Vicent Ferrer Guasch

LTE network Virtualisation

School of Electrical Engineering

Final Project
Espoo 15.10.2013

Thesis supervisor:

Prof. Jukka Manner

Thesis advisor:

Dr. Jose Costa-Requena

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE FINAL PROJECT

Author: Vicent Ferrer Guasch

Title: LTE network Virtualisation

Date: 15.10.2013 Language: English Number of pages:10+79

Department of Communications and Networking

Professorship: S-38

Supervisor: Prof. Jukka Manner

Advisor: Dr. Jose Costa-Requena

The mobile networks technologies require great investments by the operators but the technology amortization is difficult due to its fast evolution and tight market competition. The virtualisation of the LTE mobile core network has been identified as a solution to share and optimize the available resources. In addition, the virtualisation may open new management and network scalability benefits. This project collects the required open source network elements and implements missing LTE components such as the Mobility Management Entity (MME) required to deploy a complete mobile core network testbed running on virtual servers. The results show the virtualisation of the core network is feasible and complies with latency requirements set for LTE networks.

Keywords: LTE, EPC, SAE, MME, virtualisation, mobile networks, core network, S1AP, NAS, GTPv2, GTP

Preface

This Final Project has been carried out in the Department of Communications and Networking (Comnet) of Aalto University between February 2013 and September 2013 under the supervision of Prof. Jukka Manner and Dr. Jose Costa-Requena.

I would like to start expressing my gratitude to Prof. Jukka Manner and Dr. Jose Costa-Requena to offer me the opportunity to finish my engineering studies on Aalto accepting me as an exchange student. Thanks to Jose Costa-Requena for his attentive supervision an guidance that helped me during this project.

Secondly, I like to thank the staff from my home university Universitat Politècnica de Catalunya (UPC) as well as the staff from Aalto University for their help and kindness in order to handle the required bureaucracy and make my Erasmus program exchange possible.

Finally, I am really thankfully to my family, specially my parents Lluís and Janet, for all their support and comprehension during my student period. I couldn't finish my studies and this thesis without them.

Otaniemi, 15.10.2013

Vicent Ferrer Guasch

Contents

Abstract iii

Preface iv

Contents v

List of Figures vii

List of Tables viii

Abbreviations ix

1 Introduction 1
- 1.1 Background and motivation 1
- 1.2 Scope 1
- 1.3 Thesis structure 2

2 Virtualisation 3
- 2.1 Host Virtualisation 3
- 2.2 Network Virtualisation 5
  - 2.2.1 Network Virtualisation Perspective 5
  - 2.2.2 Network Virtualisation Architecture 6
  - 2.2.3 Virtual Network Classes 7
- 2.3 LTE possibilities 10
- 2.4 Conclusions 10

3 LTE 12
- 3.1 LTE Context 12
- 3.2 LTE Architecture 12
  - 3.2.1 MME 13
  - 3.2.2 eNB 15
  - 3.2.3 S-GW 16
  - 3.2.4 P-GW 17
  - 3.2.5 HSS 17
  - 3.2.6 PCRF 18
- 3.3 EPS bearer layered architecture 18
- 3.4 Protocols 19
  - 3.4.1 S1AP 20
  - 3.4.2 NAS 23
  - 3.4.3 GTPv2 29
  - 3.4.4 GTP-U 31
- 3.5 Procedures of Interest 32
  - 3.5.1 Attach Procedure 33

3.5.2 Detach Procedure . . . 34
3.5.3 X2 Handover . . . 35
3.5.4 S1 Handover . . . 36
3.6 Conclusions . . . 37

4 MME Design and Implementation 38
4.1 Design decisions. . . . 38
4.2 Engine . . . 40
4.3 Interface State Machines singularities . . . 42
4.3.1 S1 State Machine . . . 42
4.3.2 NAS State Machine . . . 42
4.3.3 S11 State Machine . . . 43
4.3.4 S16a State Machine . . . 43
4.4 Database Design . . . 43
4.5 Protocol Libraries . . . 46
4.5.1 GTPv2 . . . 46
4.5.2 S1AP . . . 49
4.5.3 NAS . . . 53
4.6 MMEcmd . . . 57
4.7 Metrics and testing . . . 57
4.8 Conclusions . . . 58

5 Testbed performance results 59
5.1 TestBed Setup . . . 59
5.2 Overhead . . . 60
5.3 Latency on unload conditions . . . 63
5.4 Latency on load conditions . . . 65
5.5 Conclusions . . . 67

6 Conclusions and future work 68
6.1 Conclusions . . . 68
6.2 Future work . . . 68

References 70

Appendix 73

A HSS Database 73
A.1 Database structure . . . 73
A.2 Database definition . . . 74

B SAE-GW patches 77
B.1 GTP-U Length Fix . . . 77
B.2 Ping response CheckSum Fix . . . 78

List of Figures

1 LTE network elements architecture 13
2 S10 Protocol Stack, source TS 23.401 [3] 14
3 S11 Protocol Stack, source TS 23.401 [3] 14
4 S1-MME Protocol Stack, source TS 23.401 [3] 15
5 S6a Protocol Stack, source TS 23.401 [3] 15
6 E-UTRAN Architecture and interfaces. Source: TS 36.401 [12] 16
7 S1-U Protocol Stack, source TS 23.401 [3] 16
8 S5/S8 Protocol Stack, source TS 23.401 [3] 17
9 User Plane protocol Stack, source TS 23.401 [3] 18
10 Bearer service architecture, based on clause 13.1 of TS 36.300 [11] . . 19
11 BASIC-PER aligned encoding of the S1AP header 22
12 Non Access Stratum protocol stack, source TS 23.401 [3] 23
13 EMM state machine on MME, source TS 24.301 [6] 24
14 EMM state machine on UE, source TS 24.301 [6] 25
15 ESM state machine on MME, source TS 24.301 [6] 26
16 ESM state machines on UE, source TS 24.301 [6] 27
17 General message organization example for a plain NAS message, source: clause 9 TS 24.301 [6] 27
18 General message organization example for a security protected NAS message, source: clause 9 TS 24.301 [6] 28
19 Format of the general GTPv2 Header, source: clause 5.1 TS 29.274 [8] 29
20 Format of the TLIV IE structure, source: clause 8.2 TS 29.274 [8] . . 30
21 Format of the GTP-U header, source: clause 5.1 TS 29.281 [9] 31
22 Attach Procedure 33
23 Detach Procedure 35
24 X2 based Handover Procedure 35
25 S1 based Handover Procedure 37
26 MME structure design 39
27 HSS database structure design 44
28 Subscriber Profile entity with attributes 44
29 Operator entity with attributes 44
30 Authentication Vector entity with attributes 45
31 PDN Subscription Context entity with attributes 45
32 TestBed diagram 59
33 Number of Attachments per second 65
34 Number of Attachments per second with 10000 UE preload 66
35 Number of Attachments per second with UE data base reset on 200s . 66
36 HSS database structure design 73

List of Tables

1 Security Header type 28
2 Header Contribution per Protocol on S1-U 60
3 S1-U Overhead on data 61
4 S1-U Overhead with fragmentation 61
5 Attach S1AP overhead 62
6 Attach S11 overhead 62
7 Attach S5 overhead 62
8 Attach Total overhead 62
9 Attach time stamps, expressed on seconds 63
10 Processing time, expressed on milliseconds 64

Abbreviations

AKA	Authentication and Key Agreement
AMBR	Aggregate Maximum Bit Rate
APN	Access Point Name
eNB	E-UTRAN Node B
EMM	EPS Mobility Management
E-RAB	E-UTRAN Radio Access Bearer
ESM	EPS Session Session Management
EPC	Evolved Packet Core
EPS	Evolved Packet System
E-UTRAN	Evolved UTRAN
GUMMEI	Globally Unique MME Identifier
KSI	Key Set Identifier
LTE	Long Term Evolution
MBR	Maximum Bit Rate
MME	Mobility Management Entity
NAS	Non Access Stratum
PDN	Packet Data Network
P-GW	PDN Gateway
PLMN	Public Land Mobile Network
QoS	Quality of Service
S-GW	Serving Gateway
S1AP	S1 Application Protocol
UDP	User Datagram Protocol
UE	User Equipment
UMTS	Universal Mobile Telecommunication System
UTRAN	Universal Terrestrial Radio Access Network

1 Introduction

1.1 Background and motivation

It is beyond doubt that the communication technologies are becoming mobile. The number and type of mobile devices are increasing. Some studies state that the number of mobile devices will be comparable with the total world population by the end of this year [37]. This increase is caused by the adoption of video related services in mobile devices. These services require high capacity and low latency which pose new requirements to the mobile networks.

In order to supply this increasing capacity demand on LTE, a cell size reduction is proposed to deploy a large number of base stations with backhaul high connectivity. The deployment of this small cell scenario increases the network capacity in the access network which is translated into a big challenge on current mobile transport networks due to its rigidity and complexity.

Current implementations of mobile core networks rely on specialized network equipment. This equipment is expensive, complex to manage and nearly impossible to modify. The systems based on specialized hardware makes more difficult the return of investment as they become obsolete on shorter periods of time due to the reduction of the hardware life cycle. A long life cycle on these systems hinders the network evolution.

A new cost effective solution is required by network operators in order to satisfy the increase of subscriber transport capacity demands. It is expected that virtualisation of the LTE network elements will deliver the required solution. Moreover, a testbed is deployed to prove the virtualisation of LTE will bring the expected benefits while still complying with the 4G latency requirements. The virtualization of LTE network nodes allows the transfer of the functionality to data centers to run on commodity servers, thus improving scalability and efficiency of LTE network elements. It is foreseen that a software based solution have the potential of speeding up the mobile network evolution.

1.2 Scope

The goal of this thesis is to build an EPC testbed based on existing open source LTE network elements running in a data center to study the feasibility of EPC virtualisation. To build the EPC testbed only open source implementations have been considered because of the advantages of having the source code available and the possibility to modify it. There are some open source projects implementing mobile network nodes, but the available open source projects including LTE nodes are very limited.

The S-GW and P-GW nodes are available as an open source project [41], but no open source implementation of the MME was available. This thesis designs and implements the MME node logic, including the different protocol stacks, the required interfaces and the MME state machines. The objective of implementing the full MME is to measure its performance as a virtualised node and provide the source code for its future extension with new functionality such as SDN controller.

The results obtained from the testbed show that is possible to virtualise the EPC nodes on a data center and still comply the signalling latency requirements included on the 3GPP standards [18].

1.3 Thesis structure

Chapter 2 provides the definition, background and types of virtualisation, the chapter ends with possible features to take into account in order to deploy LTE using a virtualisation environment. Chapter 3 describes LTE and summarizes technical details included in the standards definitions This chapter is intended as a state of the art analysis. Chapter 4 exposes the design and implementation strategies used on the MME development. Chapter 5 includes a description of the testbed built with the implemented MME plus other open source LTE network elements. This chapter also provides some measurement results in order to study the testbed performance and compliance with the standards. Finally, chapter 6 provides final conclusions and comments on future work.

2 Virtualisation

The term "Virtualisation" is used to describe the creation of virtual resources by the means of software. Virtualisation provides the required abstractions on a host environment to emulate a required resource. Virtualisation can emulate resources of different kind, some of them are hardware platforms, operative systems, storage devices and network resources.

This chapter introduces host virtualisation in Section 2.1, Section 2.2 describes network virtualisation, Section 2.3 discusses LTE virtualisation and finally the conclusions are in Section 2.4.

2.1 Host Virtualisation

The Virtualisation concept was introduced for the first time in the 60's by IBM while developing time-sharing systems for mainframes [26], [27]. On the 50's computer mainframes used to run batch jobs with punch cards, these jobs ran on sequence. Time sharing concepts were introduced due to the need for scientific computing to interact with the computer.

The first aim of virtualisation was to optimize the hardware resources for multiuser usage, it was a method to logically divide the available resources.

The mainframe IBM System/360 was designed to allow backward compatibility with older IBM systems. At that time, that was a notable innovation because the migration to a new system supposed a large cost for the companies due to the need for rewriting programs and change all the printers, card readers, tape drives, etc. The IBM System/360 was the IBM proposal to the MAC project.

The announcement of the MAC project by MIT opened a competition between IBM and General Electrics (GE) to satisfy the computer hardware requirements. Time sharing was not seen as important in IBM and the focus was on virtual memory. The loss of the project by IBM provoked repercussions on IBM and they change their strategy.

The first product of the new strategy was CP-40. CP-40 became the first virtual machine OS in a fully virtualised hardware on an IBM System/360 but it had a limited distribution because it was an internal solution. The architecture evolved to CP/CMS, to separate the resource management and the user support. CMS stated for "Cambridge Monitor System" and was responsible for virtual memory time-sharing. CP-67 was launched in 1968 as the evolution of CP-40. These were essentially research systems. The evolution continued to support MVS and UNIX.

In 1985, Intel 80386 microprocessor included a virtual machine monitor "Simultask", by Locus Computing Corporation and AT&T, that enabled the execution of a virtual i8086 guest OS on a UNIX System V OS host.

The hardware virtualisation evolution continued with SoftPC, VirtualPC, VMWare, Xen and VirtualBox. However, the virtualisation on the application level must also be noted. Solutions like Cygwin to run Linux applications on Windows or Wine to run Windows applications on Linux virtualise the system libraries of the original OS. Java has to be mentioned as a special case of application level virtualisation. Java was created by Sun Microsystems to enable applications to be executed on any OS using a run-time environment, Java Virtual Machine (JVM). The Java application is compiled to a platform independent bytecode binary that can be executed on the JVM.

The software or firmware to create a virtual resource on a host system is called hypervisor. Virtualisation is intensively used on current data centers. It is implemented with software, however, hardware vendors include virtualisation optimization in their products.

The new tendencies explore the expansion of the virtualisation concept to new levels. The most popular are the “Cloud” concept [28], Network Virtualisation [29] and Network Functions Virtualisation [30].

The most common virtualisation nowadays is hardware virtualisation, a virtual machine capable of running an unmodified operative system (OS). This allows in practice to have copies of real computers running at the same time on one physical computer.

By means of virtualisation appeared other new concepts as Snapshotting or Teleportation. A snapshot stores the state of a virtual machine at an exact point of time. The snapshot feature allows to backup the virtual machine before a risky operation. Teleportation or migration uses the snapshot concept to restore a previous virtual machine snapshot on a different host, thus a virtual machine is temporary stopped, snapshotted, moved and resumed on a new host on its own hypervisor. When the snapshots are synchronized on different hosts, they are able to provide a continuous service when the source host is taken down.

The hardware virtualisation is divided into tree types depending on the virtualisation degree and how the host hardware is accessed. These types are detailed as follows.

- **Full virtualisation** provides hardware emulation to allow a guest operating system to run unmodified.
- **Partial virtualisation** some hardware simulation is provided but not all, some guest programs may need modifications to run on such environment.
- **Paravirtualisation** the hardware is not emulated, however the applications are executed on isolated environments as they were running on different systems. The applications running on a paravirtualised system require special modifications.

Paravirtualisation is the most efficient virtualisation type as the hardware is directly accessed although it is more complex to develop applications for it.

On a full virtualised environment, the application development is simpler due to the provided abstractions although these abstractions reduce the efficiency of the resources. On the other side paravirtualisation is the virtualisation type nearer to the real host thus the hardware is accessed quicker but the management and the software development on this framework is more complex.

2.2 Network Virtualisation

This section introduces the state of the art of network virtualisation, based on paper [31]. Section 2.2.1 introduces Network Virtualisation and explain the design requirements identified. Section 2.2.2 exposes the network virtualisation architecture and its requirements. Section 2.2.3 include the different implementations of network virtualisation.

2.2.1 Network Virtualisation Perspective

Virtual Networks (VN) are defined as the concurrent use of physical networks for multiple variants or instances of networks, slicing the physical network. Each slice or network instance is a set of isolated resources for different purposes on the same shared physical infrastructure.

Currently, Internet fundamental innovation is limited by the deployment resistance of the Service providers. It is referred to as Internet ossification. The decentralized structure on multiple Internet Service Providers (ISP) require coordination for a wide-scale deployment of new services. The required coordination is not achieved due to the little benefits the ISP obtain until the service is deployed on other domains. The problem is that ISP doesn't control the entire end to end path.

Virtual Networks deployment on wide-scale is foreseen unsuccessful due to the commented Internet ossification. In order to address this problem a new business architecture has been proposed. A decouple of the ISP role is required into two different entities, the Infrastructure Provider (InP) and the Service Provider (SP). The InPs manage and deploy the physical network and offer their networks to different SPs by means of programmable interfaces. The InPs compete with others with the quality of their resources, and the programmable options they offer. The SPs lease the resources from different InPs to offer an end to end service to the final users. The network resources of different InPs are aggregated to create Virtual Networks. These VNs suppose an abstraction to hide current heterogeneous network architectures that limit Internet. The SPs are responsible for the protocols used on their VNs to offer the end to end service. The Virtual Network concept may become a new networking paradigm.

Following are the identified design criteria of a virtual network:

Flexibility Virtual networks should be flexible on every networking aspect. The Virtual network manager should be allowed to deploy any topology without any physical restrictions, provide any standard or custom protocol and routing, forwarding functions. In addition, the manager of the virtual network should have that freedom without the need to coordinate with any other.

