS :
{
## MySQL mandatory options
MYSQL_server = "127.0.0.1";
MYSQL_user   = "root";
MYSQL_pass   = "linux";
MYSQL_db     = "oai_db";

## HSS options
# OPERATOR_key = "1006020f0a478bf6b699f15c062e42b3"; # OP key for oai_db.sql
OPERATOR_key = "11111111111111111111111111111111";

RANDOM = "true";

## Freediameter options
FD_conf = "/usr/local/etc/oai/freeDiameter/hss_fd.conf";
};
```

Appendix J. MME configuration file

```
################################################################################
# Licensed to the OpenAirInterface (OAI) Software Alliance under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The OpenAirInterface Software Alliance licenses this file to You under
# the Apache License, Version 2.0  (the "License"); you may not use this file
# except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------------------------
# For more information about the OpenAirInterface (OAI) Software Alliance:
#      contact@openairinterface.org
################################################################################

MME :
{
    RUN_MODE                                      = "TEST";                        #
ALLOWED VALUES: "TEST", "OTHER"
    REALM                                         = "openair4G.eur";               #  YOUR
REALM HERE
    # Define the limits of the system in terms of served eNB and served UE.
    # When the limits will be reached, overload procedure will take place.
    MAXENB                                        = 2;
    MAXUE                                         = 16;
    RELATIVE_CAPACITY                             = 10;

    EMERGENCY_ATTACH_SUPPORTED                         = "no";
    UNAUTHENTICATED_IMSI_SUPPORTED                     = "no";

    # EPS network feature support
    EPS_NETWORK_FEATURE_SUPPORT_IMS_VOICE_OVER_PS_SESSION_IN_S1      = "no";     # DO
NOT CHANGE
```

```
    EPS_NETWORK_FEATURE_SUPPORT_EMERGENCY_BEARER_SERVICES_IN_S1_MODE = "no";    # DO
NOT CHANGE
    EPS_NETWORK_FEATURE_SUPPORT_LOCATION_SERVICES_VIA_EPC             = "no";    # DO
NOT CHANGE
    EPS_NETWORK_FEATURE_SUPPORT_EXTENDED_SERVICE_REQUEST              = "no";    # DO
NOT CHANGE

    # Display statistics about whole system (expressed in seconds)
    MME_STATISTIC_TIMER                    = 10;

    IP_CAPABILITY = "IPV4V6";                                                   #
UNUSED, TODO


    INTERTASK_INTERFACE :
    {
        # max queue size per task
        ITTI_QUEUE_SIZE            = 2000000;
    };

    S6A :
    {
        S6A_CONF                   = "/usr/local/etc/oai/freeDiameter/mme_fd.conf"; #
YOUR MME freeDiameter config file path
        HSS_HOSTNAME               = "hss";                                     # THE
HSS HOSTNAME
    };

    # ------- SCTP definitions
    SCTP :
    {
        # Number of streams to use in input/output
        SCTP_INSTREAMS  = 8;
        SCTP_OUTSTREAMS = 8;
    };

    # ------- S1AP definitions
    S1AP :
    {
        # outcome drop timer value (seconds)
        S1AP_OUTCOME_TIMER = 10;
    };

    # ------- MME served GUMMEIs
    # MME code DEFAULT  size = 8 bits
    # MME GROUP ID size = 16 bits
    GUMMEI_LIST = (
         {MCC="208" ; MNC="93"; MME_GID="4" ; MME_CODE="1"; }                   # YOUR
GUMMEI CONFIG HERE
    );

    # ------- MME served TAIs
    # TA (mcc.mnc:tracking area code) DEFAULT = 208.34:1
    # max values = 999.999:65535
    # maximum of 16 TAIs, comma separated
# !!! Actually use only one PLMN
    TAI_LIST = (
         {MCC="208" ; MNC="93";  TAC = "15"; },                                 # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "14"; },                                 # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "13"; },                                 # YOUR
TAI CONFIG HERE
```

```
         {MCC="208" ; MNC="93";  TAC = "12"; },                                # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "11"; },                                # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "10"; },                                # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "9"; },                                 # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "8"; },                                 # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "7"; },                                 # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "6"; },                                 # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "5"; },                                 # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "4"; },                                 # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "3"; },                                 # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "2"; },                                 # YOUR
TAI CONFIG HERE
         {MCC="208" ; MNC="93";  TAC = "1"; }                                  # YOUR
TAI CONFIG HERE
    );


    NAS :
    {
        # 3GPP TS 33.401 section 7.2.4.3 Procedures for NAS algorithm selection
        # decreasing preference goes from left to right
        ORDERED_SUPPORTED_INTEGRITY_ALGORITHM_LIST = [ "EIA2" , "EIA1" , "EIA0" ];
        ORDERED_SUPPORTED_CIPHERING_ALGORITHM_LIST = [ "EEA0" , "EEA1" , "EEA2" ];

        # EMM TIMERS
        # T3402 start:
        # At attach failure and the attempt counter is equal to 5.
        # At tracking area updating failure and the attempt counter is equal to 5.
        # T3402 stop:
        # ATTACH REQUEST sent, TRACKING AREA REQUEST sent.
        # On expiry:
        # Initiation of the attach procedure, if still required or TAU procedure
        # attached for emergency bearer services.
        T3402                                  =  1                            # in
minutes (default is 12 minutes)

        # T3412 start:
        # In EMM-REGISTERED, when EMM-CONNECTED mode is left.
        # T3412 stop:
        # When entering state EMM-DEREGISTERED or when entering EMM-CONNECTED mode.
        # On expiry:
        # Initiation of the periodic TAU procedure if the UE is not attached for
        # emergency bearer services. Implicit detach from network if the UE is
        # attached for emergency bearer services.
        T3412                                  =  54                           # in
minutes (default is 54 minutes, network dependent)

        # ESM TIMERS
        T3485                                  =  8                            #
UNUSED in seconds (default is 8s)
        T3486                                  =  8                            #
UNUSED in seconds (default is 8s)
```

```
        T3489                                    =  4                              #
UNUSED in seconds (default is 4s)
        T3495                                    =  8                              #
UNUSED in seconds (default is 8s)
    };

    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         = "eth0";                           # YOUR
NETWORK CONFIG HERE
        MME_IPV4_ADDRESS_FOR_S1_MME           = "172.19.0.2/24";               # YOUR
NETWORK CONFIG HERE

        # MME binded interface for S11 communication (GTPV2-C)
        MME_INTERFACE_NAME_FOR_S11_MME        = "eth0:11";                         # YOUR
NETWORK CONFIG HERE
        MME_IPV4_ADDRESS_FOR_S11_MME          = "192.171.11.1/24";                 # YOUR
NETWORK CONFIG HERE
        MME_PORT_FOR_S11_MME                  = 2123;                              # YOUR
NETWORK CONFIG HERE
    };

    LOGGING :
    {
        # OUTPUT choice in { "CONSOLE", "`path to file`", "`IPv4@`:`TCP port num`"}
        # `path to file` must start with '.' or '/'
        # if TCP stream choice, then you can easily dump the traffic on the remote or
local host: nc -l `TCP port num` > received.txt
        OUTPUT            = "CONSOLE";

        # THREAD_SAFE choice in { "yes", "no" } means use of thread safe intermediate
buffer then a single thread pick each message log one
        # by one to flush it to the chosen output
        THREAD_SAFE       = "no";

        # COLOR choice in { "yes", "no" } means use of ANSI styling codes or no
        COLOR             = "yes";                                                 # TODO

        # Log level choice in { "EMERGENCY", "ALERT", "CRITICAL", "ERROR", "WARNING",
"NOTICE", "INFO", "DEBUG", "TRACE"}
SCTP_LOG_LEVEL    = "TRACE";
        S11_LOG_LEVEL     = "TRACE";
        GTPV2C_LOG_LEVEL  = "TRACE";
        UDP_LOG_LEVEL     = "TRACE";
        S1AP_LOG_LEVEL    = "TRACE";
        NAS_LOG_LEVEL     = "TRACE";
        MME_APP_LOG_LEVEL = "TRACE";
        S6A_LOG_LEVEL     = "TRACE";
        UTIL_LOG_LEVEL    = "TRACE";
        MSC_LOG_LEVEL     = "ERROR";
        ITTI_LOG_LEVEL    = "ERROR";

        # ASN1 VERBOSITY: none, info, annoying
        # for S1AP protocol
        ASN1_VERBOSITY    = "none";
    };
};

S-GW :
{
    # S-GW binded interface for S11 communication (GTPV2-C), if none selected the ITTI
message interface is used
    SGW_IPV4_ADDRESS_FOR_S11                = "192.171.11.2/24";            # YOUR
NETWORK CONFIG HERE

};
```

Appendix K. S/PGW configuration file

```
################################################################################
# Licensed to the OpenAirInterface (OAI) Software Alliance under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The OpenAirInterface Software Alliance licenses this file to You under
# the Apache License, Version 2.0  (the "License"); you may not use this file
# except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------------------------
# For more information about the OpenAirInterface (OAI) Software Alliance:
#      contact@openairinterface.org
################################################################################
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              = "eth0:21";                # YOUR
NETWORK CONFIG HERE
        SGW_IPV4_ADDRESS_FOR_S11                = "192.171.11.2/24";        # 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    = "eth0";                   # YOUR
NETWORK CONFIG HERE, USE "lo" if S-GW run on eNB host
        SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP      = "172.19.0.2/24";       # 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 YET)
        SGW_IPV4_ADDRESS_FOR_S5_S8_UP           = "0.0.0.0/24";             # DO
NOT CHANGE (NOT IMPLEMENTED YET)
    };

    INTERTASK_INTERFACE :
    {
        # max queue size per task
        ITTI_QUEUE_SIZE            = 2000000;
    };

    LOGGING :
    {
        # OUTPUT choice in { "CONSOLE", "`path to file`", "`IPv4@`:`TCP port num`"}
        # `path to file` must start with '.' or '/'
        # if TCP stream choice, then you can easily dump the traffic on the remote or
local host: nc -l `TCP port num` > received.txt
        OUTPUT             = "CONSOLE";

        # THREAD_SAFE choice in { "yes", "no" } means use of thread safe intermediate
buffer then a single thread pick each message log one
        # by one to flush it to the chosen output
        THREAD_SAFE        = "no";

        # COLOR choice in { "yes", "no" } means use of ANSI styling codes or no
        COLOR              = "yes";                                             # TODO

        # Log level choice in { "EMERGENCY", "ALERT", "CRITICAL", "ERROR", "WARNING",
"NOTICE", "INFO", "DEBUG", "TRACE"}
        UDP_LOG_LEVEL       = "TRACE";
        GTPV1U_LOG_LEVEL    = "TRACE";
        GTPV2C_LOG_LEVEL    = "TRACE";
        SPGW_APP_LOG_LEVEL = "TRACE";
        S11_LOG_LEVEL       = "TRACE";
    };
};

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)
        PGW_IPV4_ADDRESS_FOR_S5_S8            = "0.0.0.0/24";                   # DO
NOT CHANGE (NOT IMPLEMENTED YET)

        # P-GW binded interface for SGI (egress/ingress internet traffic)
        PGW_INTERFACE_NAME_FOR_SGI            = "eth0";                         # YOUR
NETWORK CONFIG HERE
        PGW_IPV4_ADDRESS_FOR_SGI              = "172.19.0.2/24";             # YOUR
NETWORK CONFIG HERE
        PGW_MASQUERADE_SGI                    = "yes";                          # YOUR
NETWORK CONFIG HERE
        UE_TCP_MSS_CLAMPING                   = "no"
    };

    # Pool of UE assigned IP addresses
    IP_ADDRESS_POOL :
{
        IPV4_LIST = (
                      "192.188.0.0/24"                                         # YOUR
NETWORK CONFIG HERE
                    );
    };

    # DNS address communicated to UEs
    DEFAULT_DNS_IPV4_ADDRESS     = "192.168.12.100";                            # YOUR
NETWORK CONFIG HERE
```

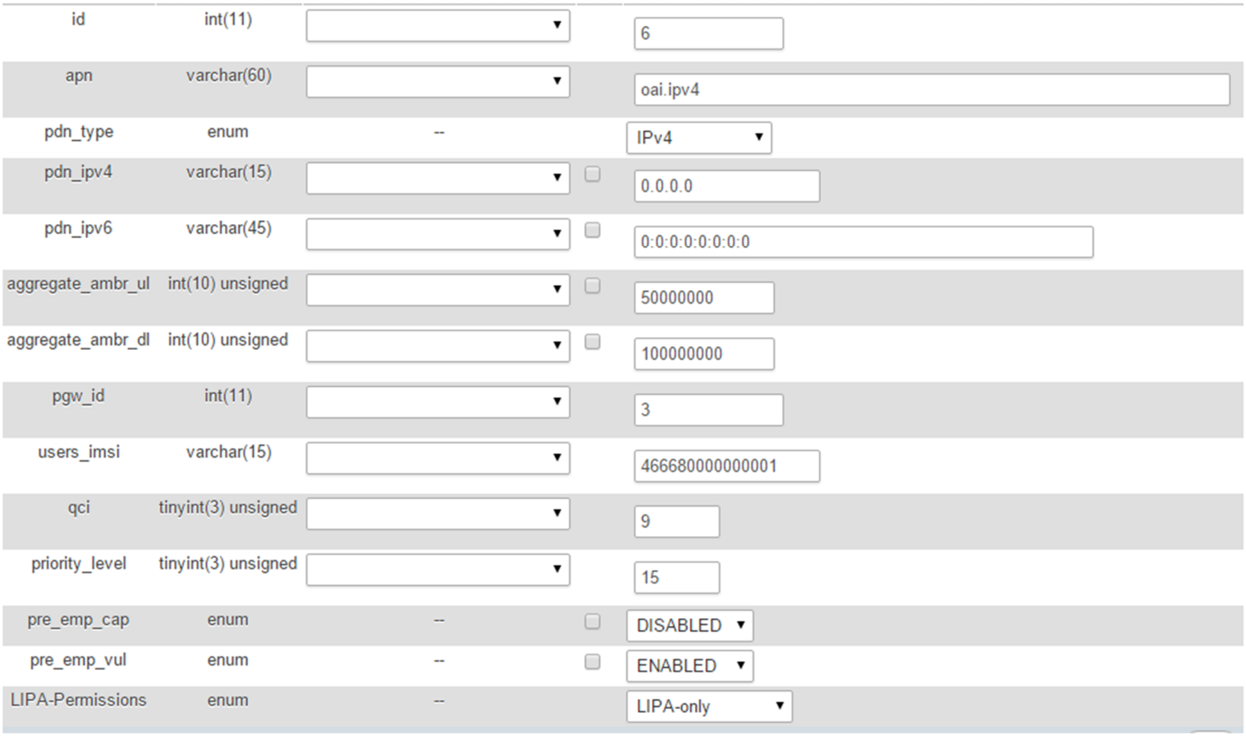
```
    DEFAULT_DNS_SEC_IPV4_ADDRESS = "8.8.8.8";                            # YOUR
NETWORK CONFIG HERE

    # Non standard feature, normally should be set to "no", but you may need to set to
yes for UE that do not explicitly request a PDN address through NAS signalling
    FORCE_PUSH_PROTOCOL_CONFIGURATION_OPTIONS = "yes";
    UE_MTU = 1500;
};
```

Appendix L. eNB configuration file

```
Active_eNBs = ( "eNB_Eurecom_LTEBox");
# Asn1_verbosity, choice in: none, info, annoying
Asn1_verbosity = "none";

eNBs =
(
 {
    ////////// Identification parameters:
    eNB_ID    =  0xe00;

    cell_type =  "CELL_MACRO_ENB";

    eNB_name  =  "eNB_Eurecom_LTEBox";

    // Tracking area code, 0x0000 and 0xfffe are reserved values
    tracking_area_code  =  "1";

    mobile_country_code =  "208";

    mobile_network_code =  "93";

       ////////// Physical parameters:

    component_carriers = (
      {
        node_function                                 = "eNodeB_3GPP";
        node_timing                                   = "synch_to_ext_device"
        node_synch_ref                                = 0;
        frame_type                                    = "FDD";
        tdd_config                                    = 3;
        tdd_config_s                                  = 0;
        prefix_type                                   = "NORMAL";
        eutra_band                                    = 3;
        downlink_frequency                            = 1865000000L;
        uplink_frequency_offset                       = -95000000;
        Nid_cell                                      = 0;
        N_RB_DL                                       = 25;
        Nid_cell_mbsfn                                = 0;
        nb_antennas_tx                                = 2;
        nb_antennas_rx                                = 2;
        nb_antenna_ports                             = 2;
        tx_gain                                       = 90;
        rx_gain                                       = 105;
        prach_root                                    = 0;
        prach_config_index                            = 0;
        prach_high_speed                              = "DISABLE";
        prach_zero_correlation                        = 1;
        prach_freq_offset                             = 2;
        pucch_delta_shift                                      = 1;
        pucch_nRB_CQI                                          = 1;
        pucch_nCS_AN                                           = 0;
        pucch_n1_AN                                            = 32;
        pdsch_referenceSignalPower                             = -24;
        pdsch_p_b                                              = 0;
        pusch_n_SB                                             = 1;
        pusch_enable64QAM                                      = "DISABLE";
        pusch_hoppingMode                                      = "interSubFrame";
        pusch_hoppingOffset                                    = 0;
        pusch_groupHoppingEnabled                              = "ENABLE";
        pusch_groupAssignment                                  = 0;
        pusch_sequenceHoppingEnabled                           = "DISABLE";
        pusch_nDMRS1                                           = 1;
        phich_duration                                         = "NORMAL";
        phich_resource                                         = "ONESIXTH";
        srs_enable                                             = "DISABLE";
        srs_BandwidthConfig                                 = 2;
        srs_SubframeConfig                                  = 0;
        srs_ackNackST                                       = "DISABLE";
        srs_MaxUpPts                                        = "DISABLE";

        pusch_p0_Nominal                                    = -90;
        pusch_alpha                                         = "AL1";
        pucch_p0_Nominal                                    = -96;
        msg3_delta_Preamble                                 = 6;
        pucch_deltaF_Format1                                = "deltaF2";
        pucch_deltaF_Format1b                               = "deltaF3";
        pucch_deltaF_Format2                                = "deltaF0";
        pucch_deltaF_Format2a                               = "deltaF0";
        pucch_deltaF_Format2b                          = "deltaF0";

        rach_numberOfRA_Preambles                           = 64;
        rach_preamblesGroupAConfig                          = "DISABLE";
        /*
        rach_sizeOfRA_PreamblesGroupA                       = ;
        rach_messageSizeGroupA                              = ;
        rach_messagePowerOffsetGroupB                       = ;
        */
        rach_powerRampingStep                               = 4;
        rach_preambleInitialReceivedTargetPower             = -104;
        rach_preambleTransMax                               = 10;
        rach_raResponseWindowSize                           = 10;
        rach_macContentionResolutionTimer                   = 48;
        rach_maxHARQ_Msg3Tx                                 = 4;

        pcch_default_PagingCycle                            = 128;
        pcch_nB                                             = "oneT";
        bcch_modificationPeriodCoeff                           = 2;
        ue_TimersAndConstants_t300                             = 1000;
        ue_TimersAndConstants_t301                             = 1000;
        ue_TimersAndConstants_t310                             = 1000;
        ue_TimersAndConstants_t311                             = 10000;
        ue_TimersAndConstants_n310                             = 20;
        ue_TimersAndConstants_n311                             = 1;
        ue_TransmissionMode                                    = 1;
        }
    );


    srb1_parameters :
    {
        # timer_poll_retransmit = (ms) [5, 10, 15, 20,... 250, 300, 350, ... 500]
        timer_poll_retransmit    = 80;