Manageability The separation between InPs and SPs modularize management, allowing a complete end-to-end control of the VN or accounting in the different layers. The management should be independent of the InPs to avoid the need of coordination of different InPs to establish an end-to-end service.

Scalability The scalability is a basic requirement in order to deploy multiple VN coexisting on the same infrastructure. The Virtualization technology must be scalable to be able to support the coexistence of VN without affecting their performance.

Isolation In order to deploy a fault tolerant environment and provide security and privacy, the VN should be isolated by the virtualization technology. A network configuration error on a VN should not affect other coexistent VN nor the virtualization environment.

Stability and Convergence Related with the Isolation requirement, in the case of a network virtualization environment error, all the coexistence VN can be affected. The virtualisation design must ensure the stability of the network virtualisation environment and in case of an error force the convergence of the VN to their stable states.

Programmability is a basic requirement to ensure flexibility and manageability. Network elements programmability is required to support custom protocols and deploy new services. Different levels of programmability can be considered during the design of the network virtualisation environment. There is a trade-off between the security and the programmability of the network resources.

Heterogeneity may be present on different parts of the network virtualisation. The underlying network may be composed of different heterogeneous networks, thus the VN provides a uniform abstraction to work on. Heterogeneity may also be present on the different services, protocols and algorithms deployed on the VN.

Legacy Support or backwards compatibility is a great concern in order to facilitate the deployment of the designed virtualisation environment. It is common to considerate the existing internet as another VN, but the efficiency or the manner it can be done is an open challenge.

2.2.2 Network Virtualisation Architecture

A VN is a subset of the physical network supporting it. VN is built with a collection of virtual nodes connected using virtual links to deploy the desired topology. A physical node may contain multiple virtual nodes. A virtual node may be either a virtual host to represent the end point of a connection or a virtual switch to manage the routing path. Virtual links use the network paths to connect the virtual nodes on different physical nodes, but they may also connect two virtual nodes on the same physical node.

The described virtual architecture forces the following requirements:

Coexistence A physical network should be able to support multiple VN simultaneously. The different VN supported on the physical network share the available resources without interfering with each other. A VN manager or a SP should be able to configure the VN without interfering with other VNs, even in the case of VN failure or configuration error.

Recursion The recursion requirement refers to the possibility of spawning a new VN on top of the existing one, creating a parent-child hierarchy between the VNs. It is also known as VN nesting.

Inheritance The inheritance requirement is related with recursion. A child VN inherits the architecture characteristics and constraints of the parent VN. This requirement allow a SP to add value to the VN before reselling it to other SPs.

Revisitation The revisitation requirement represents the possibility hosting multiple virtual nodes on a single physical node. The VN is decoupled from the physical network. Revisitation allow a SP to rearrange the VN topology without limits to simplify the management. In addition, revisitation allow the creation of VN on a single physical host.

2.2.3 Virtual Network Classes

The concept of multiple coexisting logical networks is common on networking literature [31] and can be categorized on the four following classes:

VLAN refers to Virtual Local Area Network. On layer 2, bridges form broadcast domains, dividing the network. VLAN is a mechanism to expand or limit these broadcast domains and decouple them from the physical location. A VLAN groups physically distributed hosts on the same broadcast domain. VLAN is used to simplify the network design grouping hosts with a common set of requirements. A VLAN has the same characteristics as a normal LAN but the logical nature improve the scalability, security and management issues present on normal LANs. VLAN is defined in the standard IEEE 802.1Q [25].

The first implementations of VLAN were using switches with port level partitioning support, known as port switching. This first approach required dedicated cables for each VLAN. This type of VLAN is a layer- 1 VLAN. The withdraw of that approach is that the mobility is not allowed.

The dedicated cables were substituted using tags on the packets, thus multiple VLANs can be served using one single physical cable. The VLAN tag is included in the MAC header of the network frames by the first switch supporting VLAN and the last switch remove the tag. The switch acts as a different switch for each VLAN, hiding the traffic of a VLAN to the rest of the VLANs, isolating the VLAN. The VLAN is defined with the list of MAC address. This type of VLAN is known as layer-2 VLAN. This approach provides full mobility for the users but the large number of MAC addresses may be difficult to manage.

In order to simplify the MAC address management, the layer-3 VLAN is defined. The VLAN membership is defined using the MAC protocol type field and the IP subnet. The VLAN is defined for packets instead of hosts. The layer-3 VLAN simplifies the configuration because the configuration is learned by the switches. Other high layer tagging exist depending of the transport protocol or even the application.

Virtual Private Network is also known as VPN. A virtual private network is a mechanism to extend a private network in public or shared networks as a WAN or Internet. The main principle is to connect multiple geographically distributed networks using tunnels. It is a point-to-point connection. VPN is widely used by companies to connect different corporate sites or to grant remote access to their employers of local resources. To prevent the disclosure of private information, VPN offers encryption and authentication.

In the edge of each VPN site there is a Costumer Edge (CE) device controlled by the corporation and a Provider Edge (PE) device managed by the service provider. These devices act as VPN tunnel endpoints depending on the VPN type.

A VPN can be classified on different classes depending on the protocols used, the tunnel termination point location, the level of security, the OSI layer used on the connection or whether they provide remote access or site-to-site connection.

Layer-2 VPNs (L2VPN) transport level 2 frames, usually Ethernet. L2VPN are more flexible than VPN based on higher layers because there is no restriction on layer 3 level. Layer-3 VPN transport layer 3 protocols and can be divided on two classes depending whether the VPN is managed by the CE or by the PE. If the traffic is tunneled on the CE, the service provider is unaware of VPN existence. When the service provider manages the VPN, the tunnel encapsulation is performed on the PE and the CE acts as it is on a private network.

VPNs on higher layers are popular because their lightweight, easy install and usage. On high layer VPNs, the remote location connection is affected by NAT and Firewalls, therefore the configuration offers higher granularity.

The VPN concept was introduced in RFC 4026 [23] and the terms generalized in RFC 2547 [22].

Programmable Networks have a long research history and deployment attempts [33]. Network Virtualisation and the Programmable Networks are two different concepts but their tight relationship has been considered in order to include the Programmable Networks in the current section. Network Virtualisation is one of the possible use cases of Programmable Networks.

During the mid 90's, the frustration of many researches on the slow standardization process of new protocols and the impossibility to test and deploy new ideas for improving network services led them to the search of an alternative approach to open the network control. The inspiration came from an analogy to the reprogramming possibilities available on a single PC. The proposed solution was the Active Network.

Active Networks proposed a programming interface with an API to expose network resources available on individual network nodes. The programming interface should also support the definition of new custom functionality to process a subset of packets passing through the node. The messages of such a network would include the required code to implement that custom functionality in addition to the usual data. The code inclusion on the messages was discriminated on two different models: 1) the capsule model with the code encapsulated in-band with the messages, 2) the programmable switch/route where the code was sent using out-band mechanisms.

Although interesting ideas involved, Active networks were not deployed on a wide scale. The adoption of Active networks was stopped by the lack of a deployment plan and a pressing need.

Next attempts try to solve smaller problems focusing on routing and configuration management. The following technologies focus was to differentiate the control plane and the data plane. In early 2000's, the network operators sought better approaches to certain network management function as path control due to the increase of traffic volumes. The innovations were the differentiation of the control plane and data plane in addition to a logical centralized control of the network. These innovations affected the network administrators. The control functionality was moved from the network equipment to separate servers with centralized controllers. The logical centralized controllers in addition to open source routing software lowered the barrier to create prototype implementations.

On the mid-2000, Software Defined Network (SDN) ideas emerged between the Active Network and the pragmatism needed in order to deploy the technology in the real world. In this context, one of the implementations, OpenFlow, gained popularity (2011). Openflow enabled more functionality than the early route controllers using commodity network equipment. An OpenFlow switch contains packet handling rules on different tables. The packets are matched with a defined bit pattern and the action associated is applied. In addition, the switch uses a set of counters to track the number of packets and bytes and a set of priorities in case the packet matches multiple rules.

The physical centralization of the SDN controller must be avoided in order to increase scalability, reliability and performance, preventing a network failure due to a controller error. The controller distribution introduces new challenges to maintain a consistent network state. The physical distributed controllers should act as a single, logically centralized controller.

Another concept related with SDN is the Network Operating System. The network operating system software abstracts the network installation of state in network switches to hide the applications and logic to control the behavior of the network.

Last tendencies in programmable networks, Network Functions Virtualisation [30], show a return to the Active Networks early work.

Overlay Networks are logical networks deployed on top of an existing physical network. Usually overlay networks are implemented on application level although implementations on lower layers also exist. The most common implementation design is using tunnels to simulate the virtual links. Overlay networks do not require changes on the lower layers in order to be deployed thus they have been used to build inexpensive new network functionality and fixes.

The issues to be addressed with overlay networks include: performance insurance, availability of internet routing, enabling internet multicasting, providing QoS guaranties, denial of service attacks protection and network isolation. However, the overlay networks inherit the restrictions of the physical network supporting it.

Overlay networks are widely used to deploy research testbeds to try new technologies. An example of an overlay network is X-BONE [34].

2.3 LTE possibilities

The most attractive gain of virtualising EPC is to deploy the core network functionality on commodity servers, avoiding specialized hardware. EPC nodes can be bundled on a single data center or split in different locations, resulting on the reduction of CAPEX during LTE deployment.

Hypervisor functionality and virtual network controller could allow an improved monitoring and operation of EPC nodes and its connections, allowing the allocation of more resources during a peak of demand, including routing devices. The monitoring system should trigger the corresponding alarm when a potential problem is detected. Both reactive and proactive strategies can be used in new virtualisation technologies to solve punctual problems. In addition, the network programmability may suppose a reduction of OPEX, simplifying the operation of the network due to a logical centralized controller.

The first developed virtual EPC nodes may be virtualised on commodity servers using cloud software such as OpenStack [49] using all virtualisation type. This type simplifies the development of the network element software implementing the functionality of one of the required EPC nodes. On all virtualisation approaches the software does not require any modification. However, all virtualised approach may not be efficient enough, in such case the hardware access to the host resources may be changed to allow a more direct access. The price to pay for speed is the increase of the complexity of the guest application until paravirtualisation type is reached. Whitepaper in [36] presents the results of a LTE deployment on a paravirtualised environment using specialized virtualisation hardware while managing power management.

Current network virtualisation usually does not consider mobile networks. Most of the network virtualisation implementations are not prepared to manage mobility, but the flexibility requirements may be wide enough to support it. Further research on network virtualisation could trigger the disruption of new mobile possibilities.

2.4 Conclusions

This chapter has exposed the virtualisation as the technology that provides software abstractions to emulate a resource. Both hardware and network virtualisation are potential disruptive technologies to be applied on mobile networks. Hardware virtualisation is mature enough to start LTE node virtualisation testing, but network virtualisation needs more research and implementation efforts in order to integrate it with LTE hardware virtualisation and deploy a fully virtualised mobile core network.

The use of software resource abstractions provides new functionality impossible in a physical framework such as snapshots and teleportation. These new functions and the environment provide more flexibility on a LTE setup.

General design and architecture requirements of network virtualisation are included and different technologies are discussed. The logical network controller feature and the centralized network view are promising features in order to simplify the operation and management of the network.

An improved VLAN technology merged with programmable networks may offer the mobility requirements to deploy a mobile network. The final solution should involve lower layers to optimize the network and provide enough flexibility. In addition, LTE network requires a granular network control to fine tune it up to the application level configuration.

3 LTE

This chapter provides an overview of LTE standards to build a functional LTE core network. Section 3.1 provides an LTE definition. Section 3.2 describes the LTE architecture and defines its network elements. Section 3.3 introduces the LTE bearer concepts used to discriminate and manage the user traffic. Section 3.4 details the main LTE protocols used on the LTE core network. Finally, Section 3.5 explains and depicts the main procedures implemented on the LTE testbed using network flow diagrams.

3.1 LTE Context

With the introduction of the smart devices into the market, the data transport requirements have increased on the mobile networks. Multimedia services are being moved to mobile devices such as smartphones or tablets. These requirements demand an improvement of broadband technologies and Long Term Evolution (LTE) is one of the proposed standards.

LTE is the mobile networks standard proposition of 3rd Generation Partnership Project (3GPP) group as an evolution of Universal Mobile Telecommunications System (UMTS). These standards are included in the 4th Generation (4G) of the mobile networks. There are other 4G standards and standardization entities, such as WiMAX, but LTE and 3GPP are the most accepted.

The LTE standards on its Release 10, called LTE-A (LTE Advanced), have passed the ITU-R requirements IMT-A (International Mobile Technology Advanced) for what is marked as 4G. The nominal data rate of these specifications is 100Mbit/s with high mobility clients and 1Gbit/s with static clients.

As required by IMT-A, LTE is based on a packet switched network over IP on all the nodes, from the User Equipment (UE) to the Packet Data Network (PDN). This characteristic simplifies the architecture and makes it more economic and scalable. Although LTE is compatible with older 3GPP technologies, allowing the fallback to circuit switched mode. This feature proposes LTE as the natural evolution of current mobile networks.

3.2 LTE Architecture

The LTE architecture can be divided on Evolved Universal Terrestrial Radio Access Network (E-UTRAN) and Evolved Packet Core (EPC). The whole system is called Evolved packet system (EPS) and the non-Radio aspects are identified under the term “System Architecture Evolution” (SAE).

In Figure 1, we can observe a simple LTE non-roaming architecture. The standards define the interfaces between the nodes and the protocols used to exchange the correct information between them. The implementation of the node functionality is specific for each vendor which relies on the standards to ensure compatibility.

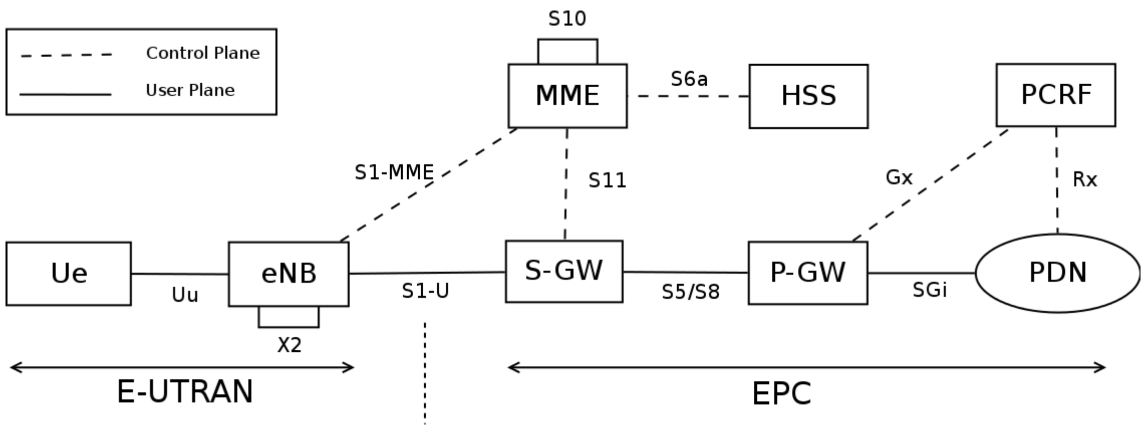


Figure 1: LTE network elements architecture

In addition, the network architecture between the nodes can be designed with the requirements of the network operators.

A major change on E-UTRAN is the increase of the complexity of the eNodeB. The new eNodeBs are capable of exchanging information between them to control the UE mobility compared with the old NodeB that required a radio network controller (RNC) for mobility management.

The traffic can be divided on two planes, user plane and control plane. The user plane transports the user data over IP protocol using a “virtual” connection called EPS bearers explained in 3.3. The management of the EPS bearer is performed on the control plane with Session Management procedures.

The following sections explain the network elements shown in Figure 1. For more details see TS 23.002 [4].

3.2.1 MME

The Mobility Management entity is the main control plane node. It processes the signalling between the Radio Access Network (RAN) and the EPC to manage the available resources. The MME main functionality is the session management. It manages the EPS bearers and the mobility processes to establish the required connections. Other MME responsibilities are the security and authentication.

The main LTE interfaces of the MME node are S10, S11, S1-MME and S6a. These interfaces are explained on the following section. There are other interfaces related with the connection to non E-UTRA networks but are beyond of the scope of this document which focuses uniquely on LTE network.

S10: This interface manages the connection between different MME nodes. S10 is intended to allow the MME relocation when the UE moves out from one area controlled by the origin MME to the new destination MME’s area. The S10 interface supports the transfer of the user information. The current implementation of the MME does not consider the development of the S10 interface to transfer user information to new MME but instead additional resources will be requested from

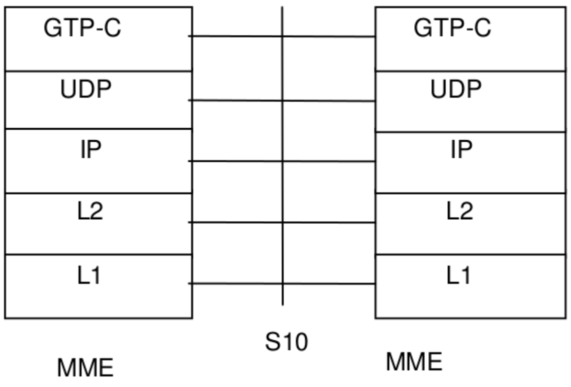


Figure 2: S10 Protocol Stack, source TS 23.401 [3]

the data center where MME is running. However, when moving MME information to different data center the S10 interface might be required to allow the relocation of subscriber information to a new virtually allocated MME. This interface uses the GTP-C protocol specified in TS 29.274 [8].

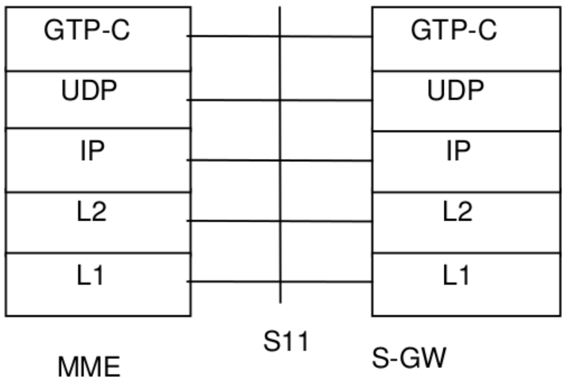


Figure 3: S11 Protocol Stack, source TS 23.401 [3]

S11: This interface connects the MME with the S-GW, to allocate new network resources and route the EPS bearers on the data plane. The MME has no direct connection with the P-GW, but it can configure the different data plane routes through the S11 interface. This interface uses the GTP-C protocol as in the S10 interface specified in TS 29.274 [8].

S1-MME: It is the control interface between the MME and the eNB. S1-MME is the signalling interface between the E-UTRAN and the EPC. This interface controls the connection between MME and eNB and uses the S1AP protocol specified in TS 36.413 [16]. The S1AP protocol uses the Non Access Stratum protocols (NAS) specified in TS 24.301 [6] between the UE and the MME.

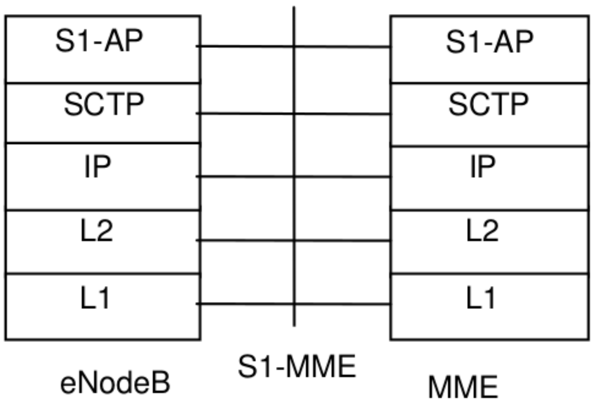


Figure 4: S1-MME Protocol Stack, source TS 23.401 [3]

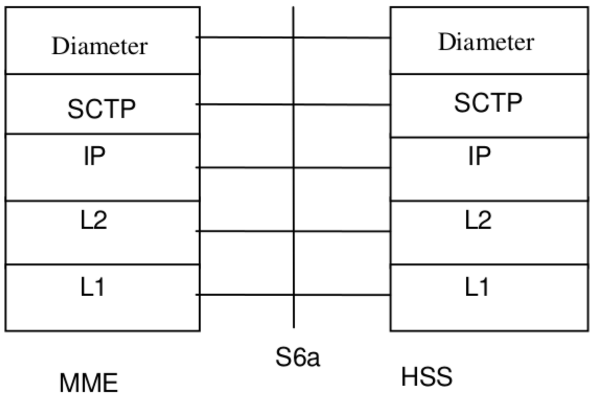


Figure 5: S6a Protocol Stack, source TS 23.401 [3]

S6a: This is the reference point between the MME and the Home Subscriber Server (HSS). The interface is used to exchange the UE location information and the subscriber information. The MME sends the UE location information to the HSS and the HSS sends to the MME the subscriber information needed to support the services that the UE requires from the network. The data exchange is done using the Diameter S6a/S6d Application as specified in TS 29.272 [7] every time a new service is requested by the UE or a UE mobility procedure is processed to refresh the UE location.

3.2.2 eNB

The E-UTRAN Node B (eNB) is the network access element for E-UTRA. It provides the UE data and control plane terminations. The eNBs are grouped on a network to form the E-UTRAN using the X2 interface. The E-UTRAN structure has no centralized node, as opposite to the previous access architectures, it is a flat structure. The eNB is connected to the EPC with the S1-MME interfaces as explained on the MME section 3.2.1).

The eNB manages all the related radio functions to allow the access of the UE

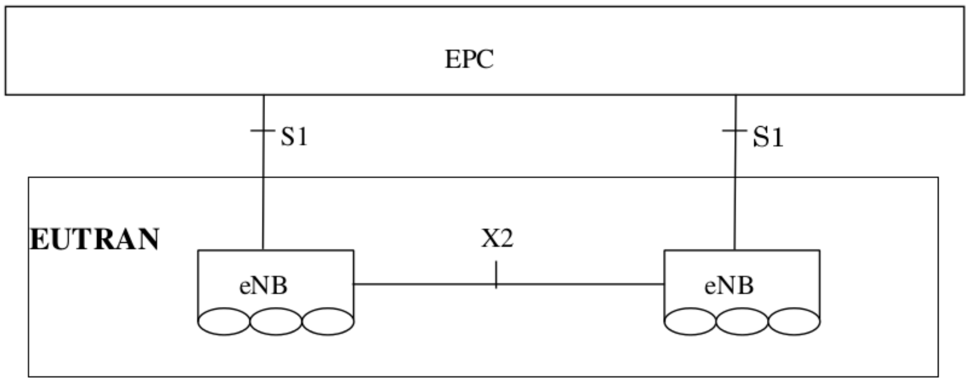


Figure 6: E-UTRAN Architecture and interfaces. Source: TS 36.401 [12]

to the EPC network. The protocols used between the eNB and the UE are known as “Access Stratum” (AS) protocols.

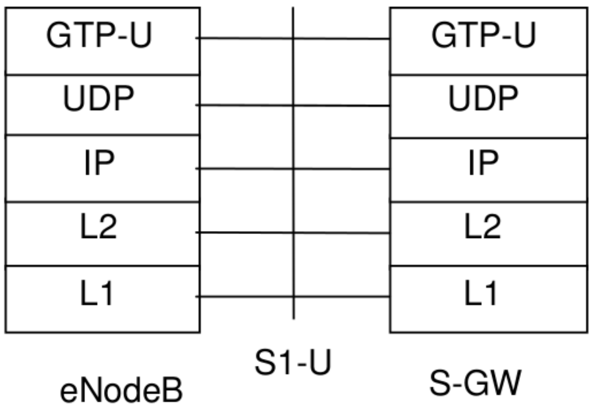


Figure 7: S1-U Protocol Stack, source TS 23.401 [3]

S1-U The reference point between the eNB and the EPC is the S1-U interface. This interface is on the data plane and is the endpoint of the tunnels used to transport all the UE data. The tunnel protocol used is the GPRS Tunneling Protocol User Plane (GTP-U) specified in the TS 29.281 [9].

3.2.3 S-GW

The Serving Gateway (S-GW) is the EPC termination to the E-UTRAN. There is a single S-GW for each associated UE on the EPS at a given time. The main function of this node is to route and forward the user data to the selected P-GW’s.

Other important functions of the Serving GW include Mobility anchoring, lawful Interception, accounting on user and QCI granularity for inter-operator charging, reporting to the PCRF, etc.

The interfaces of the S-GW are the S1-U, the S11 and the S5/S8.

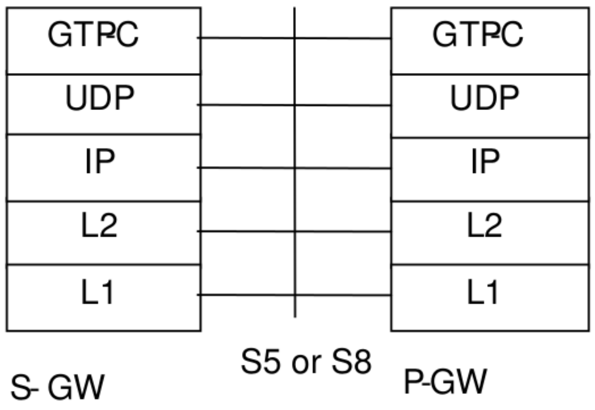


Figure 8: S5/S8 Protocol Stack, source TS 23.401 [3]

S5/S8 This interface is located between the S-GW and the P-GW for packet data services. The S8 is the inter PLMN variant of S5. The protocols used are the GTP-C and GTP-U for the control plane and the user plane. The Proxy Mobile IP (PMIP) is another alternative to these protocols that can be used in this interface. The PMIP is used by operators to simplify the inter-working with WiMAX/CDMA2000.

3.2.4 P-GW

The Packet Data Network Gateway (P-GW) is the LTE termination to the Packet Data Network (PDN). A UE can access multiple P-GW if it is accessing more than one PDN on the same PLMN. The P-GW main function is to decapsulate the UE data of the GTP tunnel and forward UE data to the PDN. The P-GW will also allocate the IP address of the UE. The P-GW has other functions as per-user packet filtering, deep packet inspection, lawful interception, Uplink (UL) and Downlink (DL) rate enforcement by a rate policing / shaping, etc. The S5/S8, the Gx, and SGi are the P-GW interfaces.

Gx is the reference point between the P-GW and the Policy and Charging Enforcement Function (PCRF). The Gx Interface provides policy and charging rules for the P-GW.

SGi is the interface between the P-GW and the Packet data network. The PDN may be an operator external public PDN, private PDN or an intra operator PDN, i.e. for IMS services. The UE IP address is exposed on this interface.

3.2.5 HSS

The Home Subscriber Server (HSS) is the main database containing the user information on the Home Network.

Usually, a single server is required on a Home network, but the HSS can be split on multiple nodes due to the organization of the network by the operator or to the

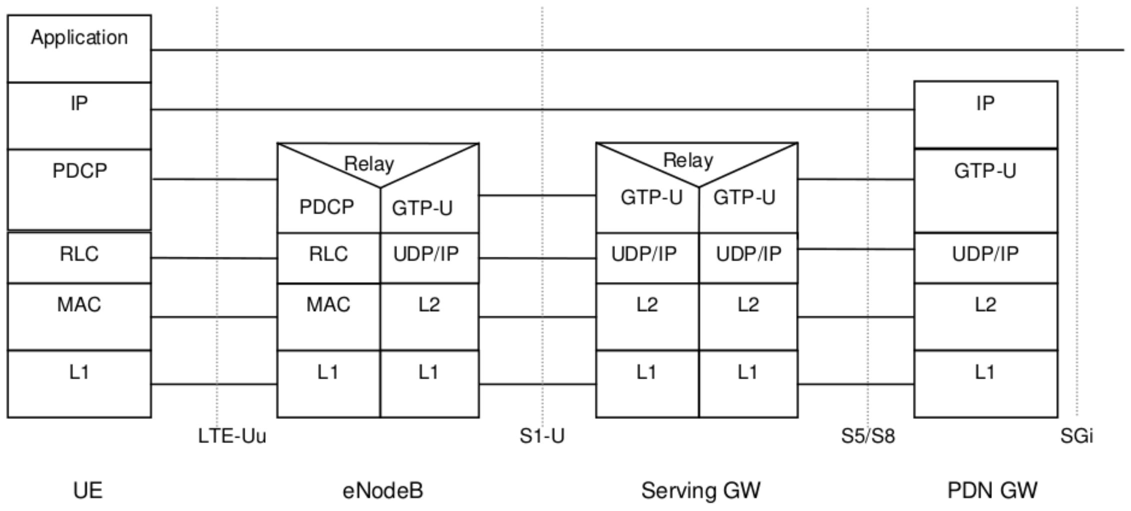


Figure 9: User Plane protocol Stack, source TS 23.401 [3]

number of subscribers that can limit the node performance.

The HSS stores the UE location information used to find the UE when needed by applications, i.e. when UE has to receive a call. The HSS also stores the subscriber information needed to support the services to the UE on the network. Subscriber information contains the user identifications, the allowed bit rates and QoS, etc. In addition, the HSS generates the security information required for the mutual authentication including the ciphering and integrity keys.

S6a is the interface considered between the HSS and the MME, explained in MME section (3.2.1).

3.2.6 PCRF

Policy and Charging Rules Function (PCRF) is the policy and charging control element of the service data flows and IP bearer resources.

PCRF functions are described in more detail in TS 23.203 [2].

The interfaces of the PCRF are the Rx and the Gx. The Gx interface has been explained before in P-GW section (3.2.4).

Rx the interface between the PCRF and the operators IP services, i.e. IMS.

3.3 EPS bearer layered architecture

The EPS Bearer concept is considered as a connection-oriented transmission between the User equipment (UE) and the Packet Data Network (PDN) endpoints. This EPS Bearer is tunneled from the UE through the switching nodes up to the PDN. Multiple EPS bearers can be established for a single UE. The key concept of the EPS is the Quality of service parameter. Each bearer provides different QoS for the same or different PDN and each QoS can be assigned to a different service. The

EPS bearers can be characterized by two endpoints, a QoS Class Identify (QCI), the granted or maximum bit rate (GBR, MBR) and a traffic flow filter.

When a UE requests the access to a PDN, a default EPS bearer is established by the network to access the default PDN. This bearer is intended to be an always on connection that will be available all the time during the UE connection. The UE can request additional EPS bearers that will be established and those will be dedicated Bearers.

Another classification of the EPS bearers is the Granted Bit Rate (GBR) EPS bearers and the non-GBR bearers. The GBR bearers provide a minimum bit rate that allows special applications such as VoIP the required bandwith. Higher bit rates are allowed but they can be limited using the Maximum bit rate parameter (MBR). The non-GBR bearers allow a non real-time, best effort, delay tolerant services. The default EPS bearer does not have any granted bit rate, thus is a non-GBR bearer. The dedicated EPS bearers can be either GBR or non-GBR bearers.

Figure 10 shows EPS bearer service layered architecture to depict the relationship between the different bearers present on the EPS. The Radio data Bearer transports the EPS Bearer packets from the UE to the eNB. The Evolved Radio Access Bearer (E-RAB) is used to transport the EPS Bearer packets from the UE to the EPC. There is a one-to-one mapping between the Radio Bearer and the E-RAB/EPS bearer and between the E-RAB and the EPS bearer.

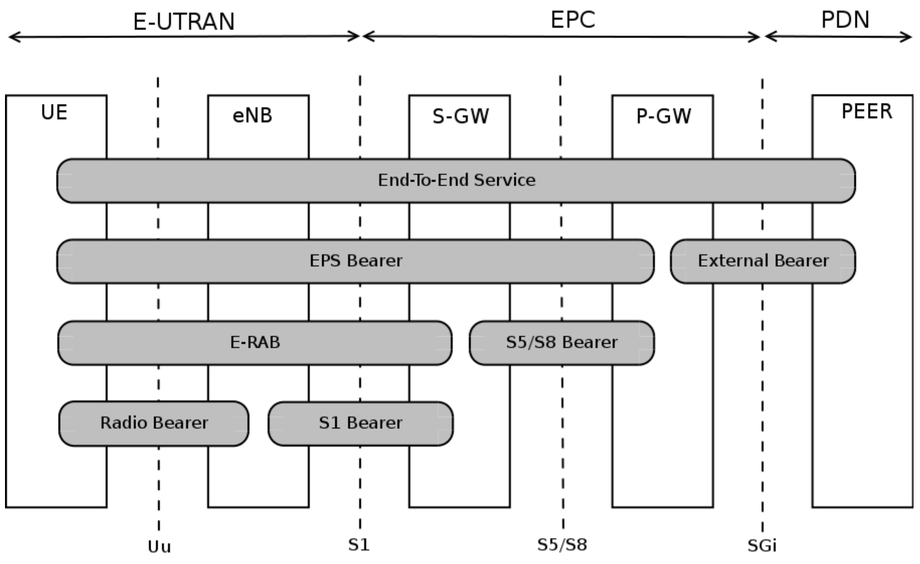


Figure 10: Bearer service architecture, based on clause 13.1 of TS 36.300 [11]

3.4 Protocols

This section includes the general characteristics and functions as summary of the main protocols used on the EPC. A detailed description of the protocols is included in the 3GPP specifications.

Following the definition of some common terminology is included.

Elementary Procedure (EP) consists of an initiating message and it might also include a response. EP is the basic element of the protocols. Therefore, protocols, are defined as a set of elementary procedures.

There are three types of Elementary Procedures:

- Class 1: Elementary procedures with response, either successful or unsuccessful.
- Class 2: Elementary procedures without response.
- Class 3: Elementary procedures with multiple responses reporting either successful or unsuccessful outcome.

Message The data package send from one node to another in order to transfer some information. The message is usually composed by a header and the Information Elements of the protocol.

The messages shown on this section are sent using network octet order sending first the bit 8, thus the most significant one.

Information Element (IE) is the simplest encapsulation unit of the information included in a message payload, which usually includes a set of IE's. The IE structure and encoding vary depending on the protocol and IE type. A typical structure is a TLV structure, type, length and value. The Value of an IE is the useful part of the transported information.