        # timer_reordering = (ms) [0,5, ... 100, 110, 120, ... ,200]
        timer_reordering         = 35;

        # timer_reordering = (ms) [0,5, ... 250, 300, 350, ... ,500]
        timer_status_prohibit    = 0;

        # poll_pdu = [4, 8, 16, 32 , 64, 128, 256, infinity(>10000)]
        poll_pdu                 =  4;

        # poll_byte = (kB)
[25,50,75,100,125,250,375,500,750,1000,1250,1500,2000,3000,infinity(>10000)]
        poll_byte                =  99999;

        # max_retx_threshold = [1, 2, 3, 4 , 6, 8, 16, 32]
        max_retx_threshold       =  4;
    }

    # ------- SCTP definitions
    SCTP :
    {
        # Number of streams to use in input/output
        SCTP_INSTREAMS  = 2;
        SCTP_OUTSTREAMS = 2;
    };

    ////////// MME parameters:
    mme_ip_address      = ( { ipv4       = "172.19.0.2";
                              ipv6       = "192:168:30::17";
                              active     = "yes";
                              preference = "ipv4";
                            }
                          );

    NETWORK_INTERFACES :
    {
        ENB_INTERFACE_NAME_FOR_S1_MME            = "eth1";
        ENB_IPV4_ADDRESS_FOR_S1_MME              = "172.19.0.3/24";

        ENB_INTERFACE_NAME_FOR_S1U               = "eth1";
        ENB_IPV4_ADDRESS_FOR_S1U                 = "172.19.0.3/24";
        ENB_PORT_FOR_S1U                         = 2152; # Spec 2152
    };

    log_config :
    {
        global_log_level                      ="info";
        global_log_verbosity                  ="medium";
        hw_log_level                          ="info";
        hw_log_verbosity                      ="medium";
        phy_log_level                         ="info";
        phy_log_verbosity                     ="medium";
        mac_log_level                         ="info";
        mac_log_verbosity                     ="high";
        rlc_log_level                         ="info";
        rlc_log_verbosity                     ="medium";
        pdcp_log_level                        ="info";
        pdcp_log_verbosity                    ="medium";
        rrc_log_level                         ="info";
        rrc_log_verbosity                     ="medium";
   };
```

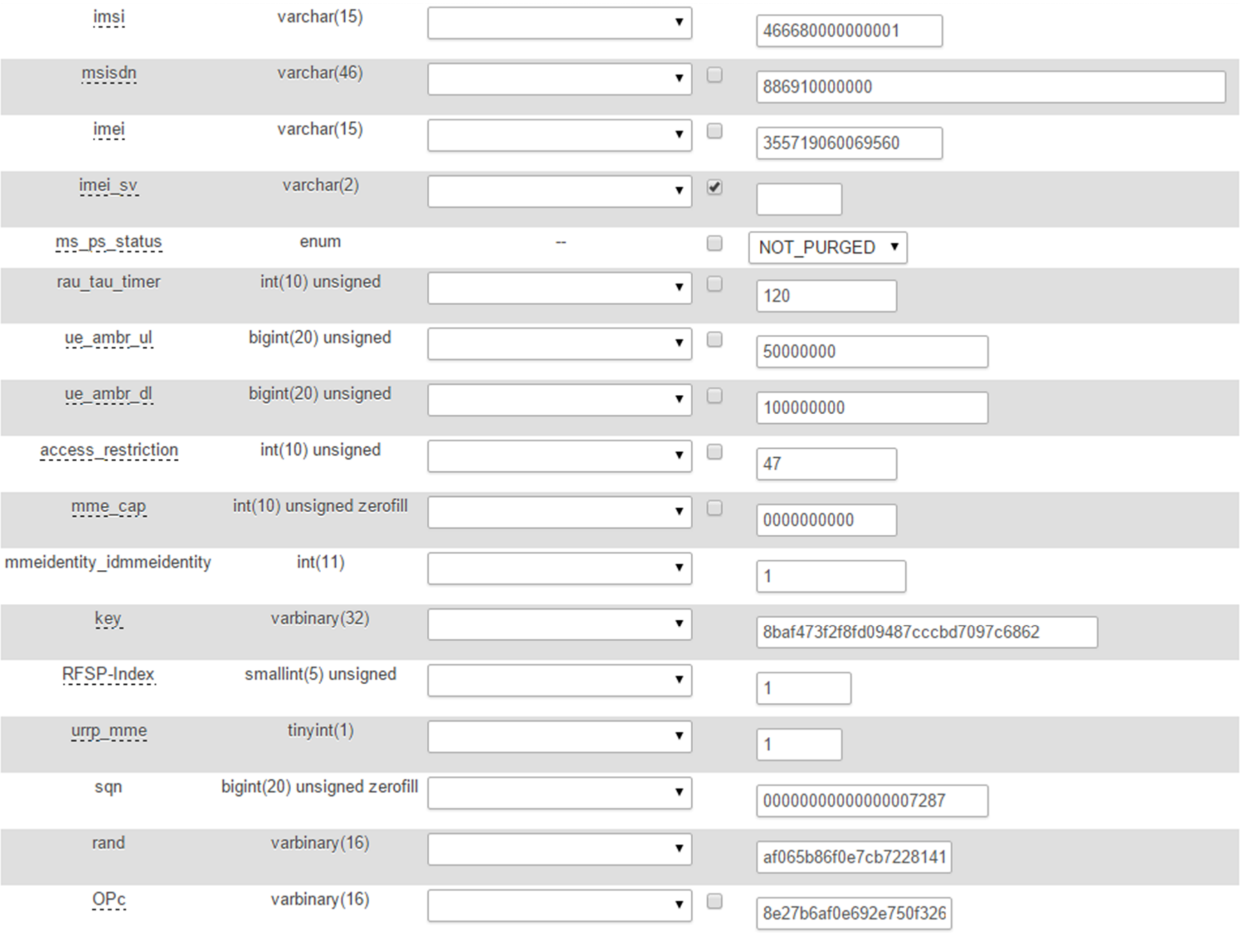
```
}
```

Appendix M. PDN table in the HSS database



id	int(11)	▾		6
apn	varchar(60)	▾		oai.ipv4
pdn_type	enum	--		IPv4 ▾
pdn_ipv4	varchar(15)	▾	☐	0.0.0.0
pdn_ipv6	varchar(45)	▾	☐	0:0:0:0:0:0:0:0
aggregate_ambr_ul	int(10) unsigned	▾	☐	50000000
aggregate_ambr_dl	int(10) unsigned	▾	☐	100000000
pgw_id	int(11)	▾		3
users_imsi	varchar(15)	▾		466680000000001
qci	tinyint(3) unsigned	▾		9
priority_level	tinyint(3) unsigned	▾		15
pre_emp_cap	enum	--	☐	DISABLED ▾
pre_emp_vul	enum	--	☐	ENABLED ▾
LIPA-Permissions	enum	--		LIPA-only ▾

Appendix N. Users table in the HSS database



imsi	varchar(15)	▾		466680000000001
msisdn	varchar(46)	▾	□	886910000000
imei	varchar(15)	▾	□	355719060069560
imei_sv	varchar(2)	▾	☑	
ms_ps_status	enum	--	□	NOT_PURGED ▾
rau_tau_timer	int(10) unsigned	▾	□	120
ue_ambr_ul	bigint(20) unsigned	▾	□	50000000
ue_ambr_dl	bigint(20) unsigned	▾	□	100000000
access_restriction	int(10) unsigned	▾	□	47
mme_cap	int(10) unsigned zerofill	▾	□	0000000000
mmeidentity_idmmeidentity	int(11)	▾		1
key	varbinary(32)	▾		8baf473f2f8fd09487cccbd7097c6862
RFSP-Index	smallint(5) unsigned	▾		1
urrp_mme	tinyint(1)	▾		1
sqn	bigint(20) unsigned zerofill	▾		00000000000000007287
rand	varbinary(16)	▾		af065b86f0e7cb7228141
OPc	varbinary(16)	▾	□	8e27b6af0e692e750f326

Appendix O. eNB configuration file

```
Active_eNBs = ( "eNB_Eurecom_LTEBox");
# Asn1_verbosity, choice in: none, info, annoying
Asn1_verbosity = "none";

eNBs =
(
 {
    ////////// Identification parameters:
    eNB_ID    =  0xe00;

    cell_type =  "CELL_MACRO_ENB";

    eNB_name  =  "eNB_Eurecom_LTEBox";

    // Tracking area code, 0x0000 and 0xfffe are reserved values
    tracking_area_code  =  "1";

    mobile_country_code =  "208";

    mobile_network_code =  "93";

       ////////// Physical parameters:
```

```
    component_carriers = (
      {
        node_function                                          = "eNodeB_3GPP";
        node_timing                                            = "synch_to_ext_device"
        node_synch_ref                                         = 0;
        frame_type                                             = "FDD";
        tdd_config                                             = 3;
        tdd_config_s                                           = 0;
        prefix_type                                            = "NORMAL";
        eutra_band                                             = 3;
        downlink_frequency                                     = 1865000000L;
        uplink_frequency_offset                                = -95000000;
        Nid_cell                                               = 0;
        N_RB_DL                                                = 25;
        Nid_cell_mbsfn                                         = 0;
        nb_antennas_tx                                         = 2;
        nb_antennas_rx                                         = 2;
        nb_antenna_ports                                      = 2;
        tx_gain                                                = 90;
        rx_gain                                                = 105;
        prach_root                                             = 0;
        prach_config_index                                     = 0;
        prach_high_speed                                       = "DISABLE";
        prach_zero_correlation                                 = 1;
        prach_freq_offset                                      = 2;
        pucch_delta_shift                                      = 1;
        pucch_nRB_CQI                                          = 1;
        pucch_nCS_AN                                           = 0;
        pucch_n1_AN                                            = 32;
        pdsch_referenceSignalPower                             = -24;
        pdsch_p_b                                              = 0;
        pusch_n_SB                                             = 1;
        pusch_enable64QAM                                      = "DISABLE";
        pusch_hoppingMode                                      = "interSubFrame";
        pusch_hoppingOffset                                    = 0;
        pusch_groupHoppingEnabled                              = "ENABLE";
        pusch_groupAssignment                                  = 0;
        pusch_sequenceHoppingEnabled                           = "DISABLE";
        pusch_nDMRS1                                           = 1;
        phich_duration                                         = "NORMAL";
        phich_resource                                         = "ONESIXTH";
        srs_enable                                             = "DISABLE";
        srs_BandwidthConfig                                 = 2;
        srs_SubframeConfig                                  = 0;
        srs_ackNackST                                       = "DISABLE";
        srs_MaxUpPts                                        = "DISABLE";

        pusch_p0_Nominal                                    = -90;
        pusch_alpha                                         = "AL1";
        pucch_p0_Nominal                                    = -96;
        msg3_delta_Preamble                                 = 6;
        pucch_deltaF_Format1                                = "deltaF2";
        pucch_deltaF_Format1b                               = "deltaF3";
        pucch_deltaF_Format2                                = "deltaF0";
        pucch_deltaF_Format2a                               = "deltaF0";
        pucch_deltaF_Format2b                          = "deltaF0";

        rach_numberOfRA_Preambles                           = 64;
        rach_preamblesGroupAConfig                          = "DISABLE";
        /*
        rach_sizeOfRA_PreamblesGroupA                       = ;
        rach_messageSizeGroupA                              = ;
        rach_messagePowerOffsetGroupB                      = ;
        */
        rach_powerRampingStep                              = 4;
        rach_preambleInitialReceivedTargetPower            = -104;
        rach_preambleTransMax                              = 10;
        rach_raResponseWindowSize                          = 10;
        rach_macContentionResolutionTimer                  = 48;
        rach_maxHARQ_Msg3Tx                                = 4;

        pcch_default_PagingCycle                           = 128;
        pcch_nB                                            = "oneT";
        bcch_modificationPeriodCoeff                          = 2;
        ue_TimersAndConstants_t300                            = 1000;
        ue_TimersAndConstants_t301                            = 1000;
        ue_TimersAndConstants_t310                            = 1000;
        ue_TimersAndConstants_t311                            = 10000;
        ue_TimersAndConstants_n310                            = 20;
        ue_TimersAndConstants_n311                            = 1;
        ue_TransmissionMode                                   = 1;
        }
    );


    srb1_parameters :
    {
        # timer_poll_retransmit = (ms) [5, 10, 15, 20,... 250, 300, 350, ... 500]
        timer_poll_retransmit    = 80;

        # timer_reordering = (ms) [0,5, ... 100, 110, 120, ... ,200]
        timer_reordering         = 35;

        # timer_reordering = (ms) [0,5, ... 250, 300, 350, ... ,500]
        timer_status_prohibit    = 0;

        # poll_pdu = [4, 8, 16, 32 , 64, 128, 256, infinity(>10000)]
        poll_pdu                 =  4;

        # poll_byte = (kB)
[25,50,75,100,125,250,375,500,750,1000,1250,1500,2000,3000,infinity(>10000)]
        poll_byte                =  99999;

        # max_retx_threshold = [1, 2, 3, 4 , 6, 8, 16, 32]
        max_retx_threshold       =  4;
    }

    # ------- SCTP definitions
    SCTP :
    {
        # Number of streams to use in input/output
        SCTP_INSTREAMS  = 2;
        SCTP_OUTSTREAMS = 2;
    };

    ////////// MME parameters:
    mme_ip_address      = ( { ipv4       = "172.19.0.2";
                              ipv6       = "192:168:30::17";
                              active     = "yes";
                              preference = "ipv4";
                            }
                          );

    NETWORK_INTERFACES :
    {
        ENB_INTERFACE_NAME_FOR_S1_MME            = "eth1";
        ENB_IPV4_ADDRESS_FOR_S1_MME              = "172.19.0.3/24";

        ENB_INTERFACE_NAME_FOR_S1U               = "eth1";
        ENB_IPV4_ADDRESS_FOR_S1U                 = "172.19.0.3/24";
        ENB_PORT_FOR_S1U                         = 2152; # Spec 2152
    };

    log_config :
    {
        global_log_level                      ="info";
        global_log_verbosity                  ="medium";
        hw_log_level                          ="info";
        hw_log_verbosity                      ="medium";
        phy_log_level                         ="info";
        phy_log_verbosity                     ="medium";
        mac_log_level                         ="info";
        mac_log_verbosity                     ="high";
        rlc_log_level                         ="info";
        rlc_log_verbosity                     ="medium";
        pdcp_log_level                        ="info";
        pdcp_log_verbosity                    ="medium";
        rrc_log_level                         ="info";
        rrc_log_verbosity                     ="medium";
   };

  }
);
```

Appendix P. USIM cards programming procedure

1. First, some librarires are required for installation:

libusb-dev; libusb++-0.1-4c2, and the *libccid* package is installed: `apt-get install libccid`

2. The *pcsc-lite* package is installed: `apt-get install pcscd`
3. At this point, since the *pcscd* service is running, the *libpcsclite1* is installed: `apt-get install libpcsclite1`
4. *libpcsclite-dev* is installed: `apt-get install libpcsclite-dev`
5. Additional tools installed:
   - *libpcsc-perl*: `apt-get install libpcsc-perl`
   - *pcsctools*: `apt-get install pcsc-tools`. PCSC-tools provide several appealing applications for Blutronics Bludrive II
6. CCID installation:
   - `cd ~/src` and `wget https://alioth.debian.org/frs/download.php/file/4140/ccid-1.4.20.tar.bz2 && tar -xjf ccid-1.4.20.tar.bz2 && cd ccid-1.4.20`
7. Installation configuration: At this point, there are some options that can be issued to customize the installation. An important example is the one which forces the CCID driver to use *udev* events so that *pcscd* will not poll the USB bus every second.