3.4.1 S1AP

S1 Application Protocol (S1AP) is the signalling protocol used on the S1-MME interface. The protocol is defined on TS 36.413 [16] using ASN.1 [19] [20] and encoded according to the Basic Packed Encoding Rules (BASIC-PER) Aligned Variant specified in ITU-T Rec. X.691 [21].

Transport The transport protocol of S1AP is the Stream Control Transmission Protocol (SCTP) specified on the RFC 4960 [24]. The Payload Protocol Identifier (**PPI**) assigned by IANA to be used by SCTP for S1AP is **18**. A single SCTP association is allowed per S1-MME interface and it is established by the eNB. The SCTP destination **port** for S1AP is the **36412**.

SCTP can manage multiple streams on a single association. These SCTP streams are unidirectional and the maximum number of streams is negotiated during the association process. S1AP reserves a pair of SCTP streams for non UE-associated procedures and at least another pair of streams for the UE-associated signalling. Although a few additional pairs of streams can be used for UE-associated signalling added on top of the mandatory pair. (see clause 19.2 TS 36.300 [11] and clause 7 of TS 36.412 [15]).

Following are described the most important functions of the S1AP protocol:

- S1 UE context management functions to establish, modify or release a UE context on the S1 interface.
- S1 interface management functions as error or overload indications, S1 Setup function, load balancing or configuration updates.
- Mobility Functions to manage different handover types.
- Capability Info Indication function to transfer the UE capabilities to the MME.
- E-RAB management functions to setup, modify or release E-RABs.
- NAS Signalling transport function between the UE and the MME. S1AP transports a high layer signalling protocols, NAS.

Following is described the message structure of the S1AP protocol and S1SetupResquest message is used as an example:

This is the basic message description using ASN.1:

```
S1AP-PDU ::= CHOICE {
    initiatingMessage   InitiatingMessage,
    successfulOutcome   SuccessfulOutcome,
    unsuccessfulOutcome UnsuccessfulOutcome,
    ...
}

InitiatingMessage ::= SEQUENCE {
    procedureCode   S1AP-ELEMENTARY-PROCEDURE.&procedureCode  ({S1AP-ELEMENTARY-PROCEDURES}),
    criticality S1AP-ELEMENTARY-PROCEDURE.&criticality        ({S1AP-ELEMENTARY-PROCEDURES}{@procedureCode}),
    value       S1AP-ELEMENTARY-PROCEDURE.&InitiatingMessage  ({S1AP-ELEMENTARY-PROCEDURES}{@procedureCode})
}
```

The S1AP message is a choice structure with three possible message types, the initiating message, the successful outcome and the unsuccessful outcome. This implements the basic structure for a protocol with class 1 and class 2 elemental procedures such as the current one. Each message type has the same structure, thus only the Initiating Message definition is shown above.

The fields included in every message type are the procedure code, the criticality and the value. The procedure code is an integer to identify the procedure that will help to interpret the value field. The criticality defines the required actions when an error is found during the decoding. The value is a variable type depending on the procedure code and the message type choice.

According to these rules, the header can be defined as the message type choice, the procedure Code and the criticality. The header is encoded as depicted in Figure 11 using the BASIC-PER aligned encoding rules.

Below there are some value definitions for the given procedure.

```
s1Setup S1AP-ELEMENTARY-PROCEDURE ::= {
    INITIATING MESSAGE      S1SetupRequest
    SUCCESSFUL OUTCOME      S1SetupResponse
    UNSUCCESSFUL OUTCOME    S1SetupFailure
    PROCEDURE CODE          id-S1Setup
    CRITICALITY             reject
}
```

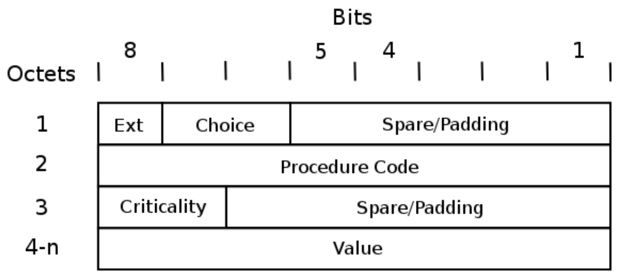


Figure 11: BASIC-PER aligned encoding of the S1AP header

This definition of the value can be interpreted as an index. Thus the s1Setup procedure links the message choice with a given structure and restricts the procedure code and criticality values. Below there is the definition of the Initiating message of the S1Setup procedure, linked to the S1SetupResquest value.

```
S1SetupRequest ::= SEQUENCE {
    protocolIEs         ProtocolIE-Container       { {S1SetupRequestIEs} },
    ...
}

S1SetupRequestIEs S1AP-PROTOCOL-IES ::= {
    { ID id-Global-ENB-ID     CRITICALITY reject  TYPE Global-ENB-ID  PRESENCE mandatory}|
    { ID id-eNBname           CRITICALITY ignore  TYPE ENBname        PRESENCE optional}|
    { ID id-SupportedTAs      CRITICALITY reject  TYPE SupportedTAs   PRESENCE mandatory}|
    { ID id-DefaultPagingDRX  CRITICALITY ignore  TYPE PagingDRX      PRESENCE mandatory}|
    { ID id-CSG-IdList        CRITICALITY reject  TYPE CSG-IdList     PRESENCE optional},
    ...
}
```

The S1Setup request has only one element, an IE-container. This IE container bundles all the other IEs present on the message.

The last block in the message structure is a restriction that applies to the protocol container. The restriction section of the message lists the allowed IE types with its IE Id, the criticality,the required IE order and whether their presence is mandatory or optional.

```
ProtocolIE-Field {S1AP-PROTOCOL-IES : IEsSetParam} ::= SEQUENCE {
    id              S1AP-PROTOCOL-IES.&id           ({IEsSetParam}),
    criticality     S1AP-PROTOCOL-IES.&criticality  ({IEsSetParam}{@id}),
    value           S1AP-PROTOCOL-IES.&Value        ({IEsSetParam}{@id})
}
```

Finally, the above definitions describe the IE structure, which contains an IE Id, a criticality field and a value field. The criticality field is again a variable type and contains the information the IE is transporting. Every IE value has a defined type on the S1AP standard. This value can be a container or a list to transport additional IE structures, which enables IE nesting as common mechanism to exchange multiple IE structures simultaneously.

The S1AP protocol is designed to be extensible in the future. The S1AP protocol allows the communication between two machines with different standard versions using the extension marker "..." to indicate where the protocol can be extended.

The explained message structure is similar to all S1AP messages and can be generalized. In this section only a summary of the protocol message structure is included, for more details see TS 36.413 [16]. The ASN.1 definitions of the protocol can be found on the clause 9.3.

3.4.2 NAS

Non Access Statum (NAS) is the signalling protocol used to communicate the UE with the MME. The protocol is defined on TS 24.301 [6].

The NAS is used to perform the following functions:

- EPS mobility management (EMM) of the user equipment (UE)
- EPS session management (ESM) to establish and maintain IP connectivity between the UE and the P-GW.
- NAS security, integrity protection and ciphering of NAS messages

This functionality differences allow separating the NAS procedures on two different types, the EMM procedures and the ESM procedures. The ESM functions are transported on an EMM IE. This separation of procedures with its own IE allows the execution of these procedures in parallel. This parallel execution for ESM and EMM allows creating two sublayers, i.e. the simultaneous execution of the EMM Attach procedure and Activate Default Bearer ESM procedure between the UE and the Network.

To implement the security function, the NAS message is ciphered and integrity protected and finally a modified message header is added to encapsulate it.

The NAS protocol is transported by the radio interface and by the S1-MME interface. On the S1-MME interface, the NAS messages are encapsulated on S1AP messages.

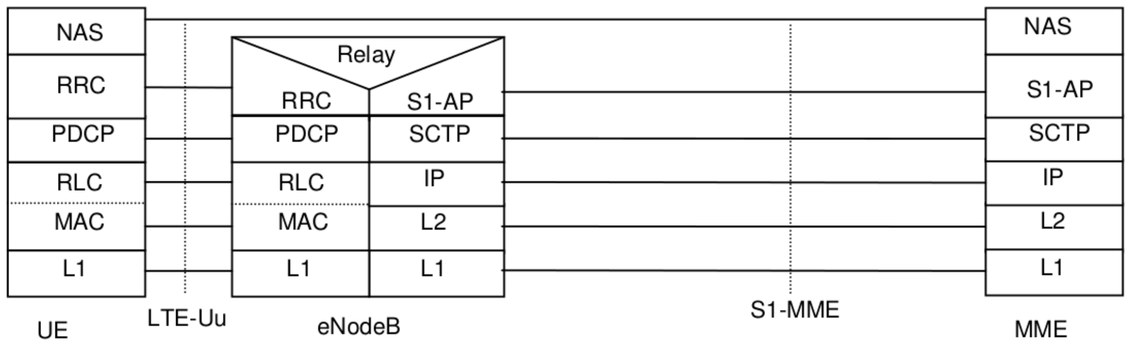


Figure 12: Non Access Stratum protocol stack, source TS 23.401 [3]

All NAS messages shall be integrity protected, except for a non authenticated UE. The ciphering on NAS messages in the network is an operator option.

The EPS security context groups all the security parameters. The EPS security context is identified by the Key Set Identifier for E-UTRAN (eKSI). The EPS security context contains the requited parameters for mutual authentication and the ciphering and integrity protection keys. The EPS authentication procedure between the UE and the MME creates the EPS security context.

The security context is taken into use by the UE and the MME after the security command procedure.

EMM The EPS mobility management (EMM) sublayer manages the UE mobility. The EMM procedures can be divided into three types:

- EMM common procedures: always initiated by the network, include GUTI reallocation, authentication, security mode control, identification, EMM information.
- EMM specific procedures: detach and combined detach and UE initiated, include attach and combined attach, normal tracking area updating and combined tracking area updating and periodic tracking area updating.
- EMM Connection Management procedures (ECM): transport of NAS messages and UE initiated service request and paging procedure.

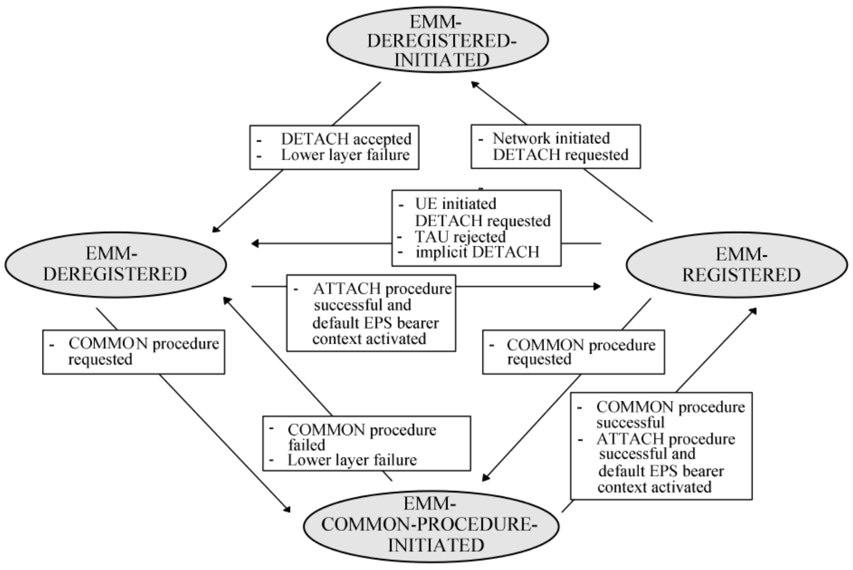
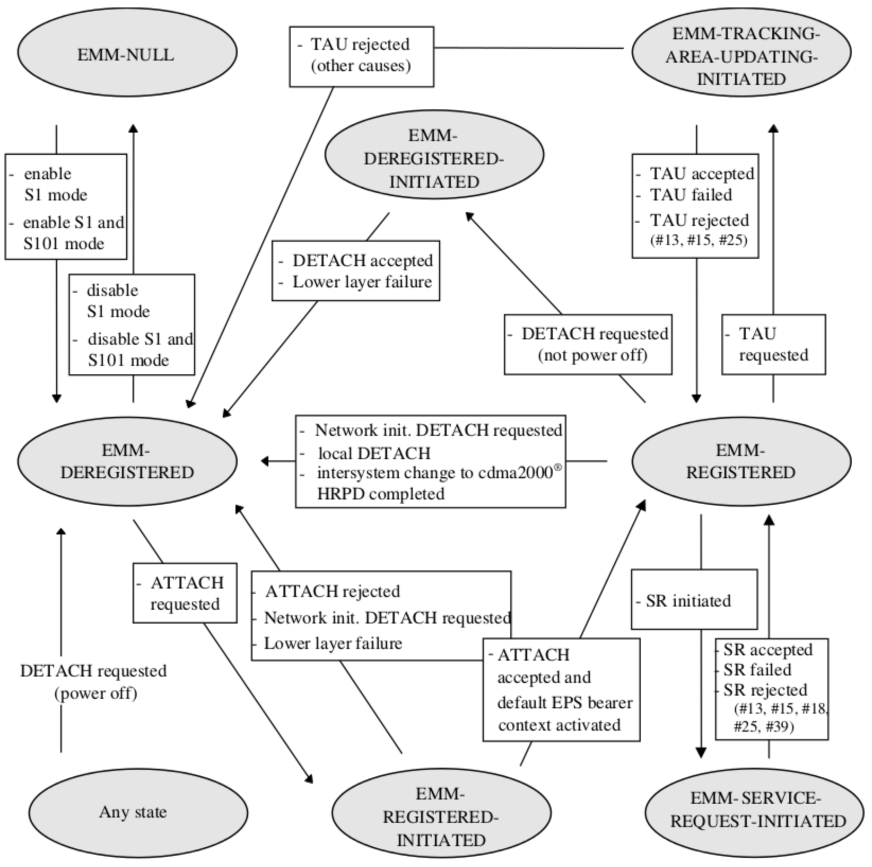

EMM-DEREGISTERED-INITIATED
- DETACH accepted
- Lower layer failure
- Network initiated DETACH requested
- UE initiated DETACH requested
- TAU rejected
- implicit DETACH
EMM-DEREGISTERED
EMM-REGISTERED
- ATTACH procedure successful and default EPS bearer context activated
- COMMON procedure requested
- COMMON procedure requested
- COMMON procedure failed
- Lower layer failure
- COMMON procedure successful
- ATTACH procedure successful and default EPS bearer context activated
EMM-COMMON-PROCEDURE-INITIATED

Figure 13: EMM state machine on MME, source TS 24.301 [6]

The EMM State Machine on the MME node is shown in Figure 13 and includes the following states:

- EMM-DEREGISTERED state, the UE is detached from the network. The expected procedures are attach, tracking area update or detach procedures. The MME will answer the messages related to these procedures.
- EMM-COMMON-PROCEDURE-INITIATED state, the MME has initiated a common procedure and it is waiting the response.
- EMM-REGISTERED state, the EMM context has been established and there is an active default bearer on the MME.

- EMM-DEREGISTERED-INITIATED state, the MME started a detach procedure and is waiting for the UE response.



EMM-NULL
- TAU rejected (other causes)
EMM-TRACKING-AREA-UPDATING-INITIATED
EMM-DEREGISTERED-INITIATED
- enable S1 mode
- enable S1 and S101 mode
- disable S1 mode
- disable S1 and S101 mode
- DETACH accepted
- Lower layer failure
- TAU accepted
- TAU failed
- TAU rejected (#13, #15, #25)
- DETACH requested (not power off)
- TAU requested
EMM-DEREGISTERED
- Network init. DETACH requested
- local DETACH
- intersystem change to cdma2000® HRPD completed
EMM-REGISTERED
- ATTACH requested
- ATTACH rejected
- Network init. DETACH requested
- Lower layer failure
- SR initiated
- ATTACH accepted and default EPS bearer context activated
- SR accepted
- SR failed
- SR rejected (#13, #15, #18, #25, #39)
DETACH requested (power off)
Any state
EMM-REGISTERED-INITIATED
EMM-SERVICE-REQUEST-INITIATED

Figure 14: EMM state machine on UE, source TS 24.301 [6]

The EMM State Machine on the UE is shown in Figure 14. The state machine in the UE is similar to the EMM state machine in the MME node. Some states are added to treat the class 1 procedures that the UE can initiate, Service request processes, tracking area update process and a initiating point NULL state. In addition to those procedures, there are some transition modifications.

ESM The EPS Session Management (ESM) sub layer handles the EPS bearer contexts and following are the two types of procedures:

- The activation, deactivation and modification of the EPS bearer contexts, initiated by the MME.
- The request for resources by the UE. These resources include the IP connectivity to a PDN or a dedicated bearer resource.

In addition, there are another two procedures, the ESM status procedure and the notification procedure that can not be included in these types.

The ESM state machine on the MME node is shown in Figure 15 and include the following states:

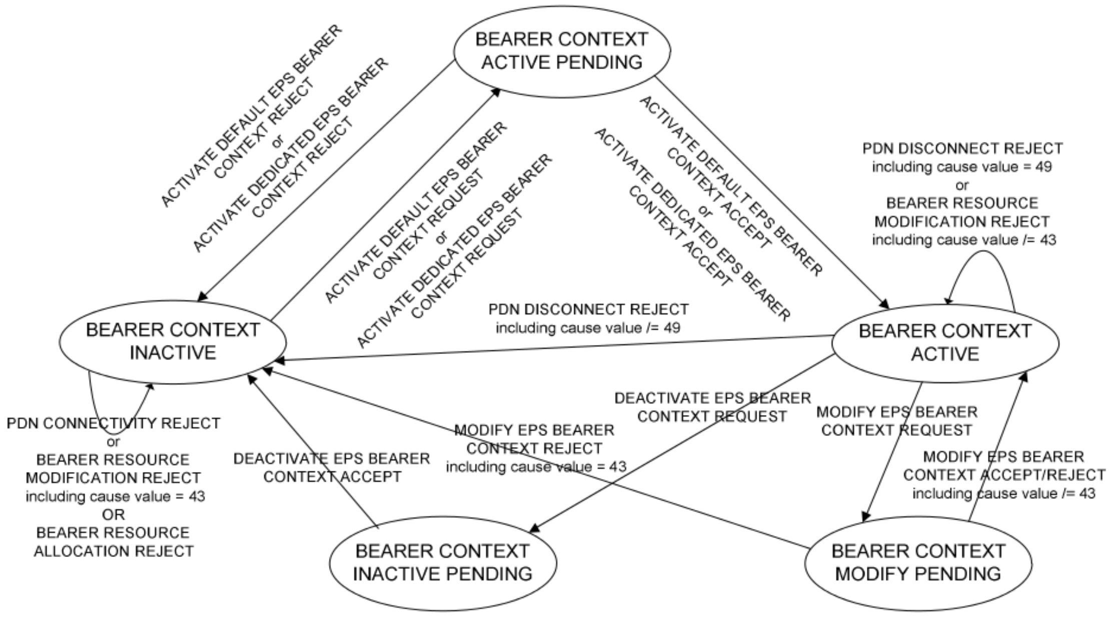

BEARER CONTEXT ACTIVE PENDING
ACTIVATE DEFAULT EPS BEARER CONTEXT REJECT or ACTIVATE DEDICATED EPS BEARER CONTEXT REJECT
ACTIVATE DEFAULT EPS BEARER CONTEXT REQUEST or ACTIVATE DEDICATED EPS BEARER CONTEXT REQUEST
ACTIVATE DEFAULT EPS BEARER CONTEXT ACCEPT or ACTIVATE DEDICATED EPS BEARER CONTEXT ACCEPT
PDN DISCONNECT REJECT including cause value = 49 or BEARER RESOURCE MODIFICATION REJECT including cause value /= 43
BEARER CONTEXT INACTIVE
PDN DISCONNECT REJECT including cause value /= 49
BEARER CONTEXT ACTIVE
PDN CONNECTIVITY REJECT or BEARER RESOURCE MODIFICATION REJECT including cause value = 43 OR BEARER RESOURCE ALLOCATION REJECT
DEACTIVATE EPS BEARER CONTEXT ACCEPT
MODIFY EPS BEARER CONTEXT REJECT including cause value = 43
DEACTIVATE EPS BEARER CONTEXT REQUEST
MODIFY EPS BEARER CONTEXT REQUEST
MODIFY EPS BEARER CONTEXT ACCEPT/REJECT including cause value /= 43
BEARER CONTEXT INACTIVE PENDING
BEARER CONTEXT MODIFY PENDING

Figure 15: ESM state machine on MME, source TS 24.301 [6]

- BEARER CONTEXT INACTIVE state, no Bearer contexts exist
- BEARER CONTEXT ACTIVE PENDING state, the network has initiated a bearer context activation procedure and it is waiting for the UE response.
- BEARER CONTEXT ACTIVE state, there is a bearer context active.
- BEARER CONTEXT INACTIVE PENDING state, the network has initiated a bearer context deactivation procedure and it is waiting for the UE reponse.
- BEARER CONTEXT MODIFY PENDING state, the network has initiated a bearer context modification procedure and it is waiting for the UE reponse.

The ESM state machine on the UE is shown in Figure 16. The ESM state machine depicted in the figure shows the first type of procedures, the ones related with the EPS bearer management. The second state machine at the bottom of the figure corresponds to the processing of the second type of procedures, handling the resource requests of the UE to the network.

The structure of these state machines is simple, the first state machine stores the status of the bearer contexts while the second one state machine in the bottom is generic and handles a class 1 elemental procedure. The left part of the state machine is the initial state and a transition is made when the procedure is initiated sending the request message, and the right part of the state machine is the state waiting response. The receiving of the network response triggers the transition to the initial state.

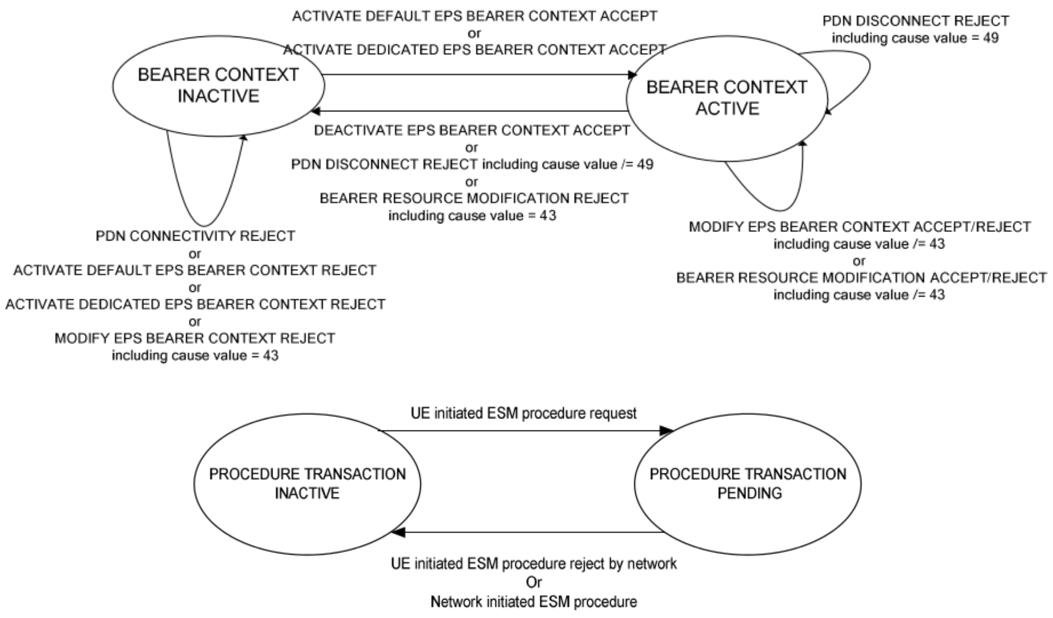
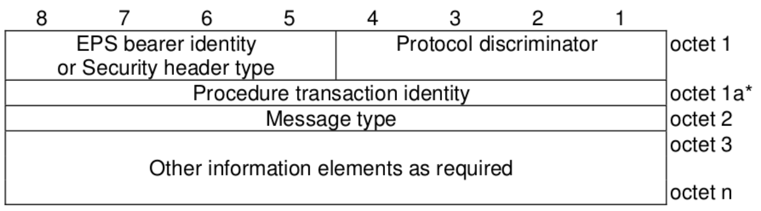

ACTIVATE DEFAULT EPS BEARER CONTEXT ACCEPT
or
ACTIVATE DEDICATED EPS BEARER CONTEXT ACCEPT
BEARER CONTEXT INACTIVE
BEARER CONTEXT ACTIVE
PDN DISCONNECT REJECT including cause value = 49
DEACTIVATE EPS BEARER CONTEXT ACCEPT
or
PDN DISCONNECT REJECT including cause value /= 49
or
BEARER RESOURCE MODIFICATION REJECT including cause value = 43
PDN CONNECTIVITY REJECT
or
ACTIVATE DEFAULT EPS BEARER CONTEXT REJECT
or
ACTIVATE DEDICATED EPS BEARER CONTEXT REJECT
or
MODIFY EPS BEARER CONTEXT REJECT including cause value = 43
MODIFY EPS BEARER CONTEXT ACCEPT/REJECT including cause value /= 43
or
BEARER RESOURCE MODIFICATION ACCEPT/REJECT including cause value /= 43
UE initiated ESM procedure request
PROCEDURE TRANSACTION INACTIVE
PROCEDURE TRANSACTION PENDING
UE initiated ESM procedure reject by network
Or
Network initiated ESM procedure

Figure 16: ESM state machines on UE, source TS 24.301 [6]



8 7 6 5	4 3 2 1	
EPS bearer identity or Security header type	Protocol discriminator	octet 1
Procedure transaction identity		octet 1a*
Message type		octet 2
Other information elements as required		octet 3 … octet n

Figure 17: General message organization example for a plain NAS message, source: clause 9 TS 24.301 [6]

Message structure The NAS message structure of a plain message is depicted in Figure 17.

Protocol Discriminator (PD) indicates the type of the protocol of the message. The values for the current protocols are: 2 for EPS session management messages and 7 for EPS mobility management messages.

Figure 17 represents both the ESM and EMM plain message. The differences on the header are the following. On the EMM messages, the higher part of the first octet corresponds to the security header type and the depicted 1a octet is not present. On ESM messages, the higher part of the first octet contains the EPS bearer identification and the 1a octet is present, containing the procedure transaction Id.

Security header type can contain the values shown in Table 1. There are some restrictions on these values. The 0011 value can only be used on a SECURITY MODE COMMAND message and the 0100 value can only be used on a SECURITY MODE COMPLETE message.

EPS bearer ID value range is 5 to 15, using the value 0 when no EPS is bearer assigned. The rest of the values are reserved. The other fields are self-explanatory

8	**7**	**6**	**5**	
0	0	0	0	Plain NAS message, not security protected
				Security protected NAS message
0	0	0	1	Integrity protected
0	0	1	0	Integrity protected and ciphered
0	0	1	1	Integrity protected with new EPS security context
0	1	0	0	Integrity protected and ciphered with new EPS security context
				Non-standard L3 message:
1	1	x	x	Security header for the SERVICE REQUEST message

All other values are reserved.

Table 1: Security Header type

and they indicate the procedure and message type of the NAS message.

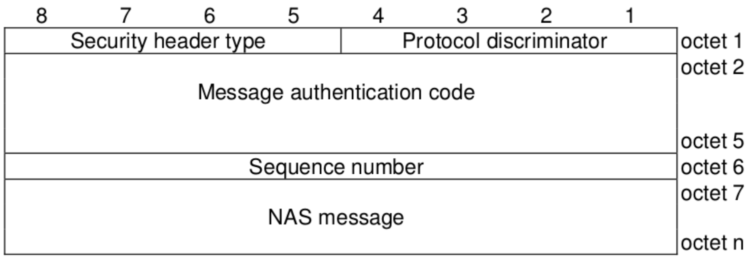


Figure 18: General message organization example for a security protected NAS message, source: clause 9 TS 24.301 [6]

The ciphered and/or integrity protected message has a different header as depicted in Figure 18. This header is added to the plain message in case of only protecting the integrity or the header is added to the ciphered message in case of ciphering and integrity protected message.

Message Authentication Code (MAC) is the information element containing the required information to protect or check the integrity of the message. The integrity protection algorithm is specified on 3GPP TS 33.401 [10].

The Sequence number (SN) is the eight least significant bits of the NAS COUNT value. There are two NAS COUNTERs on a security context, one for the uplink and the other for the downlink. The NAS COUNTER is represented by 24 bits and is constructed by a NAS sequence number on the 8 least significant bits together with a NAS overflow counter in the 16 most significant bits. However, NAS COUNTER is represented by 32 bits when the value is used on NAS ciphering and NAS integrity protection algorithms and is constructed by padding the previous 24 bits with 8 bits with the 0 value in the most significant bits.

The next relevant components in the message structure are the Information element structures which are described next.

Following are the different formats defined on 3GPP TS 24.007 [5]: V, LV, T, TV, TLV, LV-E, TLV-E. In these formats, the V corresponds to the Value field, the

L to the length indicator (LI), containing the length of the encoded value, and the T the Type field, containing the Information Element Identifier (IEI). The E indicates a new version of the IE structure whose length range has been increased (2 octets instead of the normal 1 octet), thus allow larger values. In addition, 5 IE categories are defined:

- Type 1: IEs of V or TV format with value part consisting of 1/2 octet.
- Type 2: IES of T format without value parts.
- Type 3: IEs of V or TV format with value part that has a fixed length of at least one octet.
- Type 4: IEs of LV or TLV format with value part consisting of zero, one or more octets.
- Type 6: IEs of LV-E or TLV-E format with value part consisting of zero, one or more octets and a maximum of 65535 octets.

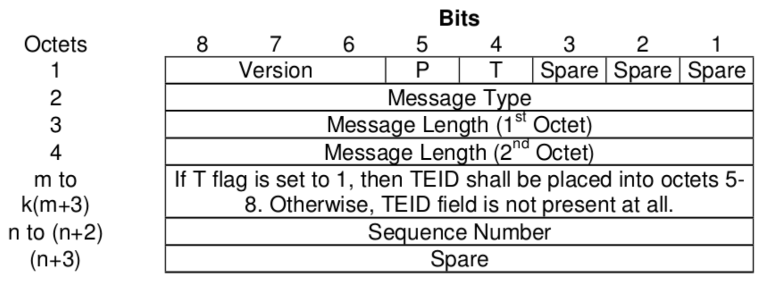
3.4.3 GTPv2

The GPRS Tunnelling Protocol (GTP) version 2 is the signalling protocol used to manage the GTP tunnels over a GTP based interface. The protocol is defined on TS 29.274 [8].

A GTP tunnel is identified with a Tunnel Endpoint Identifier (TEID), an IP address and a UDP port. These parameters are also used on the GTPv2 signalling protocol. The TEID is allocated by the receiving node and shall be used on further messages by the transmitting node. When the peer's TEID is not available and a TEID is required on the message, the TEID=0 is allowed.

The GTP protocol is used over UDP. The destination UDP port for an initial GTP-C message is the 2123 and is a registered port only for GTP-C.

Message structure The General GTPv2 Header structure is detailed in Figure 19.



	Bits							
Octets	8	7	6	5	4	3	2	1
1	Version			P	T	Spare	Spare	Spare
2	Message Type							
3	Message Length (1st Octet)							
4	Message Length (2nd Octet)							
m to k(m+3)	If T flag is set to 1, then TEID shall be placed into octets 5-8. Otherwise, TEID field is not present at all.							
n to (n+2)	Sequence Number							
(n+3)	Spare							

Figure 19: Format of the general GTPv2 Header, source: clause 5.1 TS 29.274 [8]

On the first octet, the highest 3 bits (6-8) correspond to the GTP version. The current version is 2. The next bits are flags to indicate the presence of optional fields, the P flag (bit 5) is the Piggybacking flag and the T flag (bit 4) represents the TEID flag. Finally, the spare bits are ignored on the decoding and are set to 0 by the sending endpoint. The spare octet at the end of the header has the same explained conditions.

When the TEID flag is active (set to 1), the TEID is present on the 5 to 8 octets. The TEID is present on all GTPv2 message, except the Echo Request, Echo Response and Version Not Supported Indication messages.

If a Piggybacking flag is set to 1, another GTPv2 message is present at the end of the current message. The piggybacked message should have its own header and body. The aim of the piggybacking is to speed up the message exchange between the endpoints but this functionality is optional.

The second octet represents the GTPv2 message type. The message type values can be consulted in the Table 6.1-1 "Message types for GTPv2" contained on TS 29.274 [8].

The 3rd and 4th octets indicate the message length in octets. The message length excludes the mandatory header part, the first 4 octets.

If the TEID flag is set to 1, the next 4 octets contain the TEID and the following 3 octets contain the sequence number, on some cases only the 3 octets with the sequence number are included. Finally the last octet is a spare octet to align the header structure.

After the header, the message continues with the body formed with a set of individual or grouped IEs. All the GTPv2 IEs have the TLIV structure depicted in Figure 20.

	Bits							
Octets	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
1	Type = xxx (decimal)							
2 to3	Length = n							
4	Spare				Instance			
5 to (n+4)	IE specific data or content of a grouped IE							

Figure 20: Format of the TLIV IE structure, source: clause 8.2 TS 29.274 [8]

The mandatory IE fields are:

- **Type**: the first octet corresponds to the IE Type used to identify the IE. The possible values are defined on the clause 8.1 of the GTPv2 standard.

- **Length**: the following 2 octets are the length field. The length field contains the length of the IE excluding the first four common octets, thus containing only the value length.

- **Instance**: on the 3rd octet, only the lower half is used as the instance field. The instance is used to identify different IE parameters with the same IE type on a message [8].

Finally, the Value field contains the useful data. The value field is encoded depending on the information that is carries. When the IE is a grouped IE type, it contains other nested IE structures. For more details, the TS 29.274 standard [8] can be consulted.