```
./configure

--bindir=

--disable-FEATURE
--dvidir=
--enable-maintainer-mode
```

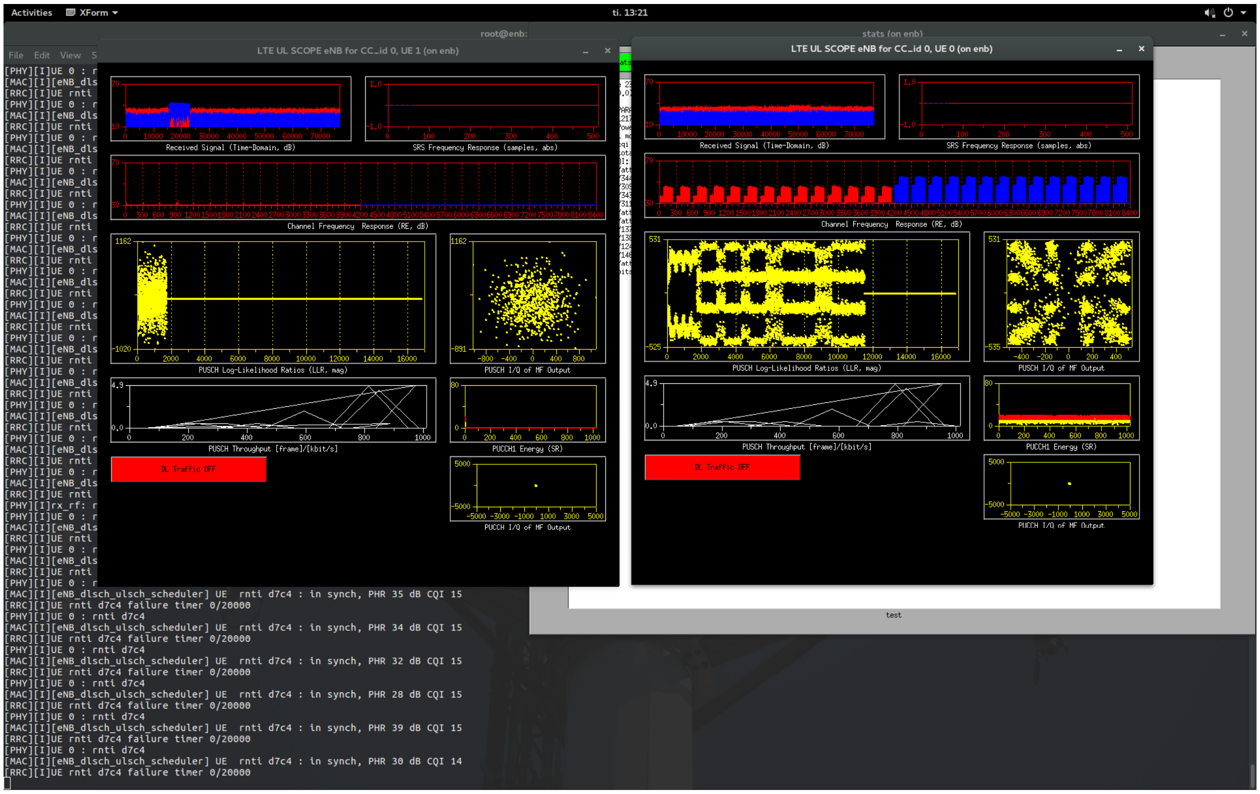
```
--exec-prefix=
--libexecdir=
--program-prefix=
--sysconfdir=
--build=
--disable-libtool-lock
--enable-bundle=
--enable-serialconfdir=
--help
--localedir=
--program-suffix=
--version
--cache-file=
--disable-libusb
--enable-ccidtwindir=
--enable-shared
--help=
--localstatedir=
--program-transform-name=
--with-gnu-ld
--config-cache
--disable-multi-thread
--enable-composite-as-multislot
--enable-silent-rules
--host=
--mandir=
--psdir=
--without-PACKAGE--datadir=
--disable-option-checking
--enable-dependency-tracking
--enable-static
--htmldir=
--no-create
--quiet
--with-PACKAGE--datarootdir=
--disable-pcsclite
--enable-embedded
--enable-syslog
--includedir=
--oldincludedir=
--sbindir=
--with-pic
--disable-class
--disable-silent-rules
--enable-fast-install
--enable-twinserial
--infodir=
--pdfdir=
--sharedstatedir=
--with-sysroot=
--disable-dependency-tracking
--docdir=
--enable-FEATURE
--enable-usbdropdir=
--libdir=
--prefix=
--srcdir=
```

8. Installation of the CCID driver:
   ```
   make –j4
   make install
   ```
9. Udev rules are copied to */etc/udev/rules.d*
10. Since a rule for altering usb auto-suspend kernel configuration exists in the configuration and Pcsc is supposed to use usb auto-suspend for CCID devices only, which is enabled: `sudo cp src/92_pcscd_ccid.rules /etc/udev/rules.d/`
11. Middleware installation: `sudo apt-get install libpcsclite1 pcscd pcsc-tools`

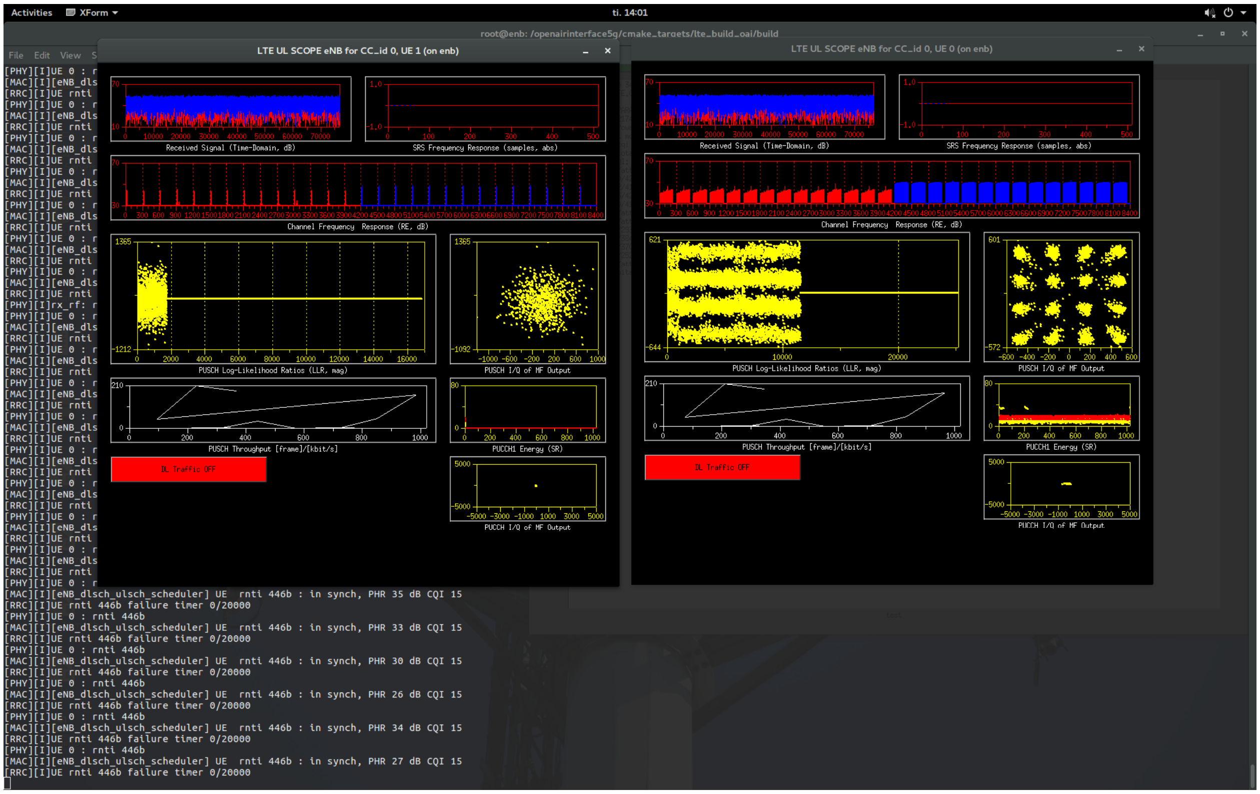
12. Detect the card reader/writer run the pcscd daemon with debug option, in foreground: `sudo pcscd -d -a -f`
13. On another terminal the following command is run: `pcsc_scan`

    That gives the following output: `PC/SC device scannerV 1.4.22 (c) 2001-2011, Ludovic Rousseau <ludovic.rousseau@free.fr>Compiled with PC/SC lite version: 1.8.10 Using reader plug 'n play mechanism. Scanning present readers...0: BLUTRONICS BLUDRIVE II CCID (62657374) 00 00Tue Sep 1 21:27:43 2015Reader 0: BLUTRONICS BLUDRIVE II CCID (62657374) 00 00Card state: Card inserted, ATR: 3B 7D 94 00 00 55 55 53 0A 74 86 93 0B 24 7C 4D 54 68ATR: 3B 7D 94 00 00 55 55 53 0A 74 86 93 0B 24 7C 4D 54 68+ TS = 3B --> Direct Convention+ T0 = 7D, Y(1): 0111, K: 13 (historical bytes)TA(1) = 94 --> Fi=512, Di=8, 64 cycles/ETU62500 bits/s at 4 MHz, fMax for Fi = 5 MHz => 78125 bits/sTB(1) = 00 --> VPP is not electrically connectedTC(1) = 00 --> Extra guard time: 0+ Historical bytes: 55 55 53 0A 74 86 93 0B 24 7C 4D 54 68Category indicator byte: 55 (proprietary format)Possibly identified card (using /usr/share/pcsc/smartcard_list.txt):3B 7D 94 00 00 55 55 53 0A 74 86 93 0B 24 7C 4D 54 68SIM from sysmocom sysmoSIM-GR2`

Appendix Q. Radio measurements of the attached UE to the eNB

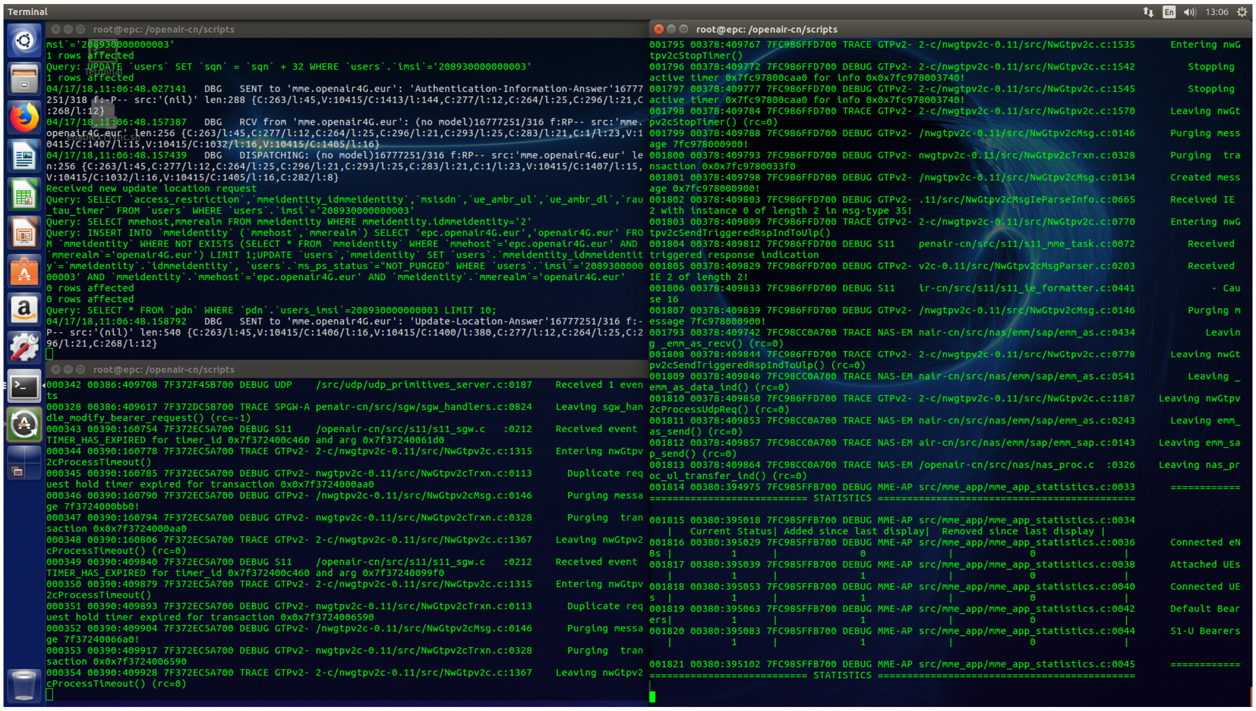

Activities
XForm
ti. 13:21
root@enb:
stats (on enb)
File Edit View S
LTE UL SCOPE eNB for CC_id 0, UE 1 (on enb)
LTE UL SCOPE eNB for CC_id 0, UE 0 (on enb)
Received Signal (Time-Domain, dB)
SRS Frequency Response (samples, abs)
Channel Frequency Response (RE, dB)
PUSCH Log-Likelihood Ratios (LLR, mag)
PUSCH I/Q of MF Output
PUSCH Throughput [frame]/[kbit/s]
PUCCH1 Energy (SR)
PUCCH I/Q of MF Output
DL Traffic OFF
test

```
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]rx_rf: r
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dls
[RRC][I]UE rnti
[PHY][I]UE 0 : r
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti d7c4 : in synch, PHR 35 dB CQI 15
[RRC][I]UE rnti d7c4 failure timer 0/20000
[PHY][I]UE 0 : rnti d7c4
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti d7c4 : in synch, PHR 34 dB CQI 15
[RRC][I]UE rnti d7c4 failure timer 0/20000
[PHY][I]UE 0 : rnti d7c4
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti d7c4 : in synch, PHR 32 dB CQI 15
[RRC][I]UE rnti d7c4 failure timer 0/20000
[PHY][I]UE 0 : rnti d7c4
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti d7c4 : in synch, PHR 28 dB CQI 15
[RRC][I]UE rnti d7c4 failure timer 0/20000
[PHY][I]UE 0 : rnti d7c4
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti d7c4 : in synch, PHR 39 dB CQI 15
[RRC][I]UE rnti d7c4 failure timer 0/20000
[PHY][I]UE 0 : rnti d7c4
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti d7c4 : in synch, PHR 30 dB CQI 14
[RRC][I]UE rnti d7c4 failure timer 0/20000
```


Activities
XForm
ti. 14:01
root@enb: /openairinterface5g/cmake_targets/lte_build_oai/build
File Edit View
LTE UL SCOPE eNB for CC_id 0, UE 1 (on enb)
LTE UL SCOPE eNB for CC_id 0, UE 0 (on enb)
Received Signal (Time-Domain, dB)
SRS Frequency Response (samples, abs)
Channel Frequency Response (RE, dB)
PUSCH Log-Likelihood Ratios (LLR, mag)
PUSCH I/Q of MF Output
PUSCH Throughput [frame]/[kbit/s]
PUCCH1 Energy (SR)
PUCCH I/Q of MF Output
DL Traffic OFF

```
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti 446b : in synch, PHR 35 dB CQI 15
[RRC][I]UE rnti 446b failure timer 0/20000
[PHY][I]UE 0 : rnti 446b
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti 446b : in synch, PHR 33 dB CQI 15
[RRC][I]UE rnti 446b failure timer 0/20000
[PHY][I]UE 0 : rnti 446b
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti 446b : in synch, PHR 30 dB CQI 15
[RRC][I]UE rnti 446b failure timer 0/20000
[PHY][I]UE 0 : rnti 446b
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti 446b : in synch, PHR 26 dB CQI 15
[RRC][I]UE rnti 446b failure timer 0/20000
[PHY][I]UE 0 : rnti 446b
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti 446b : in synch, PHR 34 dB CQI 15
[RRC][I]UE rnti 446b failure timer 0/20000
[PHY][I]UE 0 : rnti 446b
[MAC][I][eNB_dlsch_ulsch_scheduler] UE  rnti 446b : in synch, PHR 27 dB CQI 15
[RRC][I]UE rnti 446b failure timer 0/20000
```

Appendix R. HSS, S/PGW and MME real-time operation insight



root@epc: /openair-cn/scripts

```
msi`='208930000000003'
1 rows affected
Query: UPDATE `users` SET `sqn` = `sqn` + 32 WHERE `users`.`imsi`='208930000000003'
1 rows affected
04/17/18,11:06:48.027141   DBG   SENT to 'mme.openair4G.eur': 'Authentication-Information-Answer'16777
251/318 f:-P-- src:'(nil)' len:288 {C:263/l:45,V:10415/C:1413/l:144,C:277/l:12,C:264/l:25,C:296/l:21,C
:268/l:12}
04/17/18,11:06:48.157387   DBG   RCV from 'mme.openair4G.eur': (no model)16777251/316 f:RP-- src:'mme.
openair4G.eur' len:256 {C:263/l:45,C:277/l:12,C:264/l:25,C:296/l:21,C:293/l:25,C:283/l:21,C:1/l:23,V:1
0415/C:1407/l:15,V:10415/C:1032/l:16,V:10415/C:1405/l:16}
04/17/18,11:06:48.157439   DBG   DISPATCHING: (no model)16777251/316 f:RP-- src:'mme.openair4G.eur' le
n:256 {C:263/l:45,C:277/l:12,C:264/l:25,C:296/l:21,C:293/l:25,C:283/l:21,C:1/l:23,V:10415/C:1407/l:15,
V:10415/C:1032/l:16,V:10415/C:1405/l:16,C:282/l:8}
Received new update location request
Query: SELECT `access_restriction`,`mmeidentity_idmmeidentity`,`msisdn`,`ue_ambr_ul`,`ue_ambr_dl`,`rau
_tau_timer` FROM `users` WHERE `users`.`imsi`='208930000000003'
Query: SELECT mmehost,mmerealm FROM mmeidentity WHERE mmeidentity.idmmeidentity='2'
Query: INSERT INTO `mmeidentity` (`mmehost`,`mmerealm`) SELECT 'epc.openair4G.eur','openair4G.eur' FRO
M `mmeidentity` WHERE NOT EXISTS (SELECT * FROM `mmeidentity` WHERE `mmehost`='epc.openair4G.eur' AND
`mmerealm`='openair4G.eur') LIMIT 1;UPDATE `users`,`mmeidentity` SET `users`.`mmeidentity_idmmeidentit
y`=`mmeidentity`.`idmmeidentity`, `users`.`ms_ps_status`="NOT_PURGED" WHERE `users`.`imsi`='2089300000
00003' AND `mmeidentity`.`mmehost`='epc.openair4G.eur' AND `mmeidentity`.`mmerealm`='openair4G.eur'
0 rows affected
0 rows affected
Query: SELECT * FROM `pdn` WHERE `pdn`.`users_imsi`=208930000000003 LIMIT 10;
04/17/18,11:06:48.158792   DBG   SENT to 'mme.openair4G.eur': 'Update-Location-Answer'16777251/316 f:-
P-- src:'(nil)' len:540 {C:263/l:45,V:10415/C:1406/l:16,V:10415/C:1400/l:380,C:277/l:12,C:264/l:25,C:2
96/l:21,C:268/l:12}
```

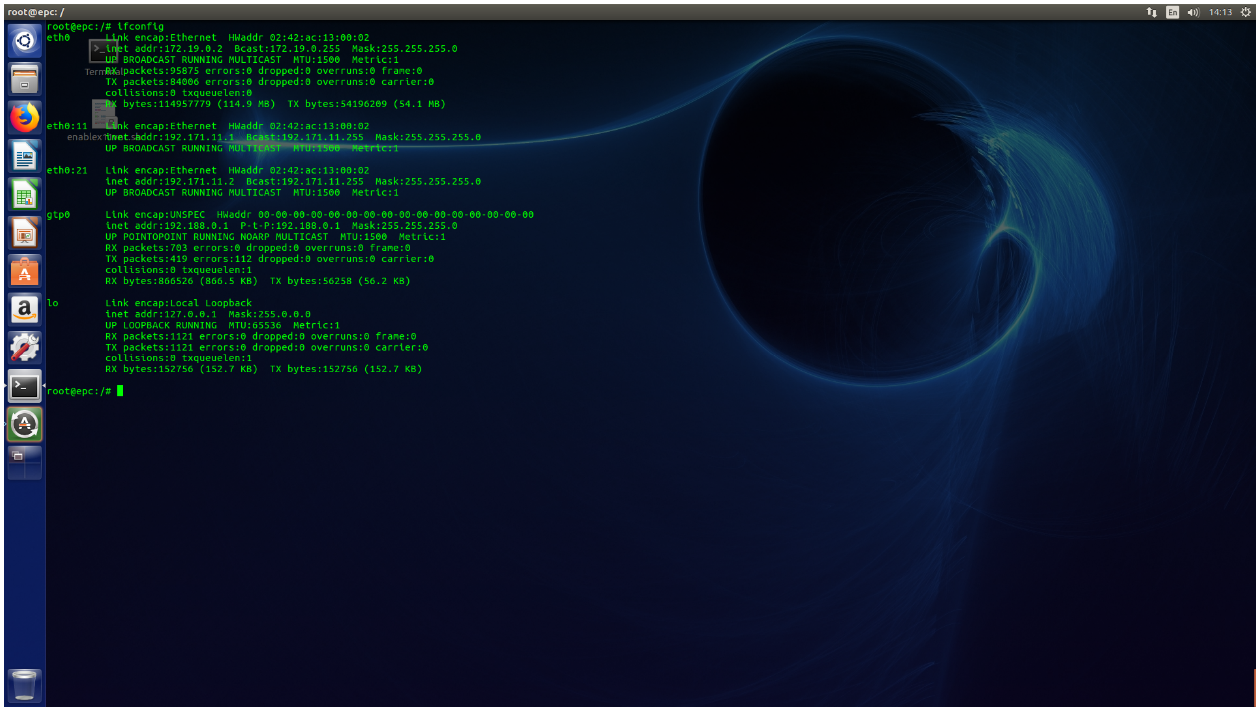
root@epc: /openair-cn/scripts