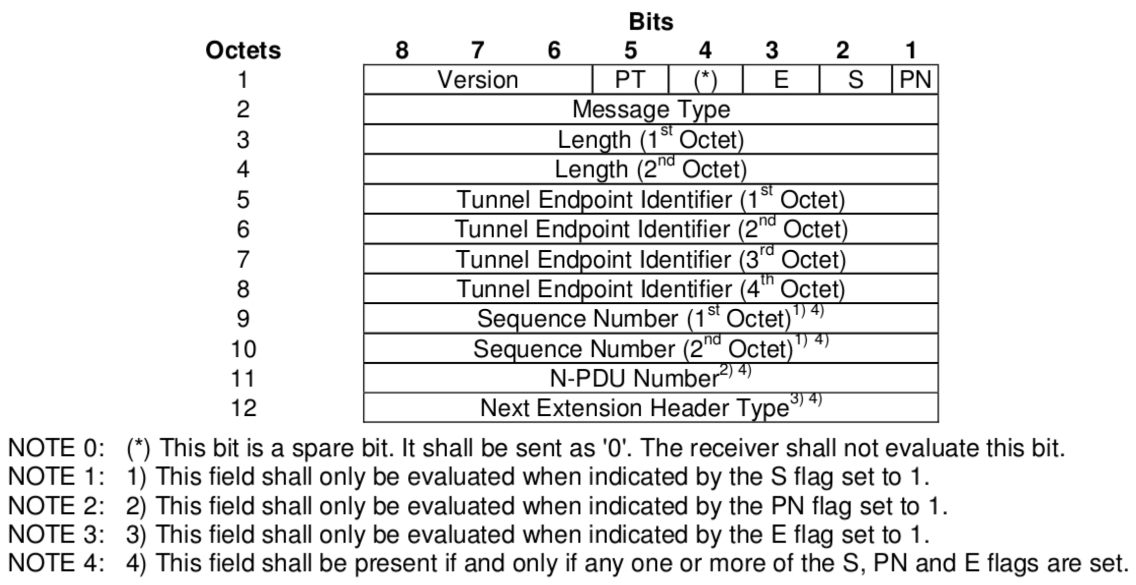
3.4.4 GTP-U

The GPRS Tunneling Protocol User Plane (GTP-U or GTPv1-U) is the protocol used to tunnel the UE data on the mobile core network. The GTP-U is defined on the TS 29.281 [9] standard.

As seen on the GTPv2 description above, the GTP tunnel is identified with a Tunnel Endpoint Identifier (TEID), an IP address and the registered UDP port 2152. The GTP can be seen as a header added to the UE IP datagram to encapsulate the user information. The encapsulated message is called G-PDU. The G-PDU contains the GTP header and the T-PDU. A T-PDU is a user data package, i.e. a UE IP datagram.

In addition some IE are available to form a limited set of signalling messages. These signal messages are the Path Management messages: Echo request, Echo response and Supported extension headers notification; and the Tunnel Management messages: Error indication and End marker.

Message structure The protocol Header is depicted in Figure 21.



	Bits							
Octets	8	7	6	5	4	3	2	1
1	Version			PT	(*)	E	S	PN
2	Message Type							
3	Length (1st Octet)							
4	Length (2nd Octet)							
5	Tunnel Endpoint Identifier (1st Octet)							
6	Tunnel Endpoint Identifier (2nd Octet)							
7	Tunnel Endpoint Identifier (3rd Octet)							
8	Tunnel Endpoint Identifier (4th Octet)							
9	Sequence Number (1st Octet)1) 4)							
10	Sequence Number (2nd Octet)1) 4)							
11	N-PDU Number2) 4)							
12	Next Extension Header Type3) 4)							

NOTE 0: (*) This bit is a spare bit. It shall be sent as '0'. The receiver shall not evaluate this bit.
NOTE 1: 1) This field shall only be evaluated when indicated by the S flag set to 1.
NOTE 2: 2) This field shall only be evaluated when indicated by the PN flag set to 1.
NOTE 3: 3) This field shall only be evaluated when indicated by the E flag set to 1.
NOTE 4: 4) This field shall be present if and only if any one or more of the S, PN and E flags are set.

Figure 21: Format of the GTP-U header, source: clause 5.1 TS 29.281 [9]

The GTP header is a variable header. The flags PN, S or E signal the presence of optional fields on the header. The first 8 octets are the mandatory part and contain the following fields:

- Version field: Field to indicate the protocol version. It is set to 1.
- Protocol Type (PT): This field is used as protocol discriminator between the GTP (with value 1) and GTP' (with value 0).

- Extension flag (E): This flag indicates the presence of a meaningful extension header when set to 1.
- Sequence Number flag (S): This flag indicates the presence of a meaningful value of the Sequence Number field when set to 1.
- N-PDU Number flag (PN): Indicates the presence of a meaningful value on the N-PDU number field explained below when it is set to 1. If PN is set to 0, the N-PDU number field must be ignored.
- Message type: this field indicates the type of the G-PDU message.
- Length: This 2 octet field indicates the length of the message payload. The optional fields of the header are considered as payload thus included on this field.
- Tunnel Endpoint Identifier (TEID): This field unambiguously relates the tunnel with the UE context. On the signalling messages, except on the End marker message, the TEID is set to 0.

When any of the optional flags are set to one, all the optional fields are included in the header, but only the ones with the active flag are processed. The optional fields are the following:

- Sequence Number: Used when transmission order is required. It includes an increasing sequence number.
- N-PDU number: This field is not used on LTE. The exact meaning depends on the scenario.
- Extension Header Type: this field indicates the type of the extension header that follows this header.

The IE used on the signalling messages follow the structure TLV and TV. For more information consult the GTPv1 specifications [9].

3.5 Procedures of Interest

This section describes the implemented procedures in detail. These procedures are defined on TS 23.401 [3] but they have been simplified to match the implemented.

The procedure explanation includes traffic diagrams to show the message exchange between nodes graphically. The procedures are composed by elemental procedures.

3.5.1 Attach Procedure

The attach procedure is used to register the UE/user on the network. Always-on IP connectivity is established after configuring a Default EPS Bearer. The message flow can be observed in Figure 22. The HSS messages have been removed to simplify the figure and because the HSS is emulated with a simple Database (DB) on the current implementation of the MME.

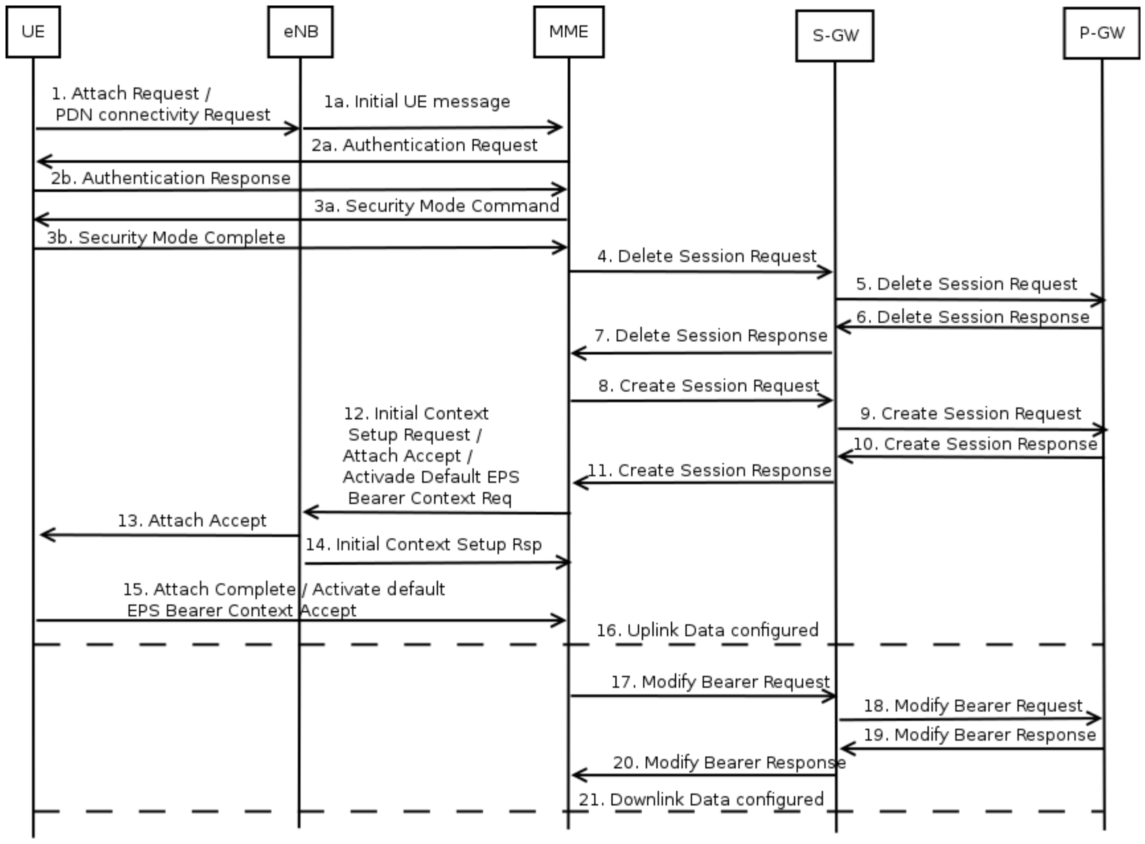

UE
eNB
MME
S-GW
P-GW
1. Attach Request / PDN connectivity Request
1a. Initial UE message
2a. Authentication Request
2b. Authentication Response
3a. Security Mode Command
3b. Security Mode Complete
4. Delete Session Request
5. Delete Session Request
6. Delete Session Response
7. Delete Session Response
8. Create Session Request
9. Create Session Request
10. Create Session Response
11. Create Session Response
12. Initial Context Setup Request / Attach Accept / Activade Default EPS Bearer Context Req
13. Attach Accept
14. Initial Context Setup Rsp
15. Attach Complete / Activate default EPS Bearer Context Accept
16. Uplink Data configured
17. Modify Bearer Request
18. Modify Bearer Request
19. Modify Bearer Response
20. Modify Bearer Response
21. Downlink Data configured

Figure 22: Attach Procedure

The procedure is triggered when the UE sends an Attach request to the eNB with a PDN connectivity request NAS messages (1), these messages are encapsulated on the eNB inside a S1-AP Initial UE message (1a).

When the MME receives the initial UE message checks whether an active security context is available. In the case there is no active context, the NAS layer triggers the Authentication procedure (2a, 2b). This procedure is used for mutual authentication and it is based on the Authentication and Key Agreement protocol [10].

After the authentication, the Security mode control Procedure (3a, 3b) initialize the NAS signalling security using the corresponding EPS NAS security algorithm and keys with the agreed EPS security context. The current MME implementation doesn't use this procedure because the NAS security is not implemented on the NAS library.

Once the UE is authenticated and the security context enabled, the MME starts the configuration of the Uplink tunnel endpoint. If the UE has an active Session on the S-GW, the MME triggers the Delete session procedure on the S11 interface. On the figure can be observed the involved messages between the MME and the S-GW (4-7) and how the S-GW forwards the configuration to the P-GW (5-6).

The attach procedure continues with the S1AP Create Context procedure to setup the Uplink endpoint. The Create Context procedure follows the same sending path as the Delete Session procedure, forwarding the configuration to the P-GW if it is decided by the S-GW (8-11).

After these procedures, the Uplink endpoint tunnel is half configured. The S1AP initial Context Setup procedure informs the eNB about the S-GW endpoint tunnel to finish the Uplink tunnel (12). The Initial Context Setup Request transports the NAS EMM and ESM replies of the Attach EP and Activate Default EPS Bearer Context. These messages transport the information about the configured EPS Bearer to the UE (13).

The next messages (14 - 15) doesn't have any specific order and the MME is prepared for both, no matter the order. The message 14 is eNodeB reply to the Initial Context Setup and contains the information required to configure the downlink tunnel to the UE. The message 15 inform the MME about the success (or failure) of the NAS attach procedure and Activate Default EPS Bearer Context. With message 15, the EPC has the knowledge that the Uplink tunnel is on use (16).

Finally, the MME forwards the Downlink Tunnel configuration information to the Gateways using the Modify Bearer procedure (17-20). After the success of this procedure, the Downlink Tunnel is configured (21).

After these procedures, the UE is attached to the EPC and have connectivity to the PDN.

3.5.2 Detach Procedure

The Detach procedure can be initiated by the UE or the EPC. In case the procedure is initiated by the UE, it informs the EPC that it does not want EPS access any longer. If the Detach procedure is initiated by the EPC, it informs the UE that it does not have access to the EPC any longer.

In addition a detach can be either explicit or implicit. On the explicit detach the UE or the MME request the detach, but on an implicit detach the EPC detaches the UE without notifying it because it presumes the connection is not available any longer, i.e. for radio conditions.

The message exchange on the detach procedure is depicted in Figure 23. The arrows on the initial message (1) and the response message (6) are to illustrate the possibility to initiate the procedure by both endpoints. These messages correspond to the NAS detach procedure.

After the initiating message, the MME starts the deallocation of the tunnel resources with the Delete Session Procedure (2 - 5). The S-GW is the responsible to deallocate the tunnel resources on the P-GW.

After these messages, the UE is detached from the network but there is still a UE context on the eNB. For this purpose, the MME initiates the UE Context Release but this procedure is not included in the figure.

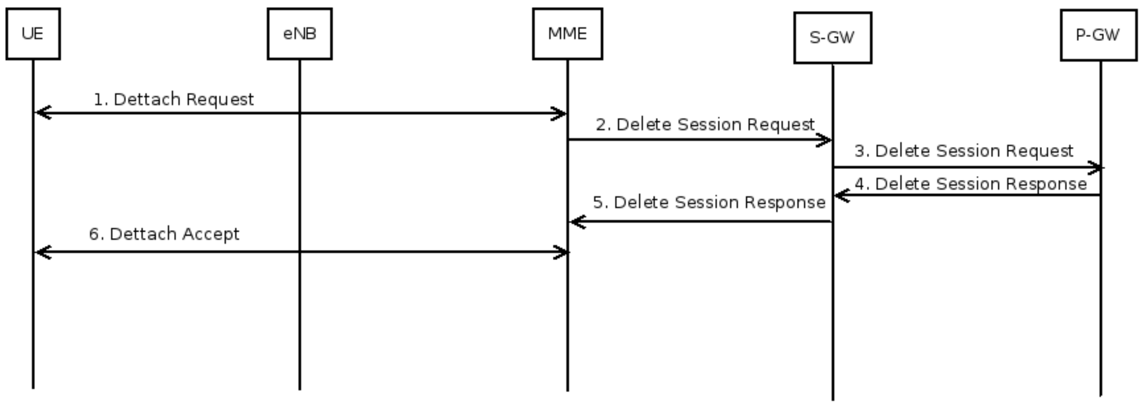

UE
eNB
MME
S-GW
P-GW
1. Dettach Request
2. Delete Session Request
3. Delete Session Request
4. Delete Session Response
5. Delete Session Response
6. Dettach Accept

Figure 23: Detach Procedure

3.5.3 X2 Handover

The X2 based Handover procedure uses the X2 interface as a reference point to hand over a UE from a source eNodeB to a target eNodeB. On the X2 base Handover, the MME is unchanged and there are 2 different variants depending if the S-GW is reallocated. In addition to the X2 interface between the source and target eNodeBs, this handover procedure relies on the S1 interface between the Source eNodeB and the MME and the S1 interface between the MME and the Target eNodeB.

Figure 24 depicts the X2 based Handover message exchange. The S-GW reallocation is not considered.

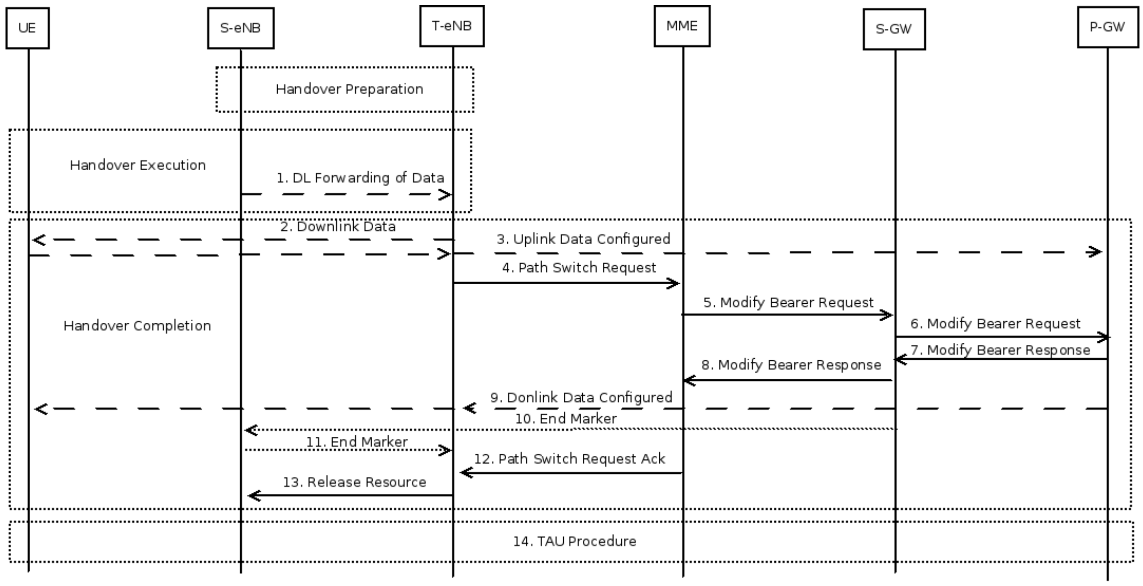


Figure 24: X2 based Handover Procedure

The Handover Preparation and Handover Execution is performed using the X2 interface. The downlink data is forwarded from the source eNodeB to the target eNodeB using the X2 interface (1 - 2) until the Handover Completion. The uplink tunnel information configuration is sent using the X2 interface so the uplink tunnel is already configured when the UE arrives on the target eNodeB (3).

The EPC doesn't know about the handover until this last step when the UE is already located on the target eNodeB. After the handover, the target eNodeB request the modification of the downlink tunnel on the S-GW endpoint using a Path Switch Request procedure (4, 12).

When the MME receives the Path Switch Request with the E-RABs to be switched it modifies the S-GW and the P-GW tunnel endpoint configuration with the Modify Bearer procedure (5 - 8) as seen on the attach procedure. After the Modify Bearer procedure, the downlink tunnel is operative.

The downlink data is sent to the UE after a reordering on the correct sequence either the packet comes from the source eNodeB during the forwarding or directly from the S-GW. The last packets from the old path are marked with the End Marker by the S-GW (10 - 11) to inform both eNodeBs about the path switch.

Finally, the Target eNodeB send a Release Resource message (13) to inform the Source eNodeB about the success of the handover and trigger the release of resources.

The UE initiates the Tracking Area Update (TAU) if it is required by one of the conditions detailed on TS 23.401 [3].

3.5.4 S1 Handover

The S1 based Handover procedure is used when the X2 interface is not available or a X2 based Handover can not be used. This procedure may reallocate the MME and/ or the S-GW, but this case is not considered on current implementation.

The message exchange to perform the S1 based Handover procedure are depicted in Figure 25.

The handover starts with the Handover preparation when the Source eNodeB initiates the Handover Preparation procedure with the Handover Required message (1). After this message, the MME prepares the resource allocation on the target eNodeB with the Handover Resource Allocation procedure (2 - 3) and request either an indirect or direct data forwarding tunnel on the S-GW depending if there is an available X2 connection between the source and the target eNodeBs (4 - 5). The Handover preparation ends with the Handover Command (6) message to the source eNodeB informing the Source eNodeB about the allocated resources.

The Handover Command sent to the UE (7) marks the border of the Handover Execution. The UE mobility is performed maintaining the downlink connectivity with the configured forwarding tunnel, either direct (10) or indirect (11). At the same time, the source eNode sends its current status to the MME (8) and the MME sent the status to the target eNodeB (9). When the UE arrives at the target eNodeB it sends the Handover Confirm message (12) to the eNodeB. The Uplink data was configured (13) during the resource allocation procedure on the Handover Preparation period and the UE can use the uplink tunnel upon its arrival.

After the Handover execution starts the Handover completion with the Handover notify (14) from the target eNodeB to the MME. The Handover completion is similar to the explained on the X2 based Handover. The downlink is configured

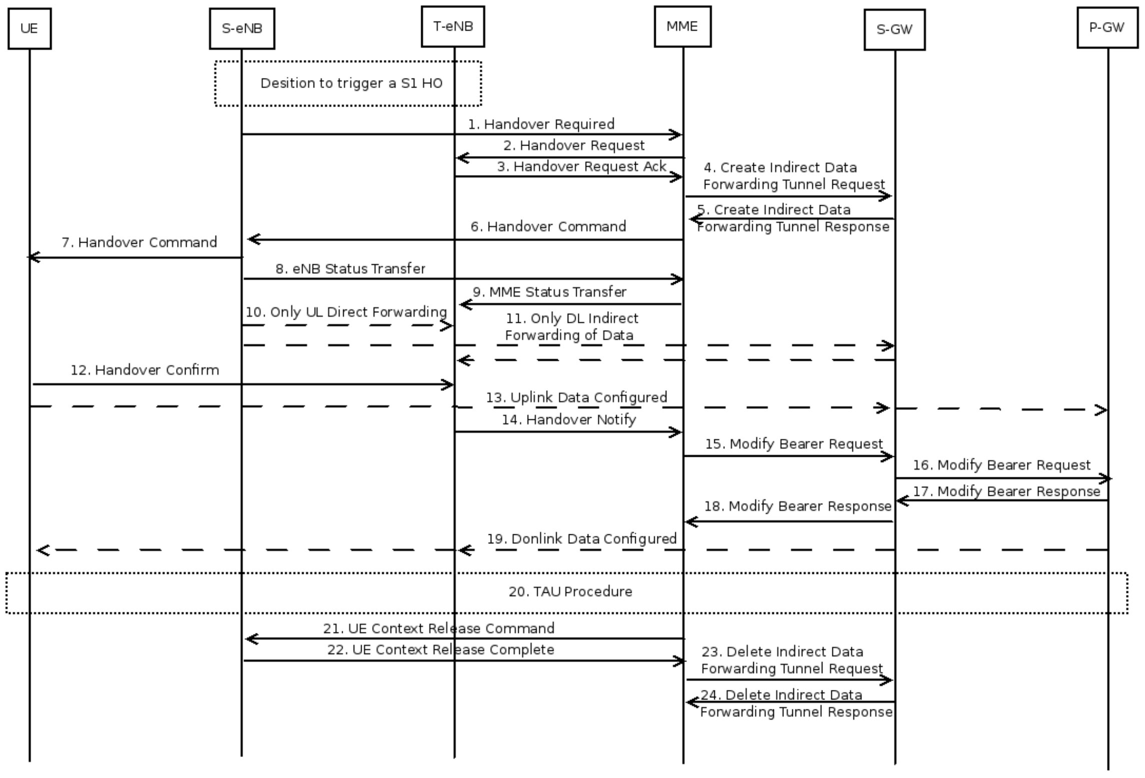

UE
S-eNB
T-eNB
MME
S-GW
P-GW
Desition to trigger a S1 HO
1. Handover Required
2. Handover Request
3. Handover Request Ack
4. Create Indirect Data Forwarding Tunnel Request
5. Create Indirect Data Forwarding Tunnel Response
6. Handover Command
7. Handover Command
8. eNB Status Transfer
9. MME Status Transfer
10. Only UL Direct Forwarding
11. Only DL Indirect Forwarding of Data
12. Handover Confirm
13. Uplink Data Configured
14. Handover Notify
15. Modify Bearer Request
16. Modify Bearer Request
17. Modify Bearer Response
18. Modify Bearer Response
19. Donlink Data Configured
20. TAU Procedure
21. UE Context Release Command
22. UE Context Release Complete
23. Delete Indirect Data Forwarding Tunnel Request
24. Delete Indirect Data Forwarding Tunnel Response

Figure 25: S1 based Handover Procedure

using the Modify Bearer procedure (15 - 19) and the old or temporal resources are deallocated with the UE Context Release procedure (21 - 22) or the Delete Indirect Data Forwarding Tunnel procedure (23 - 24) if used. The TAU procedure may be triggered by the UE.

3.6 Conclusions

This chapter introduced LTE definitions and technical descriptions included in the 3GPP standards needed for the MME implementation. The focus has been on the EPC and the used protocols. The EPS bearer concept is important to be understood to facilitate a future integration with SDN networks. Finally, the chapter describes a basic set of procedures to be supported by the MME implementation.

4 MME Design and Implementation

The current chapter details the MME design and implementation. Section 4.1 explains the basic architecture of the MME implementation. Section 4.2 describes the basic software components of the MME, the engine and how the state machines are implemented. Section 4.3 complements the previous section with some singularities of the state machines implemented for each interface. Next section 4.4 includes the description of the designed data base to simulate the HSS. Section 4.5 details developed libraries to encode and decode the different protocols used by the MME.

Finally, Sections 4.6 and 4.7 explain parallel tools and tests used to verify the MME implementation.

4.1 Design decisions.

Before initiating any development, some open source initiatives that claim having implemented LTE network elements were analyzed. An experimental project implementing old 3GPP standards for mobile networks (openBSC [43]) has been found. However, the conclusion was that there is no current open source implementation of the MME available at the time of initiating this work.

The main goal is to develop a MME application to build an experimental EPC with basic functionality that can be used for further development of LTE network based on SDN technologies. We have available from a third party company an emulator to simulate the UE and the eNB. We can then use the SAE-GW (S-GW and P-GW) functionality from another open source project [41]. However, there is no EPS compatible HSS open source project available so we will simulate the HSS with a basic Database (DB).

In order to develop the desired EPC functionality with the available nodes, the MME shall include the S1 and the S11 interfaces, but the S6a interface will be internally emulated with an internal connection to the DB.

The required protocols on the interfaces are SCTP, S1-AP, NAS and GTPv2. There is an SCTP protocol implementation on the Linux kernels, so to simplify the development, the Linux platform has been chosen. The rest of the protocols are not mature enough, thus there are no open source implementations available.

The Open BSC project has a GTP-C implementation on an Open GGSN subproject, but only the version 1 of the protocol is available. The GTPv2 coder/decoder functions have been implemented considering the Open BSC architecture design. The S1AP protocol is defined using ASN.1 so it can be compiled to get a coder/decoder library of the protocol. However, the available open source ASN.1 compilers do not work with the required BASIC-PER aligned encoding. Finally, some projects using the NAS protocol have been found, but the lack of modularity and the immaturity of them discourage their use and instead all the required components and protocols had to be implemented from scratch.

Given the situation, the GTP library is upgraded to version 2 and the S1AP and NAS encoder and decoder functions are developed from scratch on two libraries to achieve modularity and simplify the MME structure. That will allow the usage of these libraries in other future projects. The main of the software structure are MME, S1AP, NAS and libgtp, generating the *mme* executable and the libs1ap.la, libnas.la and libgtp.la library binaries.

These libraries only implement the encoding and decoding functionality, hence the state machines of the interfaces should be developed within the MME. The same modularity strategy has been taken on the MME application. The different state machines are divided by interface and protocol, hiding the inner complexity but maintaining a common structure.

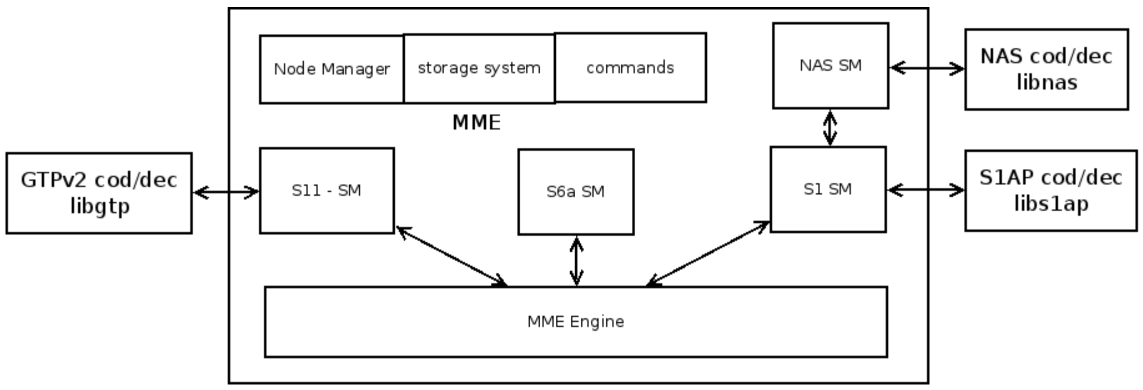


Figure 26: MME structure design

An engine is defined to execute the states sequentially and common structures are developed to allow the desired modularity. The S1, S11 and S6a modules use the common state structures, but the NAS state machines are hidden on the S1 interface module because it is a higher layer protocol. The S6a interface module has no external communication because it is an emulator of the real interface. Instead of having a HSS, the MME accesses a database containing the subscriber information.

Other modules have been added to process more specific functionality, as access to configuration files, the storage of the UE information on hash tables, etc. In addition, the MME can receive operator commands on run time. To implement this functionally another module is added, but it is simpler than the normal interfaces. This command line module is meant to follow the same structure as the normal interfaces, coder/decoder functions and state machine structures.

On the project folder, other folders can be found containing some parallel applications and tools:

- MMEcmd: Tool to send commands to the MME engine locally or remotely.
- Example applications such as eping to send GTPv2 pings.
- Testing framework to perform unit testing during the development and track any improvement breaking a working functionality.

4.2 Engine

The engine is the main part of the MME where the rest of the functionality is built on. It is basically a queue with priorities and a loop to process the states from the state machine and detect any incoming packet.

A single thread process approach is chosen to simplify the development. This can be changed in the future, creating a processing pool to handle multiple users at the same time or a thread per MME interface to increase the incoming packet acceptance rate. Even both approaches can be considered once the restrictions and requirements are identified.

To allow the maximum scalability, the libevent[44] library has been chosen to receive the incoming packets. This library is used on some successful open source projects and it is proven to be scalable[1]. It provides a mechanism to execute callback functions when a new message is received.

In order to include the libevent's loop in the engine's loop, the library has some options available to only iterate a single time. These options allow the developer to include additional actions within the event processing. On the current engine, these other actions are the processing of the states and the processing of a new received command.

The explained loop structure is implemented on the engine_main() function of MME_engine.c. This function loop is called from the engine_initialize() function after starting all the required contexts and structures, thus this last one is the function to start the mme from the main function. In addition, the engine provides the structures of the state machines and some functions to start, modify or delete them.

On the engine files, the t_process and t_signal structs are included to store the information during the state processing of the state machines. The t_process contains the information of the current state and the storage of other signals, to execute this process a signal should be send to the engine. That means that a new or existing signal is included in the engine signal's queue. That signal has a reference to the origin and destination process and a name to identify it.

This is the t_process structure:

```
struct t_process{
  engine_stateFunc       next_state;
  struct t_process       *parent;
  void                   *data;
  struct t_signal_queue  *firstSignal;
  struct t_signal_queue  *lastSignal;
  bool                   stop;
};
```

[1] The libevent library provides the best event mechanism offered on the OS with a common API. For more information about these mechanisms see Dan Kegel's "The C10K problem" webpage [45].

The parameters are next_state containing the function callback to process the next state, the parent process reference, containing the process that has created the current one, the data reference, usually an EndpointStruct_t struct, a signal queue to store pendent signals for this process and finally a stop flag to free the process structure on the next execution.

The t_signal structure has this format:

```
typedef struct t_signal{
  enum t_signal_name name;
  int                priority;
  struct t_process   *processTo;
  struct t_process   *processFrom;
  void               *data;
  void               (*freedataFunc)(void *);
}Signal;
```

The first field, the signal's name identifies the signal and communicates its purpose to the receiving process. The priority field purpose is to order the signal queue on the engine. The next fields are self-explanatory since they contain the destination and source process involved in this signal. Finally, a data reference and the function to deallocate it ends the signal format. Usually, the data contains the involved decoded messages.

There are other complementary structures as this t_signal_queue, whose purpose is to construct a signal queue, to perform other complementary functionality, but the commented ones are the most important.

```
struct t_signal_queue{
  struct t_signal       *signal;
  struct t_signal_queue  *next;
};
```

Finally, all the state functions (except NAS) have this format to be executed on the engine:

```
typedef int (*engine_stateFunc)(struct t_signal *signal);
```

The return parameter can be 1 or 0 depending if the current signal is saved on the current process after its execution or it is deallocated. The parameter is a reference to the signal send to the process.

The combination of the process structures to store the current state, the different signals available and the possibility to store them to continue its process on future transitions form a complex and flexible state machine structure with memory.

On the interface modules, other functions are implemented to reduce the size of the state functions and increase its modularity. The low level tasks are implemented in functions like this one:

```
static uint8_t TASK_MME_S1___S1Setup(Signal *signal);
```

These functions allow to group the parts of the code where the decisions are taken on the state functions and isolate the packet creation, validation and other functions on these TASK routines.

4.3 Interface State Machines singularities

4.3.1 S1 State Machine

This is the most complex interface on the MME. It receives the UE requests and is connection oriented. In addition, with its NAS layers has the biggest protocol stack.

The connection oriented has been implemented using a process structure per eNB association managing the end-point. In addition every UE has its own process structure storing its state on the state machine. This process is maintained while the UE context exists on the MME.

4.3.2 NAS State Machine

The NAS layer has its own state machine structure because it is on a different level of the stack.

The process structure is the same as the one used on the S1 because it is on the same interface, but on different levels of the stack. The API of its implementation has only two entry points:

```
void NAS_process(uint8_t *returnbuffer, uint32_t *bsize, void *msg,
    uint32_t size, Signal *signal);
void NAS_sessionAvailable(uint8_t *returnbuffer, uint32_t *bsize,
    Signal *signal);
```

The first is to process a received message passed with the *msg* pointer and the *size* parameter containing the buffer length. The *signal* parameter is the same used on the S1 state. Finally, if there is any result to be send to the lower layer it is stored on the *returnBuffer* with the *bsize* size.

The second function is used to continue processing a NAS message, it is used to return to NAS processing when a workflow is needed to continue to the lower layer in order to acquire some additional information. Usually, it is used when there is a need to populate a NAS transport IE element without receiving any NAS message. Due to this functionality, the parameters are the same as the other function without the msg and size, because there is no input NAS message to be processed.