```
000342 00386:409708 7F372F45B700 DEBUG UDP    /src/udp/udp_primitives_server.c:0187    Received 1 even
ts
000328 00386:409617 7F372DC58700 TRACE SPGW-A penair-cn/src/sgw/sgw_handlers.c:0824    Leaving sgw_han
dle_modify_bearer_request() (rc=-1)
000343 00390:160754 7F372EC5A700 DEBUG S11    /openair-cn/src/s11/s11_sgw.c   :0212    Received event
TIMER_HAS_EXPIRED for timer_id 0x7f372400c460 and arg 0x7f37240061d0
000344 00390:160778 7F372EC5A700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1315    Entering nwGtpv
2cProcessTimeout()
000345 00390:160785 7F372EC5A700 DEBUG GTPv2- nwgtpv2c-0.11/src/NwGtpv2cTrxn.c:0113     Duplicate req
uest hold timer expired for transaction 0x0x7f3724000aa0
000346 00390:160790 7F372EC5A700 DEBUG GTPv2- /nwgtpv2c-0.11/src/NwGtpv2cMsg.c:0146     Purging messa
ge 7f3724000bb0!
000347 00390:160794 7F372EC5A700 DEBUG GTPv2- nwgtpv2c-0.11/src/NwGtpv2cTrxn.c:0328     Purging  tran
saction 0x0x7f3724000aa0
000348 00390:160806 7F372EC5A700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1367    Leaving nwGtpv2
cProcessTimeout() (rc=0)
000349 00390:409840 7F372EC5A700 DEBUG S11    /openair-cn/src/s11/s11_sgw.c   :0212    Received event
TIMER_HAS_EXPIRED for timer_id 0x7f372400c460 and arg 0x7f37240099f0
000350 00390:409879 7F372EC5A700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1315    Entering nwGtpv
2cProcessTimeout()
000351 00390:409893 7F372EC5A700 DEBUG GTPv2- nwgtpv2c-0.11/src/NwGtpv2cTrxn.c:0113     Duplicate req
uest hold timer expired for transaction 0x0x7f3724006590
000352 00390:409904 7F372EC5A700 DEBUG GTPv2- /nwgtpv2c-0.11/src/NwGtpv2cMsg.c:0146     Purging messa
ge 7f37240066a0!
000353 00390:409917 7F372EC5A700 DEBUG GTPv2- nwgtpv2c-0.11/src/NwGtpv2cTrxn.c:0328     Purging  tran
saction 0x0x7f3724006590
000354 00390:409928 7F372EC5A700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1367    Leaving nwGtpv2
cProcessTimeout() (rc=0)
```

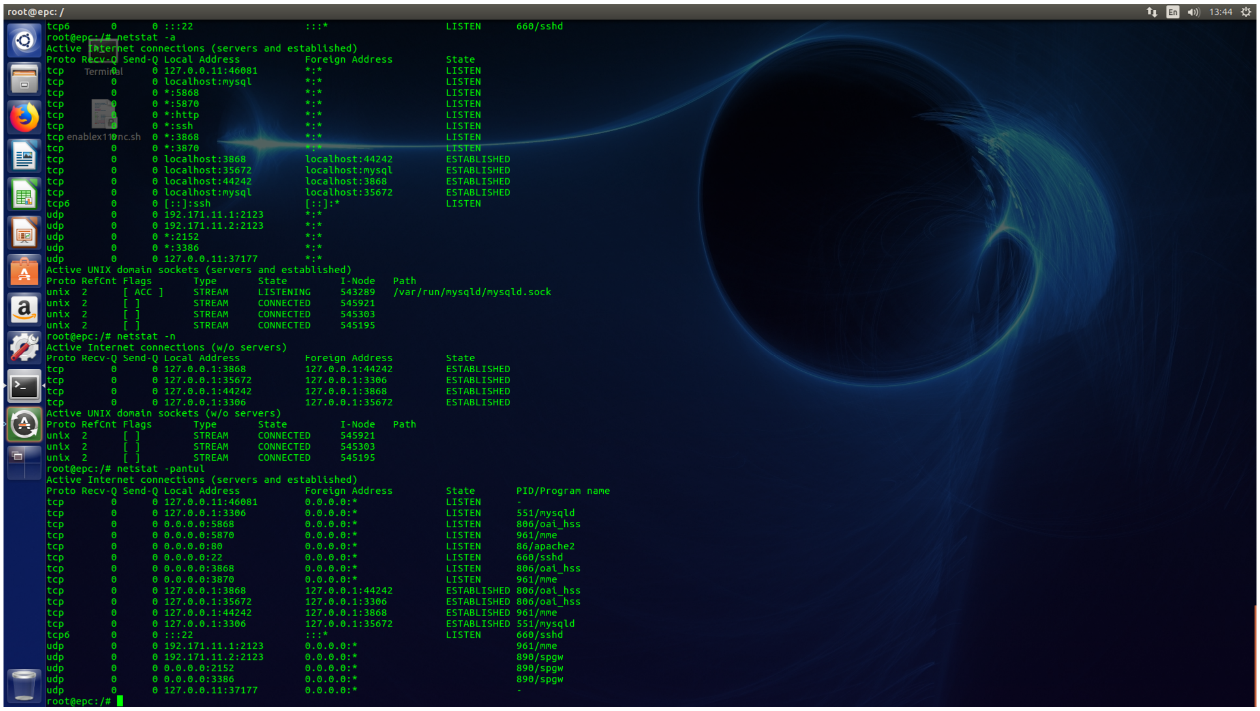
root@epc: /openair-cn/scripts

```
001795 00378:409767 7FC986FFD700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1535     Entering nwG
tpv2cStopTimer()
001796 00378:409772 7FC986FFD700 DEBUG GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1542        Stopping
active timer 0x7fc97800caa0 for info 0x0x7fc978003740!
001797 00378:409777 7FC986FFD700 DEBUG GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1545        Stopping
active timer 0x7fc97800caa0 for info 0x0x7fc978003740!
001798 00378:409784 7FC986FFD700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1570     Leaving nwGt
pv2cStopTimer() (rc=0)
001799 00378:409788 7FC986FFD700 DEBUG GTPv2- /nwgtpv2c-0.11/src/NwGtpv2cMsg.c:0146     Purging mess
age 7fc978000900!
001800 00378:409793 7FC986FFD700 DEBUG GTPv2- nwgtpv2c-0.11/src/NwGtpv2cTrxn.c:0328     Purging  tra
nsaction 0x0x7fc9780033f0
001801 00378:409798 7FC986FFD700 DEBUG GTPv2- /nwgtpv2c-0.11/src/NwGtpv2cMsg.c:0134     Created mess
age 0x7fc978000900!
001802 00378:409803 7FC986FFD700 DEBUG GTPv2- .11/src/NwGtpv2cMsgIeParseInfo.c:0665     Received IE
2 with instance 0 of length 2 in msg-type 35!
001803 00378:409809 7FC986FFD700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:0770     Entering nwG
tpv2cSendTriggeredRspIndToUlp()
001804 00378:409812 7FC986FFD700 DEBUG S11    penair-cn/src/s11/s11_mme_task.c:0072        Received
triggered response indication
001805 00378:409829 7FC986FFD700 DEBUG GTPv2- v2c-0.11/src/NwGtpv2cMsgParser.c:0203        Received
IE 2 of length 2!
001806 00378:409833 7FC986FFD700 DEBUG S11    ir-cn/src/s11/s11_ie_formatter.c:0441           - Cau
se 16
001807 00378:409839 7FC986FFD700 DEBUG GTPv2- /nwgtpv2c-0.11/src/NwGtpv2cMsg.c:0146        Purging m
essage 7fc978000900!
001793 00378:409742 7FC98CC0A700 TRACE NAS-EM nair-cn/src/nas/emm/sap/emm_as.c:0434          Leavin
g _emm_as_recv() (rc=0)
001808 00378:409844 7FC986FFD700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:0778     Leaving nwGt
pv2cSendTriggeredRspIndToUlp() (rc=0)
001809 00378:409846 7FC98CC0A700 TRACE NAS-EM nair-cn/src/nas/emm/sap/emm_as.c:0541        Leaving _
emm_as_data_ind() (rc=0)
001810 00378:409850 7FC986FFD700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1187    Leaving nwGtpv
2cProcessUdpReq() (rc=0)
001811 00378:409853 7FC98CC0A700 TRACE NAS-EM nair-cn/src/nas/emm/sap/emm_as.c:0243     Leaving emm_
as_send() (rc=0)
001812 00378:409857 7FC98CC0A700 TRACE NAS-EM air-cn/src/nas/emm/sap/emm_sap.c:0143    Leaving emm_sa
p_send() (rc=0)
001813 00378:409864 7FC98CC0A700 TRACE NAS-EM /openair-cn/src/nas/nas_proc.c  :0326    Leaving nas_pr
oc_ul_transfer_ind() (rc=0)
001814 00380:394975 7FC985FFB700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0033     ===========
========================== STATISTICS ======================================

001815 00380:395018 7FC985FFB700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0034
   |   Current Status| Added since last display|  Removed since last display |
001816 00380:395029 7FC985FFB700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0036     Connected eN
Bs |         1      |             0             |            0              |
001817 00380:395039 7FC985FFB700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0038     Attached UEs
   |         1      |             1             |            0              |
001818 00380:395053 7FC985FFB700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0040     Connected UE
s  |         1      |             1             |            0              |
001819 00380:395063 7FC985FFB700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0042     Default Bear
ers|         1      |             1             |            0              |
001820 00380:395083 7FC985FFB700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0044     S1-U Bearers
   |         1      |             1             |            0              |

001821 00380:395102 7FC985FFB700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0045     ===========
========================== STATISTICS ======================================
```

Appendix S. Starting *iperf3* server on the *gtp0* interface of the S/PGW

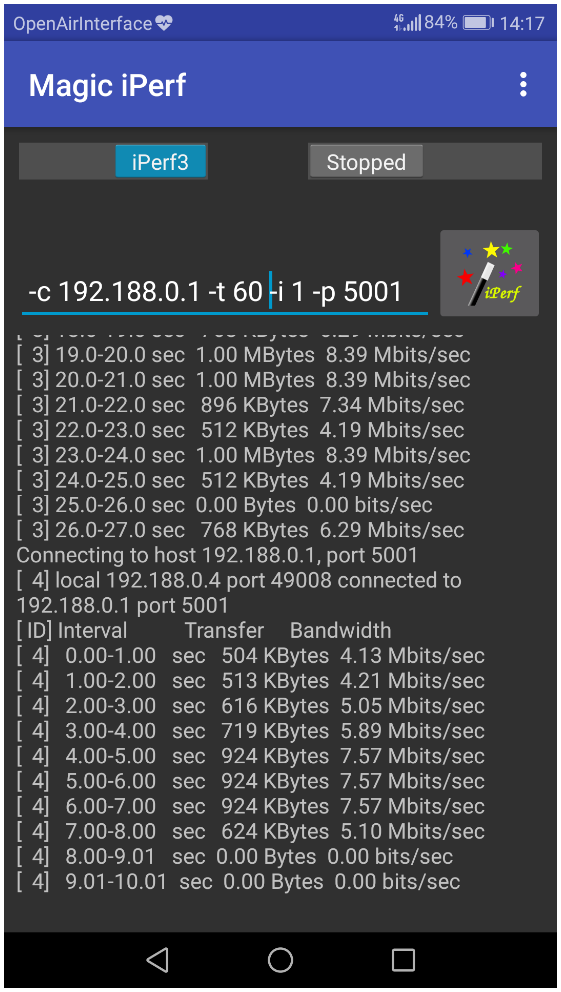


```
root@epc:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:13:00:02
          inet addr:172.19.0.2  Bcast:172.19.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:95875 errors:0 dropped:0 overruns:0 frame:0
          TX packets:84006 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:114957779 (114.9 MB)  TX bytes:54196209 (54.1 MB)

eth0:11   Link encap:Ethernet  HWaddr 02:42:ac:13:00:02
          inet addr:192.171.11.1  Bcast:192.171.11.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:21   Link encap:Ethernet  HWaddr 02:42:ac:13:00:02
          inet addr:192.171.11.2  Bcast:192.171.11.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

gtp0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:192.188.0.1  P-t-P:192.188.0.1  Mask:255.255.255.0
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:703 errors:0 dropped:0 overruns:0 frame:0
          TX packets:419 errors:112 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:866526 (866.5 KB)  TX bytes:56258 (56.2 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1121 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1121 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:152756 (152.7 KB)  TX bytes:152756 (152.7 KB)

root@epc:/# 
```



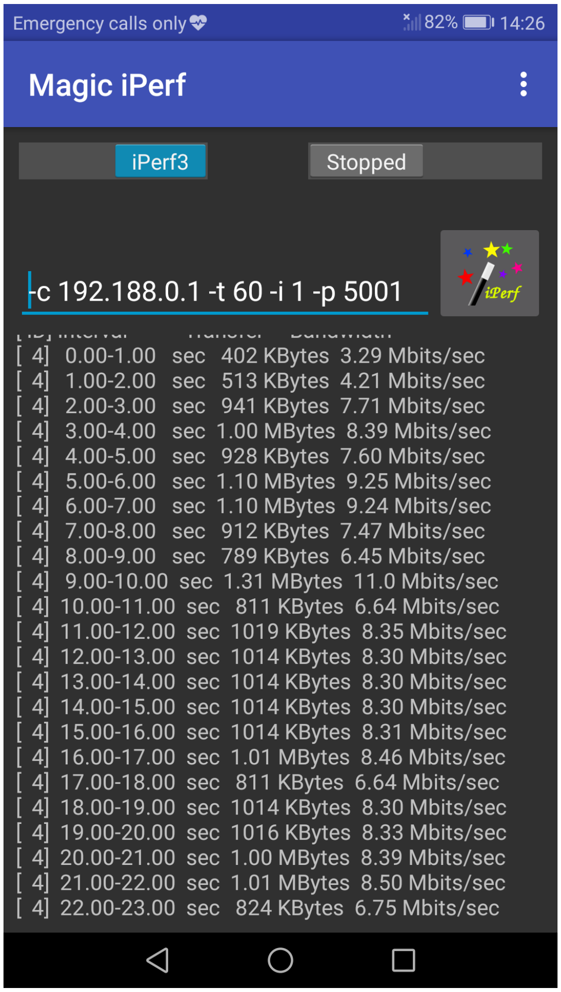
```
tcp6       0      0 :::22                   :::*                    LISTEN      660/sshd
root@epc:/# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:46081        *:*                     LISTEN
tcp        0      0 localhost:mysql         *:*                     LISTEN
tcp        0      0 *:5868                  *:*                     LISTEN
tcp        0      0 *:5870                  *:*                     LISTEN
tcp        0      0 *:http                  *:*                     LISTEN
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 *:3868                  *:*                     LISTEN
tcp        0      0 *:3870                  *:*                     LISTEN
tcp        0      0 localhost:3868          localhost:44242         ESTABLISHED
tcp        0      0 localhost:35672         localhost:mysql         ESTABLISHED
tcp        0      0 localhost:44242         localhost:3868          ESTABLISHED
tcp        0      0 localhost:mysql         localhost:35672         ESTABLISHED
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
udp        0      0 192.171.11.1:2123       *:*
udp        0      0 192.171.11.2:2123       *:*
udp        0      0 *:2152                  *:*
udp        0      0 *:3386                  *:*
udp        0      0 127.0.0.11:37177        *:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ACC ]     STREAM     LISTENING     543289   /var/run/mysqld/mysqld.sock
unix  2      [ ]         STREAM     CONNECTED     545921
unix  2      [ ]         STREAM     CONNECTED     545303
unix  2      [ ]         STREAM     CONNECTED     545195
root@epc:/# netstat -n
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:3868          127.0.0.1:44242         ESTABLISHED
tcp        0      0 127.0.0.1:35672         127.0.0.1:3306          ESTABLISHED
tcp        0      0 127.0.0.1:44242         127.0.0.1:3868          ESTABLISHED
tcp        0      0 127.0.0.1:3306          127.0.0.1:35672         ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ]         STREAM     CONNECTED     545921
unix  2      [ ]         STREAM     CONNECTED     545303
unix  2      [ ]         STREAM     CONNECTED     545195
root@epc:/# netstat -pantul
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.11:46081        0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      551/mysqld
tcp        0      0 0.0.0.0:5868            0.0.0.0:*               LISTEN      806/oai_hss
tcp        0      0 0.0.0.0:5870            0.0.0.0:*               LISTEN      961/mme
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      86/apache2
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      660/sshd
tcp        0      0 0.0.0.0:3868            0.0.0.0:*               LISTEN      806/oai_hss
tcp        0      0 0.0.0.0:3870            0.0.0.0:*               LISTEN      961/mme
tcp        0      0 127.0.0.1:3868          127.0.0.1:44242         ESTABLISHED 806/oai_hss
tcp        0      0 127.0.0.1:35672         127.0.0.1:3306          ESTABLISHED 806/oai_hss
tcp        0      0 127.0.0.1:44242         127.0.0.1:3868          ESTABLISHED 961/mme
tcp        0      0 127.0.0.1:3306          127.0.0.1:35672         ESTABLISHED 551/mysqld
tcp6       0      0 :::22                   :::*                    LISTEN      660/sshd
udp        0      0 192.171.11.1:2123       0.0.0.0:*                           961/mme
udp        0      0 192.171.11.2:2123       0.0.0.0:*                           890/spgw
udp        0      0 0.0.0.0:2152            0.0.0.0:*                           890/spgw
udp        0      0 0.0.0.0:3386            0.0.0.0:*                           890/spgw
udp        0      0 127.0.0.11:37177        0.0.0.0:*                           -
root@epc:/# 
```

Appendix T. iPerf3 UE side results



-c 192.188.0.1 -t 60 -i 1 -p 5001

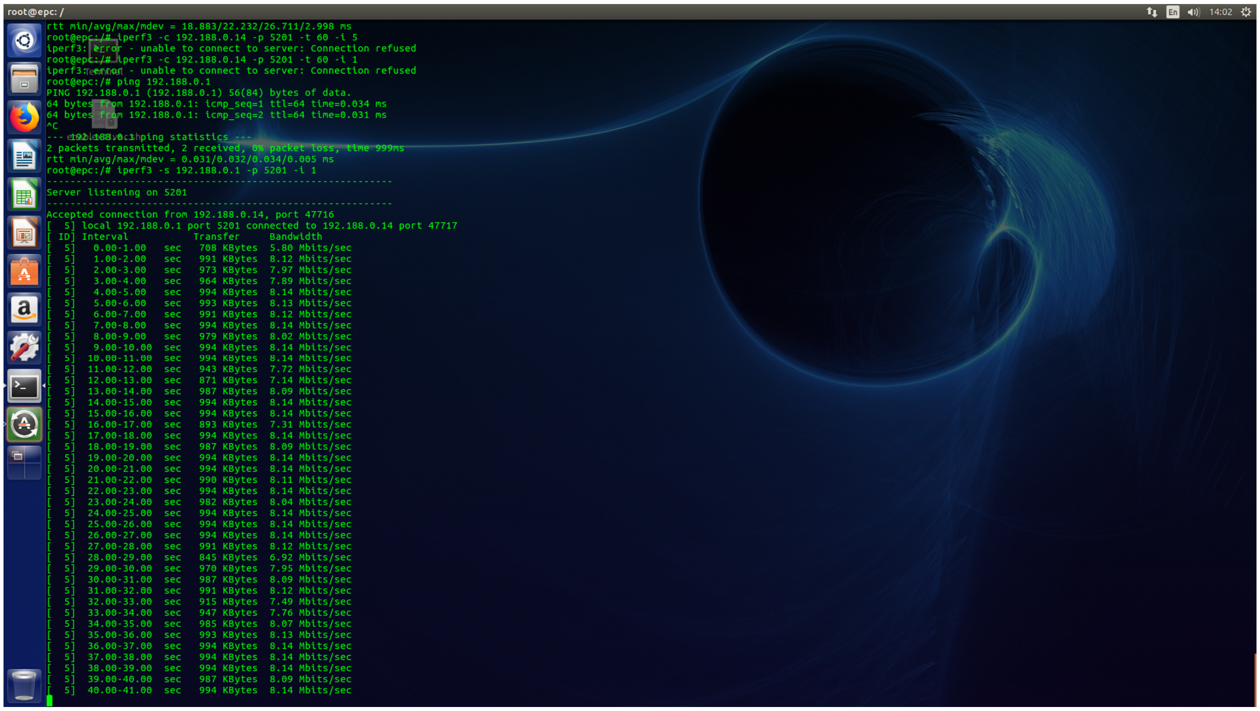
```
[ 3] 19.0-20.0 sec  1.00 MBytes  8.39 Mbits/sec
[ 3] 20.0-21.0 sec  1.00 MBytes  8.39 Mbits/sec
[ 3] 21.0-22.0 sec   896 KBytes  7.34 Mbits/sec
[ 3] 22.0-23.0 sec   512 KBytes  4.19 Mbits/sec
[ 3] 23.0-24.0 sec  1.00 MBytes  8.39 Mbits/sec
[ 3] 24.0-25.0 sec   512 KBytes  4.19 Mbits/sec
[ 3] 25.0-26.0 sec  0.00 Bytes  0.00 bits/sec
[ 3] 26.0-27.0 sec   768 KBytes  6.29 Mbits/sec
Connecting to host 192.188.0.1, port 5001
[  4] local 192.188.0.4 port 49008 connected to
192.188.0.1 port 5001
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-1.00   sec   504 KBytes  4.13 Mbits/sec
[  4]   1.00-2.00   sec   513 KBytes  4.21 Mbits/sec
[  4]   2.00-3.00   sec   616 KBytes  5.05 Mbits/sec
[  4]   3.00-4.00   sec   719 KBytes  5.89 Mbits/sec
[  4]   4.00-5.00   sec   924 KBytes  7.57 Mbits/sec
[  4]   5.00-6.00   sec   924 KBytes  7.57 Mbits/sec
[  4]   6.00-7.00   sec   924 KBytes  7.57 Mbits/sec
[  4]   7.00-8.00   sec   624 KBytes  5.10 Mbits/sec
[  4]   8.00-9.01   sec  0.00 Bytes  0.00 bits/sec
[  4]   9.01-10.01  sec  0.00 Bytes  0.00 bits/sec
```



-c 192.188.0.1 -t 60 -i 1 -p 5001

```
[  4]   0.00-1.00   sec   402 KBytes  3.29 Mbits/sec
[  4]   1.00-2.00   sec   513 KBytes  4.21 Mbits/sec
[  4]   2.00-3.00   sec   941 KBytes  7.71 Mbits/sec
[  4]   3.00-4.00   sec  1.00 MBytes  8.39 Mbits/sec
[  4]   4.00-5.00   sec   928 KBytes  7.60 Mbits/sec
[  4]   5.00-6.00   sec  1.10 MBytes  9.25 Mbits/sec
[  4]   6.00-7.00   sec  1.10 MBytes  9.24 Mbits/sec
[  4]   7.00-8.00   sec   912 KBytes  7.47 Mbits/sec
[  4]   8.00-9.00   sec   789 KBytes  6.45 Mbits/sec
[  4]   9.00-10.00  sec  1.31 MBytes  11.0 Mbits/sec
[  4]  10.00-11.00  sec   811 KBytes  6.64 Mbits/sec
[  4]  11.00-12.00  sec  1019 KBytes  8.35 Mbits/sec
[  4]  12.00-13.00  sec  1014 KBytes  8.30 Mbits/sec
[  4]  13.00-14.00  sec  1014 KBytes  8.30 Mbits/sec
[  4]  14.00-15.00  sec  1014 KBytes  8.30 Mbits/sec
[  4]  15.00-16.00  sec  1014 KBytes  8.31 Mbits/sec
[  4]  16.00-17.00  sec  1.01 MBytes  8.46 Mbits/sec
[  4]  17.00-18.00  sec   811 KBytes  6.64 Mbits/sec
[  4]  18.00-19.00  sec  1014 KBytes  8.30 Mbits/sec
[  4]  19.00-20.00  sec  1016 KBytes  8.33 Mbits/sec
[  4]  20.00-21.00  sec  1.00 MBytes  8.39 Mbits/sec
[  4]  21.00-22.00  sec  1.01 MBytes  8.50 Mbits/sec
[  4]  22.00-23.00  sec   824 KBytes  6.75 Mbits/sec
```

Appendix U. iPerf3 EPC side results



```
rtt min/avg/max/mdev = 18.883/22.232/26.711/2.998 ms
root@epc:/# iperf3 -c 192.188.0.14 -p 5201 -t 60 -i 5
iperf3: error - unable to connect to server: Connection refused
root@epc:/# iperf3 -c 192.188.0.14 -p 5201 -t 60 -i 1
iperf3: error - unable to connect to server: Connection refused
root@epc:/# ping 192.188.0.1
PING 192.188.0.1 (192.188.0.1) 56(84) bytes of data.
64 bytes from 192.188.0.1: icmp_seq=1 ttl=64 time=0.034 ms
64 bytes from 192.188.0.1: icmp_seq=2 ttl=64 time=0.031 ms
^C
--- 192.188.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.031/0.032/0.034/0.005 ms
root@epc:/# iperf3 -s 192.188.0.1 -p 5201 -i 1
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.188.0.14, port 47716
[  5] local 192.188.0.1 port 5201 connected to 192.188.0.14 port 47717
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec   708 KBytes  5.80 Mbits/sec
[  5]   1.00-2.00   sec   991 KBytes  8.12 Mbits/sec
[  5]   2.00-3.00   sec   973 KBytes  7.97 Mbits/sec
[  5]   3.00-4.00   sec   964 KBytes  7.89 Mbits/sec
[  5]   4.00-5.00   sec   994 KBytes  8.14 Mbits/sec
[  5]   5.00-6.00   sec   993 KBytes  8.13 Mbits/sec
[  5]   6.00-7.00   sec   991 KBytes  8.12 Mbits/sec
[  5]   7.00-8.00   sec   994 KBytes  8.14 Mbits/sec
[  5]   8.00-9.00   sec   979 KBytes  8.02 Mbits/sec
[  5]   9.00-10.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  10.00-11.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  11.00-12.00  sec   943 KBytes  7.72 Mbits/sec
[  5]  12.00-13.00  sec   871 KBytes  7.14 Mbits/sec
[  5]  13.00-14.00  sec   987 KBytes  8.09 Mbits/sec
[  5]  14.00-15.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  15.00-16.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  16.00-17.00  sec   893 KBytes  7.31 Mbits/sec
[  5]  17.00-18.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  18.00-19.00  sec   987 KBytes  8.09 Mbits/sec
[  5]  19.00-20.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  20.00-21.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  21.00-22.00  sec   990 KBytes  8.11 Mbits/sec
[  5]  22.00-23.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  23.00-24.00  sec   982 KBytes  8.04 Mbits/sec
[  5]  24.00-25.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  25.00-26.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  26.00-27.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  27.00-28.00  sec   991 KBytes  8.12 Mbits/sec
[  5]  28.00-29.00  sec   845 KBytes  6.92 Mbits/sec
[  5]  29.00-30.00  sec   970 KBytes  7.95 Mbits/sec
[  5]  30.00-31.00  sec   987 KBytes  8.09 Mbits/sec
[  5]  31.00-32.00  sec   991 KBytes  8.12 Mbits/sec
[  5]  32.00-33.00  sec   915 KBytes  7.49 Mbits/sec
[  5]  33.00-34.00  sec   947 KBytes  7.76 Mbits/sec
[  5]  34.00-35.00  sec   985 KBytes  8.07 Mbits/sec
[  5]  35.00-36.00  sec   993 KBytes  8.13 Mbits/sec
[  5]  36.00-37.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  37.00-38.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  38.00-39.00  sec   994 KBytes  8.14 Mbits/sec
[  5]  39.00-40.00  sec   987 KBytes  8.09 Mbits/sec
[  5]  40.00-41.00  sec   994 KBytes  8.14 Mbits/sec
```

Appendix V. OpenStack Heat template for deploying core network EPC

```
heat_template_version: 2015-04-30

description: >
  Heat Orchestration Template (HOT) to deploy OpenAir Core Network vEPC.
  Networks, security groups, volumes, ports and instances get created by
heat,
  latest openair core network code is compiled on the build instance and
  deployed to the SPGW/MME/HSS instances. Once the stack is created, it
  provides the SSH commandline to access the vEPC.

### STACK INPUTS
parameters:
  key:
    type: string
    label: SSH Keypair
    description: Name of the SSH keypair for logging in into instances
    constraints:
    - custom_constraint: nova.keypair
    default: "OAI-Admin"
  image:
    type: string
    label: Ubuntu Image
    description: Name of the Ubuntu image (needs os-*-config installed)
    constraints:
    - custom_constraint: glance.image
    default: "ubuntu-trusty-os-config"
  extnet:
    type: string
    label: External Network
    description: Name of the external network containing 2 free floating IPs
    default: "Lab3-Internet"
  base_url:
    type: string
    label: Base URL
    description: Base URL to fetch kernel config and database dump from
    default: "https://inostack.vptt.ch"
  run_flavor:
    type: string
    label: Run Flavor
    description: Flavor to use for normal instances
    constraints:
    - custom_constraint: nova.flavor
    default: "m1.tiny"
  build_flavor:
    type: string
    label: Build Flavor
    description: Flavor to use for the building instance
    constraints:
    - custom_constraint: nova.flavor
    default: "m1.medium"
  run_vol_size:
    type: number
    label: Run Volume Size
    description: Volume size in GB to use for normal instances
    default: 5
    constraints:
    - range: { min: 5, max: 10 }
  build_vol_size:
    type: number
    label: Build Volume Size
    description: Volume size in GB to use for the building instance
    default: 20
    constraints:
    - range: { min: 20, max: 50 }
  hss_name:
    type: string
    label: HSS Hostname
    description: Hostname of the HSS instance
    constraints:
    - length: { min: 3, max: 10 }
      description: Hostnameg name must be between 3 and 10 characters
    - allowed_pattern: "[a-z0-9]*"
      description: Hostname contains only lowercase characters and numbers
    default: hss
  mme_name:
    type: string
    label: MME Hostname
    description: Hostname of the MME instance
    constraints:
    - length: { min: 3, max: 10 }
      description: Hostname name must be between 3 and 10 characters
    - allowed_pattern: "[a-z0-9]*"
      description: Hostname contains only lowercase characters and numbers
    default: mme
  spgw_name:
    type: string
    label: SPGW Hostname
    description: Hostname of the SPGW instance
    constraints:
    - length: { min: 3, max: 10 }
      description: Hostname name must be between 3 and 10 characters
    - allowed_pattern: "[a-z0-9]*"
      description: Hostname contains only lowercase characters and numbers
    default: spgw
  build_name:
    type: string
    label: Build Hostname
    description: Hostname of the build instance
    constraints:
    - length: { min: 3, max: 10 }
      description: Hostname name must be between 3 and 10 characters
    - allowed_pattern: "[a-z0-9]*"
      description: Hostname contains only lowercase characters and numbers
    default: build
  dns1:
    type: string
    label: Internal Network DNS 1
    description: Upstream DNS server 1
    default: "8.8.8.8"
  dns2:
    type: string
    label: Internal Network DNS 2
    description: Upstream DNS server 2
    default: "8.8.4.4"
  int_cidr:
    type: string
    label: Internal Network CIDR
    description: Internal network IPv4 Adressing in CIDR notation
    default: 172.16.0.0/24
  enb_cidr:
    type: string
    label: eNB (VPN) CIDR
    description: eNB (VPN) network IPv4 Adressing in CIDR notation
    default: 172.31.0.0/24
  realm:
    type: string
    label: Realm
    description: Realm (depends on database and hostnames used)
    default: inostack
  enb_count:
    type: number
    label: eNB Count
    description: Number of eNBs to support (power of 2)
    default: 4
    constraints:
    - allowed_values:
      - 2
      - 4
      - 8
      - 16
  enb_start:
    type: number
    label: eNB Start Address
    description: Start address of eNBs in eNB VPN network
    default: 10
    constraints:
      - range: { min: 10, max: 230 }
  enb_hostname_prefix:
    type: string
    label: Hostname Prefix for eNBs
    description: Hostname prefix for eNBs
    constraints:
    - length: { min: 3, max: 9 }
      description: Hostname prefix must be between 3 and 9 characters
    - allowed_pattern: "[a-z]*"
      description: Hostname prefix contains only lowercase characters
    default: enb
  spgw_kernel_version:
    type: string
    label: SPWG Kernel Version
    description: Kernel version to use for SPGW instance (>=4.7)
    default: 4.7.7
  ue_dns:
    type: string
    label: UE DNS
    description: DNS pushed to UEs
    default: 8.8.8.8
  ue_cidr:
    type: string
    label: UE CIDR
    description: UE address pool in CIDR notation
    default: 10.10.10.0/24
  ue_mtu:
    type: number
    label: UE MTU
    description: MTU pushed to UEs
    default: 1500
    constraints:
    - range: { min: 1000, max: 8000 }
  db_file:
    type: string
    label: HSS DB File
    description: SQL dump file for HSS database to download from base_url
    default: oai_db_swisscom_ino.sql
  db_pass:
    type: string
    label: HSS DB Password
    description: Password for the HSS database (user hssadmin)
    hidden: true
    default: change_me
  operator_key:
    type: string
    label: Operator Key
    description: Operator Key to use (depends on DB)
    default: "11111111111111111111111111111111"
    constraints:
    - length: { min: 32, max: 32 }
```

```
      description: Operator key is 32 digits
    - allowed_pattern: "[0-9]*"
      description: Operator key is 32 digits
  mcc:
    type: number
    label: MCC
    description: MCC to use (depends on DB)
    default: 228
  mnc:
    type: number
    label: MNC
    description: MNC to use (depends on DB)
    default: 88
  mme_gid:
    type: number
    label: MME_GID
    description: MME_GID to use (depends on DB)
    default: 32768
  mme_code:
    type: number
    label: MME_CODE
    description: MME_CODE to use (depends on DB)
    default: 1
  tac:
    type: number
    label: TAC
    description: TAC to use (depends on DB)
    default: 1
  ca_country:
    type: string
    label: CA Country
    description: Country for Certificates
    default: CH
  ca_state:
    type: string
    label: CA State
    description: State for Certificates
    default: Bern
  ca_city:
    type: string
    label: CA City
    description: City for Certificates
    default: Bern
  ca_company:
    type: string
    label: CA Company
    description: Company for Certificates
    default: Swisscom
  ca_unit:
    type: string
    label: CA Unit
    description: Organisational unit for Certificates
    default: INO
  ca_email:
    type: string
    label: CA Email
    description: Email Address for Certificates
    default: Daniel.Balsiger@swisscom.com
  ssh_pub:
    type: string
    label: SSH Public Key
    description: SSH public key for instance interconnect
    default: "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC6WwxDHQKgvVTqzC2S4+apYFmVTTI3N1+kTnOJZ5+K/Po9D
3Uq89LY2hdRUNcBvOIDW1lGJHLoodyR/W6UaLvT1eTG1le72yCpcliq/nuDc6eUex2Mqz3CMBqWhg
3L+21yme0vlT3w0S+uhkiq8s95OMCYMc60bSTHN/RqLB1o8dsLAqix6W5lDxcwVpl8V7viBTvZBUS
qTHKA0AXJ4uQDdBqHZY9iUcyyKVzcEaOnd4RdAJPAD+au3dKlpfEdKQjxHTV41QM+VkuImQUMe5MQ
7gU4DbHj7YSA3fJhki3jwFHMzFNbYulUgwBdNCFVTcGdQEUib6dXW8LOH59FmUut"
  ssh_priv:
    type: string
    label: SSH Private Key
    description: SSH private key for instance interconnect
    hidden: true
    default: >
      -----BEGIN RSA PRIVATE KEY-----
      MIIEowIBAAKCAQEAulsMQx0CoL1U6swtkuPmqWBZlU0yNzdfpE5ziWefivz6PQ91
      KvPS2NoXUVDXAbziA1tZRiRy6KHckf1ulGi709XkxtZXu9sgqXJYqv57g3OnlHsd
      jKs9wjAaloYNy/ttcpntL5U98NEvroZIqvLPeTjAmDHOtG0kxzf0aiwdaPHbCwKo
      seluZQ8XMFaZfFe74gU72QVEqkxygNAFyeLkA3Qah2WPYlHMsilc3BGjp3eEXQCT
      wA/mrt3SpaXxHSkI8R01eNUDPlZLiJkFDHuTEO4FOA2x4+2EgN3yYZIt48BRzMxT
      W2LpVIMAXTQhVU3BnUBFIm+nV1vCzh+fRZlLrQIDAQABAoIBACnu8OxtK7k1wVTw
      StUB2VaFqsLQ0xrfp/LWAGOL4LeqwzhYMRpQMULAmHygvzDR6t2sgYMqEn1MZtCn
      AWn9wz4gpFElzComXcwjQdaAWxSyQqRDq9uKcOQwZNs4IQSkd/VQs7GAWKbGu37/
      Enz9FDiHz7avhn7NDHiTm7kEYj3JxBCw9QnpuljFu7BZlg1S4eK/rGz7in1hLo62
      gDfGcDgUgmlJ1aG1rYP2bETsXe0necHDOvQe6rlmdTsX89y+q4QxqYGEGdjCoFZZ
      F0mEPbS9//9GS+rER5tuSi3bD3k73zg1+e8lHAjlAR3hNmT3JKOEHB0kmTAJyrpL
      +Gl9WIECgYEA4ekwnl3wT2p2eqmhqC+MoMcUpOtAaoA2ShG3CHcp/U6VcWIMBe7o
      KMaJcfLp3zerAeyzRfR7t1E83alIQb9cRRd1pbhLjM60d+n6HwT0//4SBiShfQ/m
      K7Ch1zOaB4h8O8qWDT9Uz6b0AMJA9S6BZ2We3iOT5ysS6MPOwgQAbCECgYEA0y0n
      4zgq1YFE+56xXAt8/BKqv9nbtNfKganXTXMah+oz546CVUJDPL3z99Zp1w3rUvlJ
      JH2wbnqvcPCuCvFdCPXpowkN5EWUTUrjeYohDXDZPaJ1pYZXeMKFyVgzZFeP+UKK
      82gxY5O+kC1I0b2J2u0gxm74huSgBIRx54ZqDg0CgYEAhQsT+vBPyjVkuTCVZ7s5
      Eqar3cQ+F3qSfmSYan/jVq6lDDU153ifeQQThewNF8xtBaEkoxoskfVh5xj+2Nmd
      uYLrYkF7HN3PIp/FEeeVcf1rF/sSr9hhMXHAnkBhgfY7U+snG34ksHYeVSQRpVNS
      GlaajTBetlGDvVkztscsiIECgYBTd0KPtVCAyLIqPaPePJAu1XX1lDcZeD0LGMUH
      UJpI5BGV0SbEagdHR9DYwT9eB5teVTdKm/8S+5zCJ+6yVomuE/w/O0HpWnLuRc44
      6JZ9yH+ks8SKItoJ2eClHx5Y5575Jwrif+kdcXTdaXihpaeKBzVwGMZUEqMIhgy7
      NM5QNQKBgAT80nwpu/KJasbPd/5vlnWKkk9wW1pJJJyGvyrHvQkHYSJmrGNhci+X
      geZreHUVqcR4VeRP14D71+zIT04r4jcio9deNBRYXdWuAhKVfvEh5iOXfuaBF24e
      m4y0viefFNyys5/BQI0kpCtJirvAtlsQg8ig+2ALnCk5rLNbgui4
      -----END RSA PRIVATE KEY-----

### ORDER OF INPUTS
parameter_groups:
- label: "Passwords and Keys"
  parameters:
  - db_pass
  - ssh_pub
  - ssh_priv
  - key
- label: "Cloud Networks, Flavors, Images and Volumes"
  parameters:
  - extnet
  - int_cidr
  - dns1
  - dns2
  - image
  - run_flavor
  - run_vol_size
  - build_flavor
  - build_vol_size
- label: "FQDNs, Realm, SPGW kernel, HSS Database and EPC Configuration"
  parameters:
  - base_url
  - spgw_kernel_version
  - realm
  - spgw_name
  - mme_name
  - hss_name
  - build_name
  - db_file
  - operator_key
  - mcc
  - mnc
  - mme_gid
  - mme_code
  - tac
- label: "eNB and UE Settings"
  parameters:
  - ue_cidr
  - ue_dns
  - ue_mtu
  - enb_cidr
  - enb_count
  - enb_start
  - enb_hostname_prefix
- label: "Certificate Settings"
  parameters:
  - ca_country
  - ca_state
  - ca_city
  - ca_company
  - ca_unit
  - ca_email

### STACK RESOURCES
resources:
### NETWORKS & ROUTERS
  internal_net:
    type: OS::Neutron::Net
    properties:
      name: OAI-InternalNet

  internal_subnet:
    type: OS::Neutron::Subnet
    depends_on: internal_net
    properties:
      name: OAI-InternalSubnet
      ip_version: 4
      network_id: { get_resource: internal_net }
      dns_nameservers:
```

```
    - { get_param: dns1 }
    - { get_param: dns2 }
   cidr: { get_param: int_cidr }

 router:
  type: OS::Neutron::Router
  properties:
   name: OAI-Router

 router_interface:
  type: OS::Neutron::RouterInterface
  depends_on: [ internal_subnet, internal_net, router ]
  properties:
   subnet: { get_resource: internal_subnet }
   router: { get_resource: router }

 router_gateway:
  type: OS::Neutron::RouterGateway
  depends_on: router
  properties:
   network: { get_param: extnet }
   router_id: { get_resource: router }

 floating_ip:
  type: OS::Neutron::FloatingIP
  depends_on: [ internal_subnet, spgw_port, router_interface ]
  properties:
   floating_network: { get_param: extnet }
   port_id: { get_resource: spgw_port }

### VOLUMES
 build_vol:
  type: OS::Cinder::Volume
  properties:
   name: { get_param: build_name }
   size: { get_param: build_vol_size }
   image: { get_param: image }

 hss_vol:
  type: OS::Cinder::Volume
  properties:
   name: { get_param: hss_name }
   size: { get_param: run_vol_size }
   image: { get_param: image }

 mme_vol:
  type: OS::Cinder::Volume
  properties:
   name: { get_param: mme_name }
   size: { get_param: run_vol_size }
   image: { get_param: image }

 spgw_vol:
  type: OS::Cinder::Volume
  properties:
   name: { get_param: spgw_name }
   size: { get_param: run_vol_size }
```

```
      image: { get_param: image }

### PORTS
  build_port:
    type: OS::Neutron::Port
    depends_on: [ build_secgroup, internal_subnet ]
    properties:
      network_id: { get_resource: internal_net }
      fixed_ips:
        - subnet_id: { get_resource: internal_subnet }
      security_groups: [{ get_resource: build_secgroup }]

  spgw_port:
    type: OS::Neutron::Port
    depends_on: [ spgw_secgroup, internal_subnet ]
    properties:
      network_id: { get_resource: internal_net }
      fixed_ips:
        - subnet_id: { get_resource: internal_subnet }
      security_groups: [{ get_resource: spgw_secgroup }]

  hss_port:
    type: OS::Neutron::Port
    depends_on: [ hss_secgroup, internal_subnet ]
    properties:
      network_id: { get_resource: internal_net }
      fixed_ips:
        - subnet_id: { get_resource: internal_subnet }
      security_groups: [{ get_resource: hss_secgroup }]

  mme_port:
    type: OS::Neutron::Port
    depends_on: [ mme_secgroup, internal_subnet ]
    properties:
      network_id: { get_resource: internal_net }
      fixed_ips:
        - subnet_id: { get_resource: internal_subnet }
      security_groups: [{ get_resource: mme_secgroup }]

### SECURITY GROUPS
  spgw_secgroup:
    type: OS::Neutron::SecurityGroup
    properties:
      description: Add security group rules for SPGW instance
      name: { get_param: spgw_name }
      rules:
        - remote_ip_prefix: 0.0.0.0/0

  build_secgroup:
    type: OS::Neutron::SecurityGroup
    properties:
      description: Add security group rules for build instance
      name: { get_param: build_name }
      rules:
        - remote_ip_prefix: 0.0.0.0/0
          protocol: tcp
          port_range_min: 22
          port_range_max: 22
        - remote_ip_prefix: 0.0.0.0/0
          protocol: icmp

  hss_secgroup:
    type: OS::Neutron::SecurityGroup
    properties:
      description: Add security group rules for HSS instance
      name: { get_param: hss_name }
      rules:
        - remote_ip_prefix: 0.0.0.0/0
          protocol: tcp
          port_range_min: 22
          port_range_max: 22
        - remote_ip_prefix: 0.0.0.0/0
          protocol: icmp
        - remote_ip_prefix: 0.0.0.0/0
          protocol: 132

  mme_secgroup:
    type: OS::Neutron::SecurityGroup
    properties:
      description: Add security group rules for MME instance
      name: { get_param: mme_name }
      rules:
        - remote_ip_prefix: 0.0.0.0/0
          protocol: tcp
          port_range_min: 22
          port_range_max: 22
        - remote_ip_prefix: 0.0.0.0/0
          protocol: icmp
        - remote_ip_prefix: 0.0.0.0/0
          protocol: 132

### INSTANCES

  build_vm:
    type: OS::Nova::Server
    depends_on: [ build_port, build_vol, router_interface ]
    properties:
      name: { get_param: build_name }
      flavor: { get_param: build_flavor }
      key_name: { get_param: key }
      block_device_mapping: [{ device_name: "vda", volume_id : { get_resource
: build_vol }, delete_on_termination : "true" }]
      networks:
        - port: { get_resource: build_port }
      user_data_format: SOFTWARE_CONFIG
      user_data: |
        #!/bin/bash
        echo "195.176.209.235 opnfv.vptt.ch opnfv" >> /etc/hosts

  spgw_vm:
    type: OS::Nova::Server
    depends_on: [ spgw_port, spgw_vol, router_interface ]
    properties:
      name: { get_param: spgw_name }
      flavor: { get_param: run_flavor }
      key_name: { get_param: key }
      block_device_mapping: [{ device_name: "vda", volume_id : { get_resource
: spgw_vol }, delete_on_termination : "true" }]
      networks:
        - port: { get_resource: spgw_port }
      user_data_format: SOFTWARE_CONFIG
      user_data: |
        #!/bin/bash
        echo "195.176.209.235 opnfv.vptt.ch opnfv" >> /etc/hosts

  hss_vm:
    type: OS::Nova::Server
    depends_on: [ hss_port, hss_vol, router_interface ]
    properties:
      name: { get_param: hss_name }
      flavor: { get_param: run_flavor }
      key_name: { get_param: key }
      block_device_mapping: [{ device_name: "vda", volume_id : { get_resource
: hss_vol }, delete_on_termination : "true" }]
      networks:
      - port: { get_resource: hss_port }
      user_data_format: SOFTWARE_CONFIG
      user_data: |
        #!/bin/bash
        echo "195.176.209.235 opnfv.vptt.ch opnfv" >> /etc/hosts

  mme_vm:
    type: OS::Nova::Server
    depends_on: [ mme_port, mme_vol, router_interface ]
    properties:
      name: { get_param: mme_name }
      flavor: { get_param: run_flavor }
      key_name: { get_param: key }
      block_device_mapping: [{ device_name: "vda", volume_id : { get_resource
: mme_vol }, delete_on_termination : "true" }]
      networks:
        - port: { get_resource: mme_port }
      user_data_format: SOFTWARE_CONFIG
      user_data: |
        #!/bin/bash
        echo "195.176.209.235 opnfv.vptt.ch opnfv" >> /etc/hosts

### SOFTWARE CONFIGURATIONS
  etc_hosts:
    type: OS::Heat::SoftwareConfig
    depends_on: [ hss_vm, mme_vm, spgw_vm, build_vm ]
    properties:
      group: script
      inputs:
      - name: realm
        default: { get_param: realm }
      - name: hss_name
        default: { get_param: hss_name }
      - name: mme_name
        default: { get_param: mme_name }
      - name: spgw_name
```

```
        default: { get_param: spgw_name }
      - name: build_name
        default: { get_param: build_name }
      - name: hss_ip
        default: { get_attr: [ hss_vm, first_address ] }
      - name: mme_ip
        default: { get_attr: [ mme_vm, first_address ] }
      - name: spgw_ip
        default: { get_attr: [ spgw_vm, first_address ] }
      - name: build_ip
        default: { get_attr: [ build_vm, first_address ] }
      - name: enb_cidr
        default: { get_param: enb_cidr }
      - name: enb_count
        default: { get_param: enb_count }
      - name: enb_start
        default: { get_param: enb_start }
      - name: enb_hostname_prefix
        default: { get_param: enb_hostname_prefix }
      config: |
        #!/bin/bash
        if [ ! -f /etc/hosts.orig ] ; then cp /etc/hosts /etc/hosts.orig ; fi
        logger "$0: Creating /etc/hosts..."
        cat > /etc/hosts << __EOF
        # this file is generated by heat with os-*-config
        127.0.0.1 localhost

        $spgw_ip $spgw_name.$realm $spgw_name
        $mme_ip $mme_name.$realm $mme_name
        $hss_ip $hss_name.$realm $hss_name
        $spgw_ip $spgw_name.$realm $spgw_name
        $build_ip $build_name.$realm $build_name

        # this is since we have no DNS for opnfv.vptt.ch
        195.176.209.235 opnfv.vptt.ch opnfv

        __EOF
        i=0
        while [ $i -lt $(expr $enb_count) ] ; do
          host_part=$(expr $enb_start + $i)
          echo ${enb_cidr%.*}.$host_part $enb_hostname_prefix$i.$realm
$enb_hostname_prefix$i >> /etc/hosts
          i=$(expr $i + 1)
        done
        cat >> /etc/hosts << __EOF

        __EOF

  ssh_keys:
    type: OS::Heat::SoftwareConfig
    depends_on: [ hss_vm, mme_vm, spgw_vm, build_vm ]
    properties:
      inputs:
      - name: ssh_priv
        default: { get_param: ssh_priv }
      - name: ssh_pub
        default: { get_param: ssh_pub }
      group: script
      config: |
        #!/bin/bash
        logger "$0: Creating SSH keys..."
        mkdir -p /root/.ssh
        echo $ssh_priv > /root/.ssh/id_rsa
        sed -e 's|-----BEGIN RSA PRIVATE KEY-----|-----BEGIN_RSA_PRIVATE_KEY-
----|' \
            -e 's|-----END RSA PRIVATE KEY-----|-----END_RSA_PRIVATE_KEY-----
|' \
            -e 's| |\n|g' \
            -e 's|-----BEGIN_RSA_PRIVATE_KEY-----|-----BEGIN RSA PRIVATE KEY-
----|' \
            -e 's|-----END_RSA_PRIVATE_KEY-----|-----END RSA PRIVATE KEY-----
|' \
            -i /root/.ssh/id_rsa
        echo $ssh_pub > /root/.ssh/id_rsa.pub
        echo $ssh_pub > /root/.ssh/authorized_keys
        chmod 0644 /root/.ssh/id_rsa.pub
        chmod 0600 /root/.ssh/id_rsa
        chmod 0644 /root/.ssh/authorized_keys
        cat > /root/.ssh/config << __EOF
        StrictHostKeyChecking no
        UserKnownHostsFile /dev/null
        __EOF

  vpn_server:
    type: OS::Heat::SoftwareConfig
    depends_on: spgw_vm
    properties:
      group: script
      inputs:
      - name: mme_name
        default: { get_param: mme_name }
      - name: spgw_name
        default: { get_param: spgw_name }
      - name: enb_cidr
        default: { get_param: enb_cidr }
      - name: enb_cidr
        default: { get_param: enb_cidr }
      - name: enb_count
        default: { get_param: enb_count }
      - name: enb_start
        default: { get_param: enb_start }
      - name: enb_hostname_prefix
        default: { get_param: enb_hostname_prefix }
      - name: realm
        default: { get_param: realm }
      - name: ca_country
        default: { get_param: ca_country }
      - name: ca_state
        default: { get_param: ca_state }
      - name: ca_city
        default: { get_param: ca_city }
      - name: ca_company
        default: { get_param: ca_company }
      - name: ca_unit
```

```
        default: { get_param: ca_unit }
      - name: ca_email
        default: { get_param: ca_email }
      config: |
        #!/bin/bash
        logger "$0: Creating /etc/openvpn/clients.txt"
        mkdir -p /etc/openvpn
        touch /etc/openvpn/clients.txt
        echo $mme_name,${enb_cidr%.*}.2 > /etc/openvpn/clients.txt
        i=0
        net=$enb_cidr
        while [ $i -lt $(expr $enb_count) ] ; do
          host_part=$(expr $enb_start + $i)
          echo $enb_hostname_prefix$i,${net%.*}.$host_part >>
/etc/openvpn/clients.txt
          i=$(expr $i + 1)
        done
        logger "$0: Installing openvpn and creating VPN keys..."
        DEBIAN_FRONTEND=noninteractive apt-get install -q -y openvpn easy-rsa
        mkdir -p /etc/openvpn/easy-rsa
        cp -R /usr/share/easy-rsa/* /etc/openvpn/easy-rsa
        mkdir -p /etc/openvpn/easy-rsa/keys
        cd /etc/openvpn/easy-rsa
        export EASY_RSA="${EASY_RSA:-.}"
        source ./vars
        export KEY_COUNTRY="$ca_country"
        export KEY_PROVINCE="$ca_state"
        export KEY_CITY="$ca_city"
        export KEY_ORG="$ca_company"
        export KEY_EMAIL="$ca_email"
        export KEY_OU="$ca_unit"
        export KEY_NAME="VPN_CA"
        ./clean-all
        "$EASY_RSA/pkitool" --initca
        export KEY_NAME=$spgw_name
        "$EASY_RSA/pkitool" --server $spgw_name
        ./build-dh
        openvpn --genkey --secret keys/hmac.key
        for client in $(cat /etc/openvpn/clients.txt) ; do
          export KEY_NAME=${client%,*}
          "$EASY_RSA/pkitool" ${client%,*}
        done
        cd -
        logger "$0: Creating /etc/openvpn/server.conf..."
        cat > /etc/openvpn/server.conf << __EOF
        daemon
        proto udp
        port 1194
        dev tun
        ca /etc/openvpn/easy-rsa/keys/ca.crt
        cert /etc/openvpn/easy-rsa/keys/$spgw_name.crt
        key /etc/openvpn/easy-rsa/keys/$spgw_name.key
        dh /etc/openvpn/easy-rsa/keys/dh2048.pem
        tls-auth /etc/openvpn/easy-rsa/keys/hmac.key 0
        server ${net%/*} 255.255.255.0
        topology subnet
        client-to-client
        ifconfig-pool-persist /etc/openvpn/clients.txt
        keepalive 10 120
        cipher AES-128-CBC
        comp-lzo
        user nobody
        group nogroup
        persist-key
        persist-tun
        verb 3
        __EOF
        logger "$0: Starting VPN server..."
        systemctl enable openvpn@server.service
        systemctl restart openvpn@server.service
        service openvpn restart

  vpn_client:
    type: OS::Heat::SoftwareConfig
    depends_on: mme_vm
    properties:
      group: script
      inputs:
      - name: spgw_name
        default: { get_param: spgw_name }
      - name: client_name
        default: { get_param: mme_name }
      - name: server_ip
        default: { get_attr: [ spgw_vm, first_address ] }
      config: |
        #!/bin/bash
        logger "$0: Installing VPN client..."
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install openvpn
        cat > /etc/openvpn/client.conf << __EOF
        daemon
        client
        proto udp
        dev tun
        nobind
        remote $server_ip 1194
        ca /etc/openvpn/keys/ca.crt
        cert /etc/openvpn/keys/$client_name.crt
        key /etc/openvpn/keys/$client_name.key
        tls-auth /etc/openvpn/keys/hmac.key 1
        ns-cert-type server
        cipher AES-128-CBC
        comp-lzo
        user nobody
        group nogroup
        persist-key
        persist-tun
        verb 3
        mute 20
        __EOF
        cd /tmp
        ssh $spgw_name "tar -c /etc/openvpn/easy-rsa/keys" | dd of=keys.tar
        tar -xvf keys.tar
        mv etc/openvpn/easy-rsa/keys .
        install -v -d -m 0700 -o root -g root /etc/openvpn/keys
        install -v -m 0400  -o root -g root keys/ca.crt
/etc/openvpn/keys/ca.crt
        install -v -m 0400  -o root -g root keys/hmac.key
/etc/openvpn/keys/hmac.key
        install -v -m 0400  -o root -g root keys/$client_name.crt
/etc/openvpn/keys/$client_name.crt
        install -v -m 0400  -o root -g root keys/$client_name.key
/etc/openvpn/keys/$client_name.key
        rm etc keys keys.tar -rf
        systemctl enable openvpn@client.service
        systemctl restart openvpn@client.service
        service openvpn restart

  update_system:
    type: OS::Heat::SoftwareConfig
    depends_on: [ hss_vm, mme_vm, spgw_vm, build_vm ]
    properties:
      group: script
      config: |
        #!/bin/bash
        logger "$0: Updating system..."
        DEBIAN_FRONTEND=noninteractive apt-get -q -y update
        DEBIAN_FRONTEND=noninteractive apt-get -q -y dist-upgrade
        DEBIAN_FRONTEND=noninteractive apt-get -q -y autoremove
        DEBIAN_FRONTEND=noninteractive apt-get -q -y autoclean
        DEBIAN_FRONTEND=noninteractive apt-get -q -y clean
        logger "$0: Installing git screen and curl..."
        DEBIAN_FRONTEND=noninteractive apt-get -q -y install curl screen git

  compile_kernel:
    type: OS::Heat::SoftwareConfig
    depends_on: build_vm
    properties:
      group: script
      inputs:
      - name: spgw_kernel_version
        default: { get_param: spgw_kernel_version }
      - name: base_url
        default: { get_param: base_url }
      config: |
        #!/bin/bash
        cd /usr/src
        DEBIAN_FRONTEND=noninteractive apt-get -q -y install xz-utils build-
essential wget libncurses5-dev libssl-dev bc
        DEBIAN_FRONTEND=noninteractive apt-get -q -y build-dep linux-image-
$(uname -r)
        wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-
$spgw_kernel_version.tar.xz
        tar -xf linux-$spgw_kernel_version.tar.xz
        curl -s -O $base_url/config-$spgw_kernel_version-gtp
        cp config-$spgw_kernel_version-gtp linux-$spgw_kernel_version/.config
        cd linux-$spgw_kernel_version
        make oldconfig
        make -j`nproc`
        make INSTALL_MOD_STRIP=1 modules_install
        make install
        cd /root
        # should reboot to use new kernel. Hangs on ubuntu14
        #shutdown -r +1
        #sleep 55

  create_freediameter_certs:
    type: OS::Heat::SoftwareConfig
    depends_on: build_vm
    properties:
      group: script
      inputs:
      - name: realm
        default: { get_param: realm }
      - name: mme_name
        default: { get_param: mme_name }
      - name: hss_name
        default: { get_param: hss_name }
      - name: ca_country
        default: { get_param: ca_country }
      - name: ca_state
        default: { get_param: ca_state }
      - name: ca_city
        default: { get_param: ca_city }
      - name: ca_company
        default: { get_param: ca_company }
      - name: ca_unit
        default: { get_param: ca_unit }
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install openssl
        certs_dir=/root/certs
        make_one_cert() {
          name=$1
          openssl genrsa -out $name.key.pem 1024
          openssl req -new -batch -out $name.csr.pem -key $name.key.pem -subj
/CN=$name.$realm/C=$ca_country/ST=$ca_state/L=$ca_city/O=$ca_company/OU=$ca_u
nit
          openssl ca -cert cacert.pem -keyfile cakey.pem -in $name.csr.pem -
out $name.cert.pem -outdir . -batch
        }
        mkdir -p $certs_dir
        cd $certs_dir
        mkdir -p $certs_dir/demoCA/
        touch $certs_dir/demoCA/index.txt
        echo 01 > $certs_dir/demoCA/serial
        openssl req -new -batch -x509 -days 3650 -nodes -newkey rsa:1024 -out
cacert.pem -keyout cakey.pem -subj
/CN=$realm/C=$ca_country/ST=$ca_state/L=$ca_city/O=$ca_company/OU=$ca_unit
        make_one_cert $hss_name
        make_one_cert $mme_name

  eurecom_certs:
    type: OS::Heat::SoftwareConfig
    depends_on: build_vm
    properties:
      group: script
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install openssl
        if [ ! -f /etc/ssl/certs/ca-certificates.crt.bak ] ; then cp
/etc/ssl/certs/ca-certificates.crt{,.bak} ; fi
        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

  compile_nettle:
    type: OS::Heat::SoftwareConfig
    depends_on: build_vm
    properties:
      group: script
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install autoconf
automake build-essential libgmp-dev wget
        cd /tmp
        rm -rf /tmp/nettle-2.5.tar.gz /tmp/nettle-2.5
        wget https://ftp.gnu.org/gnu/nettle/nettle-2.5.tar.gz
        tar -xzf /tmp/nettle-2.5.tar.gz
        cd /tmp/nettle-2.5
        ./configure --disable-openssl --enable-shared --prefix=/usr/local
        make
        make check
        make install
        cd /tmp
        rm -rf /tmp/nettle-2.5.tar.gz /tmp/nettle-2.5
        ldconfig

  compile_gnutls:
    type: OS::Heat::SoftwareConfig
    depends_on: build_vm
    properties:
      group: script
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q purge libgnutls-dev
'libgnutlsxx2?'
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install libtasn1-6-dev
libp11-kit-dev libtspi-dev libidn11-dev wget
        cd /tmp
        rm -rf /tmp/gnutls-3.1.23.tar.xz* /tmp/gnutls-3.1.23
        wget ftp://ftp.gnutls.org/gcrypt/gnutls/v3.1/gnutls-3.1.23.tar.xz
        tar -xJf /tmp/gnutls-3.1.23.tar.xz
        cd /tmp/gnutls-3.1.23
        ./configure --prefix=/usr/local
        make
        make install
        cd /tmp
        rm -rf /tmp/gnutls-3.1.23 /tmp/gnutls-3.1.23.tar.xz
        ldconfig

  compile_freediameter:
    type: OS::Heat::SoftwareConfig
    depends_on: build_vm
    properties:
      group: script
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install autoconf
automake bison flex build-essential cmake libsctp-dev libidn11-dev libgcrypt-
dev
        cd /tmp
        rm -rf /tmp/freediameter
        git clone https://gitlab.eurecom.fr/oai/freediameter.git -b eurecom-
1.2.0
        cd /tmp/freediameter
        mkdir build
        cd build
        cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr/local ../
        make
        make install
        cd /tmp
        rm -rf /tmp/freediameter
        ldconfig

  compile_asn1c:
    type: OS::Heat::SoftwareConfig
    depends_on: build_vm
    properties:
      group: script
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install autoconf
automake bison flex build-essential libtool
        cd /tmp
        rm -rf /tmp/asn1c
        git clone https://gitlab.eurecom.fr/oai/asn1c.git
        cd /tmp/asn1c
        ./configure --prefix=/usr/local
        make
        make install
        cd /tmp
        rm -rf /tmp/asn1c

  compile_libgtpnl:
    type: OS::Heat::SoftwareConfig
    depends_on: build_vm
    properties:
      group: script
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install autoconf
automake bison flex build-essential libtool libmnl-dev
        cd /tmp
        rm -rf /tmp/libgtpnl
        git clone git://git.osmocom.org/libgtpnl
        cd /tmp/libgtpnl
        autoreconf -fi
        ./configure --prefix=/usr/local
        make
        make install
        cd /tmp
        rm -rf /tmp/libgtpnl
        ldconfig

  compile_openair_cn:
    type: OS::Heat::SoftwareConfig
    depends_on: build_vm
    properties:
      group: script
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install autoconf
automake bison flex build-essential cmake libsctp-dev libconfig8-dev libgmp-
dev libsctp-dev libssl-dev libxml2-dev mscgen openssl mariadb-client
libmysqlclient-dev check
        cd /tmp
        rm -rf openair-cn
        git clone https://gitlab.eurecom.fr/oai/openair-cn.git
        cd /tmp/openair-cn/SCRIPTS
        ./build_mme --clean
        ./build_mme --clean --daemon
        ./build_spgw --clean
        ./build_spgw --clean --daemon
        ./build_hss --clean
        ./build_hss --clean --daemon

  mme_conf:
    type: OS::Heat::SoftwareConfig
    depends_on: mme_vm
    properties:
      group: script
      inputs:
      - name: hss_name
        default: { get_param: hss_name }
      - name: mme_name
        default: { get_param: mme_name }
      - name: realm
        default: { get_param: realm }
      - name: enb_count
        default: { get_param: enb_count }
      - name: int_cidr
        default: { get_param: int_cidr }
      - name: enb_cidr
        default: { get_param: enb_cidr }
      - name: mcc
        default: { get_param: mcc }
      - name: mnc
        default: { get_param: mnc }
      - name: tac
        default: { get_param: tac }
      - name: mme_gid
        default: { get_param: mme_gid }
      - name: mme_code
        default: { get_param: mme_code }
      - name: spgw_ip
        default: { get_attr: [ spgw_vm, first_address ] }
      - name: mme_ip
        default: { get_attr: [ mme_vm, first_address ] }
      - name: hss_ip
        default: { get_attr: [ hss_vm, first_address ] }
      config: |
        #!/bin/bash
        logger "$0: Creating MME configuration..."
        mkdir -p /etc/oai
        intnet=$int_cidr
        enbnet=$enb_cidr
        cat > /etc/oai/mme.conf <<__EOF
        MME :
        {
          REALM                                  = "$realm";
          MAXENB                                 = $enb_count;
          MAXUE                                  = 16;
          RELATIVE_CAPACITY                      = 10;
          EMERGENCY_ATTACH_SUPPORTED             = "no";
          UNAUTHENTICATED_IMSI_SUPPORTED         = "no";
          EPS_NETWORK_FEATURE_SUPPORT_IMS_VOICE_OVER_PS_SESSION_IN_S1      =
"no";
          EPS_NETWORK_FEATURE_SUPPORT_EMERGENCY_BEARER_SERVICES_IN_S1_MODE =
"no";
          EPS_NETWORK_FEATURE_SUPPORT_LOCATION_SERVICES_VIA_EPC            =
"no";
          EPS_NETWORK_FEATURE_SUPPORT_EXTENDED_SERVICE_REQUEST             =
"no";
          IP_CAPABILITY                                                    =
"IPV4V6";
          MME_STATISTIC_TIMER                    = 10;
          INTERTASK_INTERFACE :
          {
            ITTI_QUEUE_SIZE                      = 2000000;
          };
          S6A :
          {
            S6A_CONF                             = "/etc/oai/mme_fd.conf";
            HSS_HOSTNAME                         = "$hss_name";
          };
          SCTP :
          {
            SCTP_INSTREAMS                       = 8;
            SCTP_OUTSTREAMS                      = 8;
          };
          S1AP :
          {
            S1AP_OUTCOME_TIMER                   = 10;
          };
          GUMMEI_LIST = (
            { MCC="$mcc" ; MNC="$mnc"; MME_GID="$mme_gid" ;
MME_CODE="$mme_code"; }
          );
          TAI_LIST = (
            { MCC="$mcc" ; MNC="$mnc"; TAC="$tac"; }
          );
          NAS :
          {
            ORDERED_SUPPORTED_INTEGRITY_ALGORITHM_LIST = [ "EIA2" , "EIA1" ,
"EIA0" ];
            ORDERED_SUPPORTED_CIPHERING_ALGORITHM_LIST = [ "EEA0" , "EEA1" ,
"EEA2" ];
            T3402                                = 1  #in minutes
            T3412                                = 54 #in minutes
            T3422                                = 6
            T3450                                = 6
            T3460                                = 6
            T3470                                = 6
            T3485                                = 8
            T3486                                = 8
            T3489                                = 4
            T3495                                = 8
          };
          LOGGING :
          {
            OUTPUT                               = "CONSOLE";
            COLOR                                = "no";
            SCTP_LOG_LEVEL                       = "TRACE";
            S1AP_LOG_LEVEL                       = "TRACE";
            NAS_LOG_LEVEL                        = "TRACE";
            MME_APP_LOG_LEVEL                    = "TRACE";
            S6A_LOG_LEVEL                        = "TRACE";
            UTIL_LOG_LEVEL                       = "TRACE";
            MSC_LOG_LEVEL                        = "ERROR";
            ITTI_LOG_LEVEL                       = "ERROR";
            ASN1_VERBOSITY                       = "none";
          };
          NETWORK_INTERFACES :
          {
            MME_INTERFACE_NAME_FOR_S1_MME        = "eth0";
            MME_IPV4_ADDRESS_FOR_S1_MME          =
"${enbnet%.*}.2/${enbnet#*/}";
            MME_INTERFACE_NAME_FOR_S11_MME       = "eth0";
            MME_IPV4_ADDRESS_FOR_S11_MME         = "$mme_ip/${intnet#*/}";
            MME_PORT_FOR_S11_MME                 = 2123;
          };
        };
        S-GW :
        {
          SGW_IPV4_ADDRESS_FOR_S11                = "$spgw_ip/${intnet#*/}";
        };
        __EOF
        cat > /etc/oai/mme_fd.conf << __EOF
        Identity       = "$mme_name.$realm";
        Realm          = "$realm";
        TLS_Cred       = "/etc/oai/$mme_name.cert.pem",
"/etc/oai/$mme_name.key.pem";
        TLS_CA         = "/etc/oai/cacert.pem";
        AppServThreads = 4;
        SCTP_streams   = 8;
        LoadExtension  = "dict_nas_mipv6.fdx";
        LoadExtension  = "dict_s6a.fdx";
        No_TCP;
        No_IPv6;
        NoRelay;
        ConnectPeer= "$hss_name.$realm" { ConnectTo = "$hss_ip"; No_IPv6;
No_TLS; port = 3868; realm = "$realm";};
        __EOF

  spgw_conf:
    type: OS::Heat::SoftwareConfig
    depends_on: spgw_vm
    properties:
      group: script
      inputs:
      - name: ue_cidr
        default: { get_param: ue_cidr }
      - name: ue_dns
        default: { get_param: ue_dns }
      - name: ue_mtu
        default: { get_param: ue_mtu }
      - name: int_cidr
        default: { get_param: int_cidr }
      - name: enb_cidr
        default: { get_param: enb_cidr }
      - name: spgw_ip
        default: { get_attr: [ spgw_vm, first_address ] }
      config: |
        #!/bin/bash
        logger "$0: Creating SPGW configuration..."
        mkdir -p /etc/oai
        intnet=$int_cidr
        enbnet=$enb_cidr
        cat > /etc/oai/spgw.conf << __EOF
        S-GW :
        {
          NETWORK_INTERFACES :
          {
            SGW_INTERFACE_NAME_FOR_S11               = "eth0";
            SGW_IPV4_ADDRESS_FOR_S11                 =
"$spgw_ip/${intnet#*/}";
            SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP     = "tun0";
            SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP       =
"${enbnet%.*}.1/${enbnet#*/}";
            SGW_IPV4_PORT_FOR_S1U_S12_S4_UP          = 2152;
            SGW_INTERFACE_NAME_FOR_S5_S8_UP          = "none";
            SGW_IPV4_ADDRESS_FOR_S5_S8_UP            = "0.0.0.0/24";
          };

          INTERTASK_INTERFACE :
          {
            ITTI_QUEUE_SIZE                          = 2000000;
          };

          LOGGING :
          {
            OUTPUT                                   = "CONSOLE";
            THREAD_SAFE                              = "yes";
            COLOR                                    = "no";
            UDP_LOG_LEVEL                            = "TRACE";
            GTPV1U_LOG_LEVEL                         = "TRACE";
            GTPV2C_LOG_LEVEL                         = "TRACE";
            SPGW_APP_LOG_LEVEL                       = "TRACE";
            S11_LOG_LEVEL                            = "TRACE";
          };
        };
        P-GW =
        {
          NETWORK_INTERFACES :
          {
            PGW_INTERFACE_NAME_FOR_S5_S8             = "none";
            PGW_INTERFACE_NAME_FOR_SGI               = "eth0";
            PGW_MASQUERADE_SGI                       = "yes";
            UE_TCP_MSS_CLAMPING                      = "no";
          };

          IP_ADDRESS_POOL :
          {
            IPV4_LIST = (
                                                     "$ue_cidr"
                        );
          };

          DEFAULT_DNS_IPV4_ADDRESS                   = "$ue_dns";
          DEFAULT_DNS_SEC_IPV4_ADDRESS               = "8.8.4.4";
          FORCE_PUSH_PROTOCOL_CONFIGURATION_OPTIONS  = "no";
          UE_MTU                                     = $ue_mtu;
        };
        __EOF

  hss_conf:
    type: OS::Heat::SoftwareConfig
    depends_on: hss_vm
    properties:
      group: script
      inputs:
      - name: db_pass
        default: { get_param: db_pass }
      - name: operator_key
        default: { get_param: operator_key }
      - name: hss_name
        default: { get_param: hss_name }
      - name: realm
        default: { get_param: realm }
      - name: hss_ip
        default: { get_attr: [ hss_vm, first_address ] }
      config: |
        #!/bin/bash
        logger "$0: Creating HSS configuration..."
        mkdir -p /etc/oai
        cat > /etc/oai/hss.conf << __EOF
        HSS :
        {
          MYSQL_server = "127.0.0.1";
          MYSQL_user   = "hssadmin";
          MYSQL_pass   = "$db_pass";
          MYSQL_db     = "oai_db";
          OPERATOR_key = "$operator_key";
```

```
          RANDOM       = "true";
          FD_conf      = "/etc/oai/hss_fd.conf";
        };
        __EOF
        cat > /etc/oai/hss_fd.conf << __EOF
        Identity       = "$hss_name.$realm";
        Realm          = "$realm";
        TLS_Cred       = "/etc/oai/$hss_name.cert.pem",
"/etc/oai/$hss_name.key.pem";
        TLS_CA         = "/etc/oai/cacert.pem";
        AppServThreads = 4;
        SCTP_streams   = 8;
        ListenOn       = "$hss_ip";
        Port           = 3868;
        SecPort        = 5868;
        LoadExtension  = "acl_wl.fdx" : "/etc/oai/hss_acl.conf";
        LoadExtension  = "dict_nas_mipv6.fdx";
        LoadExtension  = "dict_s6a.fdx";
        No_TCP;
        No_IPv6;
        NoRelay;
        __EOF
        cat > /etc/oai/hss_acl.conf <<__EOF
        ALLOW_OLD_TLS   *.$realm
        __EOF

  install_nettle:
    type: OS::Heat::SoftwareConfig
    depends_on: [ hss_vm, mme_vm ]
    properties:
      group: script
      inputs:
      - name: build_name
        default: { get_param: build_name }
      config: |
        #!/bin/bash
        scp -r $build_name:/usr/local/lib /tmp
        install -v -m 0644 -o root -g root /tmp/lib/libnettle.so.4.4
/tmp/lib/libhogweed.so.2.2 /usr/local/lib/
        ln -sfv libnettle.so.4.4 /usr/local/lib/libnettle.so.4
        ln -sfv libnettle.so.4.4 /usr/local/lib/libnettle.so
        ln -sfv libhogweed.so.2.2 /usr/local/lib/libhogweed.so.2
        ln -sfv libhogweed.so.2.2 /usr/local/lib/libhogweed.so
        ldconfig
        rm /tmp/lib -rf

  install_gnutls:
    type: OS::Heat::SoftwareConfig
    depends_on: [ hss_vm, mme_vm ]
    properties:
      group: script
      inputs:
      - name: build_name
        default: { get_param: build_name }
      config: |
        #!/bin/bash
        scp -r $build_name:/usr/local/lib /tmp
        install -v -m 0644 -o root -g root /tmp/lib/libgnutls.so.28.21.3
/usr/local/lib/
        ln -sfv libgnutls.so.28.21.3 /usr/local/lib/libgnutls.so.28
        ln -sfv libgnutls.so.28.21.3 /usr/local/lib/libgnutls.so
        ldconfig
        rm /tmp/lib -rf

  install_freediameter:
    type: OS::Heat::SoftwareConfig
    depends_on: [ hss_vm, mme_vm ]
    properties:
      group: script
      inputs:
      - name: build_name
        default: { get_param: build_name }
      config: |
        #!/bin/bash
        scp -r $build_name:/usr/local/lib /tmp
        install -v -m 0644 -o root -g root /tmp/lib/libfdcore.so.1.2.0
/tmp/lib/libfdproto.so.1.2.0 /usr/local/lib/
        ln -sfv libfdproto.so.1.2.0 /usr/local/lib/libfdproto.so.6
        ln -sfv libfdproto.so.6 /usr/local/lib/libfdproto.so
        ln -sfv libfdcore.so.1.2.0 /usr/local/lib/libfdcore.so.6
        ln -sfv libfdcore.so.6 /usr/local/lib/libfdcore.so
        install -v -d -m 0755 -o root -g root /usr/local/lib/freeDiameter
        install -v -m 0644 -o root -g root /tmp/lib/freeDiameter/*
/usr/local/lib/freeDiameter
        ldconfig
        rm /tmp/lib -rf

  install_libgtpnl:
    type: OS::Heat::SoftwareConfig
    depends_on: spgw_vm
    properties:
      group: script
      inputs:
      - name: build_name
        default: { get_param: build_name }
      config: |
        #!/bin/bash
        scp -r $build_name:/usr/local/lib /tmp
        install -v -m 0644 -o root -g root /tmp/lib/libgtpnl.so.0.0.0
/usr/local/lib/
        ln -sfv libgtpnl.so.0.0.0 /usr/local/lib/libgtpnl.so.0
        ln -sfv libgtpnl.so.0.0.0 /usr/local/lib/libgtpnl.so
        ldconfig
        rm /tmp/lib -rf

  install_kernel:
    type: OS::Heat::SoftwareConfig
    depends_on: spgw_vm
    properties:
      group: script
      inputs:
      - name: spgw_kernel_version
        default: { get_param: spgw_kernel_version }
      - name: build_name
```

```
        default: { get_param: build_name }
      config: |
        #!/bin/bash
        #sleep 90 # wait for build to reboot  # we dont reboot on ubuntu14
        scp $build_name:/boot/*$spgw_kernel_version* /tmp
        install -o root -g root -m 0644 -v /tmp/*$spgw_kernel_version* /boot/
        rm /tmp/*$spgw_kernel_version*
        ssh $build_name "tar -c /lib/modules/$spgw_kernel_version" | dd
of=/tmp/modules.tar
        cd /tmp
        tar -xf modules.tar
        cp -av lib/modules/$spgw_kernel_version /lib/modules
        rm modules.tar lib -rf
        update-initramfs -c -k $spgw_kernel_version
        update-grub
        # should reboot to use new kernel. Hangs on ubuntu14
        #shutdown -r +1
        #sleep 55

  install_freediameter_certs:
    type: OS::Heat::SoftwareConfig
    depends_on: [ hss_vm, mme_vm ]
    properties:
      group: script
      inputs:
      - name: hostname
      - name: build_name
        default: { get_param: build_name }
      config: |
        #!/bin/bash
        scp -r $build_name:/root/certs /tmp
        install -v -m 0600 -o root -g root /tmp/certs/cacert.pem /etc/oai
        install -v -m 0600 -o root -g root /tmp/certs/$hostname.key.pem
/etc/oai
        install -v -m 0600 -o root -g root /tmp/certs/$hostname.cert.pem
/etc/oai
        rm /tmp/certs -rf

  install_database:
    type: OS::Heat::SoftwareConfig
    depends_on: hss_vm
    properties:
      group: script
      inputs:
      - name: db_file
        default: { get_param: db_file }
      - name: base_url
        default: { get_param: base_url }
      - name: db_pass
        default: { get_param: db_pass }
      config: |
        #!/bin/bash
        # this was the only way I found to let it work on xenial and trusty
        function reply_mysql_server_questions() {
          echo
          echo
          echo
        }
        reply_mysql_server_questions | DEBIAN_FRONTEND=noninteractive apt-get
-y -q install autoconf automake bison flex build-essential cmake libsctp-dev
libconfig8-dev libgmp-dev libsctp-dev libssl-dev libxml2-dev mscgen openssl
mariadb-server mariadb-client libmysqlclient-dev check
        sleep 5
        echo "CREATE DATABASE oai_db; GRANT ALL PRIVILEGES ON oai_db.* TO
'hssadmin'@'localhost' IDENTIFIED BY '$db_pass';" | mysql -u root
        cd /root
        curl -s -O $base_url/$db_file
        mysql -u hssadmin -p$db_pass -D oai_db < /root/$db_file

  install_hss:
    type: OS::Heat::SoftwareConfig
    depends_on: hss_vm
    properties:
      group: script
      inputs:
      - name: build_name
        default: { get_param: build_name }
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install libconfig9
libsctp1 libtspi1 libgmp10
        scp $build_name:/usr/local/bin/oai_hss /tmp
        scp $build_name:/usr/sbin/oai_hssd /tmp
        install -v -m 0755 -o root -g root /tmp/oai_hss /tmp/oai_hssd
/usr/local/sbin
        rm /tmp/oai_hss*

  install_mme:
    type: OS::Heat::SoftwareConfig
    depends_on: mme_vm
    properties:
      group: script
      inputs:
      - name: build_name
        default: { get_param: build_name }
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install libconfig9
libsctp1 libtspi1 libgmp10
        scp $build_name:/usr/local/bin/mme /tmp
        scp $build_name:/usr/sbin/mmed /tmp
        install -v -m 0755 -o root -g root /tmp/mme /tmp/mmed /usr/local/sbin
        rm /tmp/mme*

  install_spgw:
    type: OS::Heat::SoftwareConfig
    depends_on: spgw_vm
    properties:
      group: script
      inputs:
      - name: build_name
        default: { get_param: build_name }
      config: |
        #!/bin/bash
        DEBIAN_FRONTEND=noninteractive apt-get -y -q install libconfig9
libmnl0
        scp $build_name:/usr/local/bin/spgw /tmp
        scp $build_name:/usr/sbin/spgwd /tmp
        install -v -m 0755 -o root -g root /tmp/spgw /tmp/spgwd
/usr/local/sbin
        rm /tmp/spgw*

### BUILD DEPLOYMENTS
  create_etc_hosts_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: etc_hosts
    properties:
      config:
        get_resource: etc_hosts
      server:
        get_resource: build_vm

  update_system_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_build, update_system ]
    properties:
      config:
        get_resource: update_system
      server:
        get_resource: build_vm

  ssh_keys_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_build, update_system_build, ssh_keys ]
    properties:
      config:
        get_resource: ssh_keys
      server:
        get_resource: build_vm

  compile_kernel_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_build, compile_kernel ]
    properties:
      config:
        get_resource: compile_kernel
      server:
        get_resource: build_vm

  eurecom_certs_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ compile_kernel_build, eurecom_certs ]
    properties:
      config:
        get_resource: eurecom_certs
      server:
        get_resource: build_vm

  create_freediameter_certs_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ compile_kernel_build, create_freediameter_certs ]
    properties:
      config:
        get_resource: create_freediameter_certs
      server:
        get_resource: build_vm

  compile_libgtpnl_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ compile_kernel_build, compile_libgtpnl ]
    properties:
      config:
        get_resource: compile_libgtpnl
      server:
        get_resource: build_vm

  compile_nettle_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ compile_kernel_build, compile_nettle ]
    properties:
      config:
        get_resource: compile_nettle
      server:
        get_resource: build_vm

  compile_gnutls_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ compile_nettle_build, compile_gnutls ]
    properties:
      config:
        get_resource: compile_gnutls
      server:
        get_resource: build_vm

  compile_freediameter_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ compile_nettle_build, compile_gnutls_build,
eurecom_certs_build, compile_freediameter ]
    properties:
      config:
        get_resource: compile_freediameter
      server:
        get_resource: build_vm

  compile_asn1c_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ compile_freediameter_build, eurecom_certs_build,
compile_asn1c ]
    properties:
      config:
        get_resource: compile_asn1c
      server:
        get_resource: build_vm

  compile_openair_cn_build:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ compile_freediameter_build, compile_libgtpnl_build,
compile_asn1c_build, create_freediameter_certs_build, eurecom_certs_build,
compile_openair_cn ]
    properties:
      config:
        get_resource: compile_openair_cn
      server:
        get_resource: build_vm

### HSS DEPLOYMENTS
  create_etc_hosts_hss:
    type: OS::Heat::SoftwareDeployment
    depends_on: etc_hosts
    properties:
      config:
        get_resource: etc_hosts
      server:
        get_resource: hss_vm

  update_system_hss:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_hss, update_system ]
    properties:
      config:
        get_resource: update_system
      server:
        get_resource: hss_vm

  create_hss_conf:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_hss, update_system_hss, hss_conf ]
    properties:
      config:
        get_resource: hss_conf
      server:
        get_resource: hss_vm

  ssh_keys_hss:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_build, update_system_build, ssh_keys ]
    properties:
      config:
        get_resource: ssh_keys
      server:
        get_resource: hss_vm

  install_database_hss:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_hss, create_hss_conf, install_database ]
    properties:
      config:
        get_resource: install_database
      server:
        get_resource: hss_vm

  install_nettle_hss:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_hss, ssh_keys_build, compile_openair_cn_build,
install_nettle ]
    properties:
      config:
        get_resource: install_nettle
      server:
        get_resource: hss_vm

  install_gnutls_hss:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_hss, ssh_keys_build, compile_openair_cn_build,
install_nettle_hss, install_gnutls ]
    properties:
      config:
        get_resource: install_gnutls
      server:
        get_resource: hss_vm

  install_freediameter_hss:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_hss, ssh_keys_build, compile_openair_cn_build,
install_nettle_hss, install_gnutls_hss, install_freediameter ]
    properties:
      config:
        get_resource: install_freediameter
      server:
        get_resource: hss_vm

  install_freediameter_certs_hss:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_mme, ssh_keys_build, compile_openair_cn_build,
install_freediameter_hss, create_hss_conf, install_freediameter_certs ]
    properties:
      config:
        get_resource: install_freediameter_certs
      server:
        get_resource: hss_vm
      input_values:
        hostname: { get_param: hss_name }

  deploy_hss:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_hss, ssh_keys_build, compile_openair_cn_build,
install_freediameter_certs_hss, install_database_hss, install_hss ]
    properties:
      config:
        get_resource: install_hss
      server:
        get_resource: hss_vm

### MME DEPLOYMENTS
  create_etc_hosts_mme:
    type: OS::Heat::SoftwareDeployment
    depends_on: etc_hosts
    properties:
      config:
        get_resource: etc_hosts
      server:
        get_resource: mme_vm

  update_system_mme:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_mme, update_system ]
    properties:
      config:
        get_resource: update_system
      server:
        get_resource: mme_vm

  create_mme_conf:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_mme, update_system_mme, mme_conf ]
    properties:
      config:
        get_resource: mme_conf
      server:
        get_resource: mme_vm

  ssh_keys_mme:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_mme, update_system_mme, ssh_keys ]
    properties:
      config:
        get_resource: ssh_keys
      server:
        get_resource: mme_vm

  vpn_client_mme:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_mme, ssh_keys_spgw, vpn_server_spgw, vpn_client ]
    properties:
      config:
        get_resource: vpn_client
      server:
        get_resource: mme_vm
      input_values:
        client_name: { get_param: mme_name }

  install_nettle_mme:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_mme, ssh_keys_build, compile_openair_cn_build,
install_nettle ]
    properties:
      config:
        get_resource: install_nettle
      server:
        get_resource: mme_vm

  install_gnutls_mme:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_mme, ssh_keys_build, compile_openair_cn_build,
install_nettle_mme, install_gnutls ]
    properties:
      config:
        get_resource: install_gnutls
      server:
        get_resource: mme_vm

  install_freediameter_mme:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_mme, ssh_keys_build, compile_openair_cn_build,
install_nettle_mme, install_gnutls_mme, install_freediameter ]
    properties:
      config:
        get_resource: install_freediameter
      server:
        get_resource: mme_vm

  install_freediameter_certs_mme:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_mme, ssh_keys_build, compile_openair_cn_build,
install_freediameter_mme, create_mme_conf, install_freediameter_certs ]
    properties:
      config:
        get_resource: install_freediameter_certs
      server:
        get_resource: mme_vm
      input_values:
        hostname: { get_param: mme_name }

  deploy_mme:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_mme, ssh_keys_build, compile_openair_cn_build,
install_freediameter_mme, install_freediameter_certs_mme, create_mme_conf,
vpn_client_mme, install_mme ]
    properties:
      config:
        get_resource: install_mme
      server:
        get_resource: mme_vm

### SPGW DEPLOYMENTS
  create_etc_hosts_spgw:
    type: OS::Heat::SoftwareDeployment
    depends_on: etc_hosts
    properties:
      config:
        get_resource: etc_hosts
      server:
        get_resource: spgw_vm

  update_system_spgw:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_spgw, update_system ]
    properties:
      config:
        get_resource: update_system
      server:
        get_resource: spgw_vm

  create_spgw_conf:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_spgw, update_system_spgw, spgw_conf ]
    properties:
      config:
        get_resource: spgw_conf
      server:
        get_resource: spgw_vm

  vpn_server_spgw:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_spgw, update_system_spgw, vpn_server ]
    properties:
      config:
        get_resource: vpn_server
      server:
        get_resource: spgw_vm

  ssh_keys_spgw:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ create_etc_hosts_spgw, update_system_spgw, ssh_keys ]
    properties:
      config:
        get_resource: ssh_keys
      server:
        get_resource: spgw_vm

  install_kernel_spgw:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_spgw, ssh_keys_build, compile_kernel_build,
install_kernel ]
    properties:
      config:
        get_resource: install_kernel
      server:
        get_resource: spgw_vm

  install_libgtpnl_spgw:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_spgw, ssh_keys_build, compile_openair_cn_build,
install_kernel_spgw, install_libgtpnl ]
    properties:
      config:
        get_resource: install_libgtpnl
      server:
        get_resource: spgw_vm

  deploy_spgw:
    type: OS::Heat::SoftwareDeployment
    depends_on: [ ssh_keys_spgw, ssh_keys_build, create_spgw_conf,
install_libgtpnl_spgw, vpn_server_spgw, compile_openair_cn_build,
install_kernel_spgw, install_spgw ]
    properties:
      config:
        get_resource: install_spgw
      server:
        get_resource: spgw_vm
```

```
### STACK OUTPUTS
outputs:
  public_ip:
    description: Floating IP address of SPGW instance in external network
    value: { get_attr: [ floating_ip, floating_ip_address ] }
  ssh_spgw:
    description: SSH connect string for SPWG host
    value:
      str_replace:
        template: |
          ssh -o ForwardAgent=yes -o StrictHostKeyChecking=no -o
UserKnownHostsFile=/dev/null -o User=ubuntu $floating_ip
        params:
          $floating_ip : { get_attr: [ floating_ip, floating_ip_address ] }
  ssh_hss:
    description: SSH connect string for HSS host
    value:
      str_replace:
        template: |
          ssh -o ForwardAgent=yes -o StrictHostKeyChecking=no -o
UserKnownHostsFile=/dev/null -o User=ubuntu -o ProxyCommand="ssh -o
ForwardAgent=yes -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null
-o User=ubuntu -q $floating_ip nc -q0 $hss_name 22" $floating_ip
        params:
          $floating_ip : { get_attr: [ floating_ip, floating_ip_address ] }
          $hss_name: { get_param: hss_name }
  ssh_mme:
    description: SSH connect string for MME host
    value:
      str_replace:
        template: |
          ssh -o ForwardAgent=yes -o StrictHostKeyChecking=no -o
UserKnownHostsFile=/dev/null -o User=ubuntu -o ProxyCommand="ssh -o
ForwardAgent=yes -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null
-o User=ubuntu -q $floating_ip nc -q0 $mme_name 22" $floating_ip
        params:
          $floating_ip : { get_attr: [ floating_ip, floating_ip_address ] }
          $mme_name: { get_param: mme_name }
  ssh_build:
    description: SSH connect string for Build host
    value:
      str_replace:
        template: |
          ssh -o ForwardAgent=yes -o StrictHostKeyChecking=no -o
UserKnownHostsFile=/dev/null -o User=ubuntu -o ProxyCommand="ssh -o
ForwardAgent=yes -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null
-o User=ubuntu -q $floating_ip nc -q0 $build_name 22" $floating_ip
        params:
          $floating_ip : { get_attr: [ floating_ip, floating_ip_address ] }
          $build_name: { get_param: build_name }
```