The first function is used to process both ESM and EMM messages, thus it is also used within the internal implementation although the NAS messages received on the S1 NAS IEs are EMM messages containing other IEs with the ESM messages.

Although being a higher layer, the state-task differentiation has been maintained on the NAS state machine implementation.

4.3.3 S11 State Machine

This interface is not connection oriented, thus the t_process structs are temporal, there is no permanent t_process struct per user, instead a t_process struct is created when a transition to the S11 state machine is required and it is dealocated when the workflow returns to the S1 state machine.

4.3.4 S16a State Machine

The same comments about the t_process structs on the S11 can be applied on the S6a state machine. This interface state machine has a great difference from the others, it is emulating the interface, thus there is no communication to any HSS.

The HSS is implemented using MariaDB [46], a MySQL open source substitute. The state machine files (MME_S6a) have been separated from the SQL queries and HSS specific functions files (HSS) in order to allow a future implementation of a real interface. The HSS files emulate the HSS features, thus in the future, these will be deleted and the MME_S6a files will be modified in order to send and receive the diameter packets. The Database design is shown in Section 4.4.

4.4 Database Design

The Database included in this project acts as an HSS emulator. The information to be stored on the HSS is detailed on clause 5.7.1 of TS 23.401 [3]. The fields of the database have been selected according to the standard and the requirements of the use cases.

In order to present the database structure, the figures have a common notation. The tables are displayed as square boxes and are called entities. The columns of the tables are shown as spherical boxes and are called attributes. Only the termination attributes have a representation of the database, thus the complex attributes such as *Subscriber Qos Profile* show the logical aggregation of attributes on a higher logical level. There are some special attributes called primary keys to read and write the information on the database, these primary keys are represented with an underlined attribute name and they are the index to query or modify the database information.

To avoid future performance issues, the database has been normalized during its design until the Third Normal Form (3NF). The normalization process aims to organize the information efficiently, this is accomplished eliminating redundant data and considering the data dependencies, thus only related data is stored on the same table. A brief explanation of normalization can be found here [50].

The resulting general structure is presented in Figure 27. It has 4 tables representing the identified entities during the design process. These entities are explained bellow and the attributes for each entity are detailed in Figures 28, 29, 30, 31. A complete structure of the database can be consulted in Appendix A.1.

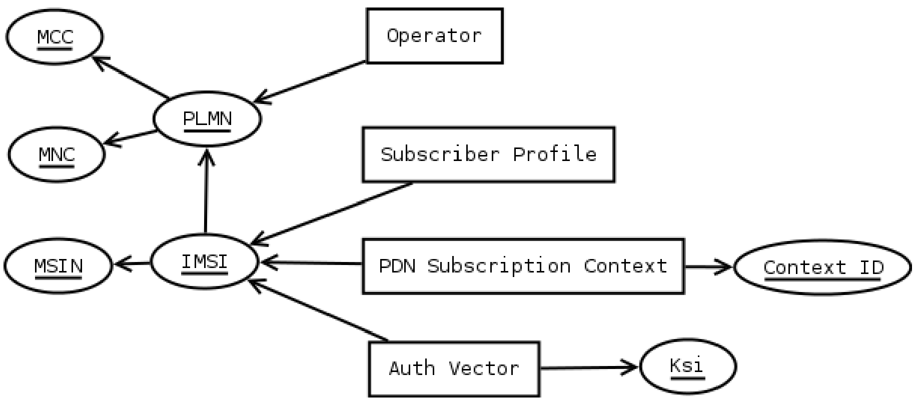


Figure 27: HSS database structure design

Subscriber Profile This is the main entity because the HSS functionality is to store the mobile subscriber information. The primary key is of this table is the IMSI, a compound key formed by the MCC, the MNC and the MSIN. The IMSI is the adequate key because is identifies a mobile subscriber on a unique way in any mobile network in the world. This value has been split on its components to facilitate its use on other tables.

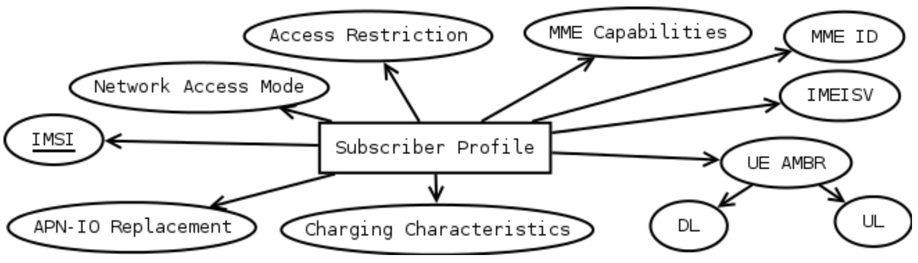


Figure 28: Subscriber Profile entity with attributes

Operator The aim of this entity is to store the operator related information. Almost all this information is related to security parameters like the OP and AMF. As expected, a subscriber profile has a single operator, but an operator manages many subscriber profiles. The primary key of this table is a compound key formed with the MNC and MCC. These attributes form the PLMN and are included in the IMSI, thus the operator can be deduced from it.

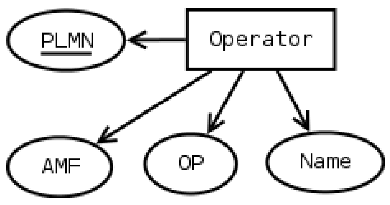


Figure 29: Operator entity with attributes

Authentication Vector It contains the authentication and security parameters produced during the Authentication and Key Agreement (AKA) procedure. These include the HSS level parameters required to derive the called Quintet, the Quintet itself and other derived parameters. A subscriber can handle multiple Authentication Vectors and the Key Set Identifier (KSI) is the parameter to recognize each, thus the natural composite key is the bundle of the IMSI and the KSI.

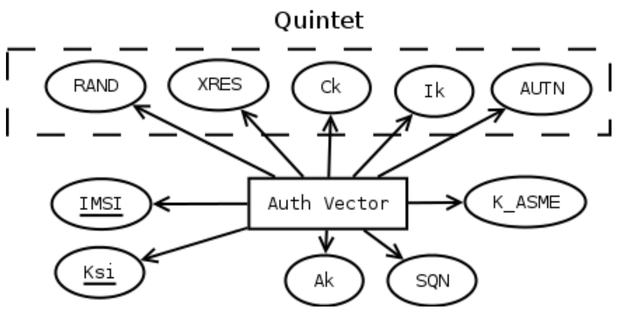


Figure 30: Authentication Vector entity with attributes

PDN Subscription Context This entity contains all the information related with the connection to the Packet Data Network (PDN). It is possible for a single subscriber to maintain simultaneous connections to different PDN, thus the PDN subscription context should be a different entity. The primary key is a composite key using the IMSI and a Context ID.

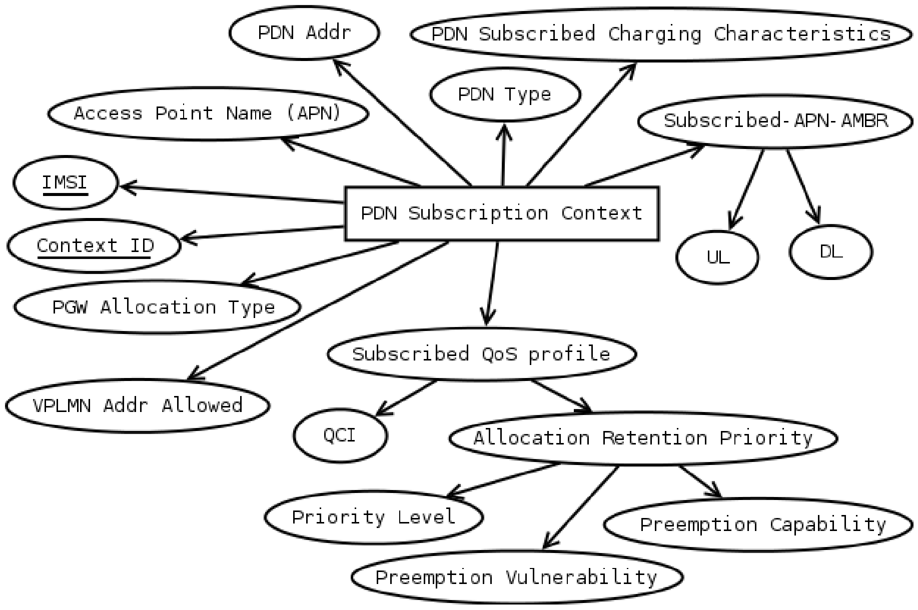


Figure 31: PDN Subscription Context entity with attributes

4.5 Protocol Libraries

This section describes the encoding and decoding functions of GTPv2, S1AP and NAS and explains the API and its singularities.

4.5.1 GTPv2

The GTPv2 encoding and decoding functions have been implemented using Open BSC project libgtp as a starting point. This library is used to implement the version 1 of the GTP protocol for the control and user plane on OpenGGSN project. Due to that fact, it uses structures that are not needed in our project, for example the *struct gsn_t* and *struct pdp_t*. Almost all the original API functions have the *struct gsn_t* as an input parameter to store variables related to GGSN.

To implement the version 2 of GTP-C, only the encoding and decoding functions have been developed, without any logic included nor other variables different from the GTP information contained on the packets.

The key point on this implementation is the definition of the structures used to store the GTP messages. Using struct and union types and bit fields, the buffer received or sent corresponds to the defined type. The packed header can be accessed with a simple cast but the IEs must be decapsulated or encapsulated.

The decapsulation is implemented linking the received message with the available IE types, thus after it, the result is a vector of IE pointers addressing the start of the IE. This implementation introduces a restriction on the use of these IE types, they cannot be used after the message dealocation.

In the encapsulation function, the filled IE structures are copied to the message structure, preparing it to be send as a simple buffer.

Bellow is explained the library use with more detail.

Types

```
struct gtp2ie_tliv
{
  uint8_t  t;                     /* Type */
  uint16_t l;                     /* Length */
  uint8_t  i;                     /* Instance */
  uint8_t  v[GTP2IE_MAX];         /* Value */
} __attribute__((packed));
```

The code above shows the implementation of the typical GTPv2 IE. This structure is combined with all the other possible on a union as seen bellow:

```
union gtpie_member {
  uint8_t  t;                     /* Type */
  /***
  Other GTPv1 IE memeber structs
  ***/
  struct gtp2ie_tliv    tliv;
  struct gtp2ie_tli     tli;
}__attribute__((packed));
```

This allows the developer to access the data on a structured manner, independently of the IE format.

The whole packet structure follows the same idea, bellow is included one of the two different header structures of the GTPv2:

```
struct gtp2_header_long {  /*    Descriptions from 3GPP 29274 */
  uint8_t  flags;          /* 01 bitfield, with typical values */
                           /*    010..... Version: 2 */
                           /*    ...1.... Piggybacking flag (P) */
                           /*    ....1... TEID flag (T) */
                           /*    .....0.. Spare = 0 */
                           /*    ......0. Spare = 0 */
                           /*    .......0 Spare = 0 */
  uint8_t  type;           /* 02 Message type. */
  uint16_t length;         /* 03 Length tei(4)+seq(3)+1(spare)+IE length*/
  uint32_t tei;            /* 05 Tunnel Endpoint ID */
  uint32_t seq : 24 ;      /* 09 Sequence Number (3bytes bit field)*/
  uint8_t  spare1;         /* 12 Spare */
}__attribute__((packed));
```

This struct definition can be compared with the protocol definition shown in Figure 19. We can see how the structure fields correspond to the C definition in a manner that the memory storage format of this type is the same, bit to bit, as the encoded protocol header. Finally, this header is included in a packet structure and a general union type allows the access to different packets, thus the format can be chosen on run time.

```
struct gtp2_packet_long {
  struct gtp2_header_long h;
  uint8_t p[GTP_MAX];
} __attribute__((packed));

union gtp_packet {
  uint8_t                   flags;
  struct flags_t            nflags;
  struct gtp0_packet        gtp0;
  struct gtp1_packet_short  gtp1s;
  struct gtp1_packet_long   gtp1l;
  struct gtp2_packet_short  gtp2s;
  struct gtp2_packet_long   gtp2l;
} __attribute__((packed));
```

It must be noticed how the first union fields correspond to a generic field in order to choose the format to be accessed on run time.

API After explaining the main types used, the API used to encode and decode the GTPv2 messages is explained bellow:

```
unsigned int get_default_gtp(int version, uint8_t type, void *packet);
```

This AP generates a GPRS Tunneling Protocol signalling packet header, depending on GTP version and message type. The packet parameter must be allocated by the calling function and it has to be large enough to hold the packet header, after the return contains the filled header. This function returns the length of the header or 0 on error.

```
int gtp2ie_encaps(union gtpie_member ie[], unsigned int size,
    void *pack, unsigned *len);
```

This function encapsulates the IE that are passed on the packet buffer. The first parameter *ie* contains the IEs to be encapsulated. The second parameter is the number of IEs to be encapsulated. The *pack* and *len* parameters correspond to the buffer and length resulting of the previous function. These parameters are refreshed with the new data. The functions return 0 on success.

```
int gtp2ie_encaps_group(int type, int instance, void *to,
    union gtpie_member ie[], unsigned int size);
```

The purpose of this function is to encapsulate the IEs on a grouped IE, of the type *type* and instance *instance*. The resulting IE is returned on the *to* pointer. The input IE to be encapsulated are passed on the IE vector of containing *size* number of IEs. The function returns 0 on success.

```
int gtpie_decaps(union gtpie_member* ie[], int version,
    void *pack, unsigned len);
```

Function to extract the IE of a received message. The first parameter is a pointer to a vector of *union gtpie_memeber* pointers. The vector must be allocated by the calling function and will be filled with the references to the first byte of each IE. The *version* parameter indicates the protocol version, for GTPv2 use 2. Finally the last parameters are the received buffer and its length. The function returns 0 on success.

```
int gtp2ie_decaps_group(union gtpie_member **ie, unsigned int *size,
    void *from,  unsigned int len);
```

This function decapsulates a grouped IE. The *ie* and *size* parameters contain the outputs. The input parameters are *from* and *len* filled with the grouped IE and its length. The function returns 0 on success.

```
int gtp2ie_gettliv(union gtpie_member* ie[], int type, int instance,
    uint8_t *dst, uint16_t *iesize);
```

Function to get the value of IE structure of type *type* and instance *instance*. The value is copied on *dst* and has *iesize* length. All the IE of the message are on the *ie* vector, this vector has been obtained with the gtpie_decaps function. The function returns 0 on success.

4.5.2 S1AP

This library has been developed from scratch as part of this project. The purpose is to provide the S1-AP protocol structures of the messages and its information elements and the functions to decode and encode these structures.

The S1AP protocol is defined using the abstract syntax notation (ASN.1). The intention of this approach is to allow the developer to compile this definition using an ASN.1 compiler in order to obtain encoding and decoding functions with the desired programming language. The use of an ASN.1 compiler simplifies the development and facilitates the integration of future protocol versions with the current project. Due to the unavailability of open source ASN.1 compilers for C with the required encoding, it was decided to implement all the protocol from scratch.

Without the intended typical ASN.1 implementation approach, the considered alternative is to implement the S1AP protocol manually. This alternative has other advantages, it allows a dedicated implementation, thus the code can be more human readable and compact and some structures can be simplified. In addition, the development of the library requires more time, but it can be partly compensated by the compiler and generated code usage learning time.

The library is implemented on C like all the other components of the project, but it follows an Object Oriented paradigm to allow a modular, flexible and extensible nature. This will simplify the implementation of future protocol versions.

The current S1AP implementation does not include the type constrains defined on the protocol, a void pointer is used to allow the required modularity and the constrains must be controlled by the developer on run time. The error management is also limited on the current implementation, when an error is detected, a message is printed on the logging system and the pointer involved is changed to NULL.

All these exposed limitations can be solved on future library versions, i.e. the type constrain can be solved using the C union type to define the allowed types on a given structure when it is defined. Some examples of implementation details can be found on the comercial ASN1C compiler whitepapers [39] [40].

Types To implement the Object oriented paradigm, almost all the types defined on the library have a constructor function to allocate it. This constructor functions have the *new* word on their name and link the destructor function and an additional function to print the structure on the standard output to a function pointers included in the struct. These function pointers usually include the words *free* and *show* on their names.

The destructor function of a type is responsible for calling the destructor of all dependent types included in the structure. This "waterfall" design allows the dealocation of a message and all the lowest hierarchy types included in it, as the IE types and its Value types, using only the message type destructor.

The most important types are introduced below. They are ordered from high to low hierarchy.

```
typedef struct Message_c{
    uint8_t                     extension;
    enum TriggeringMessage_c    choice;
    S1AP_PDU_t                  *pdu;
    void                        (*freemsg)(struct Message_c*);
    void                        (*showmsg)(struct Message_c*);
}S1AP_Message_t;

S1AP_Message_t *S1AP_newMsg();
```

The above structure defines the highest hierarchy struct on the library, the S1AP message. The choice field with the procedure code on the next PDU struct define the message type, limiting the expected or allowed information elements. The constructor function is also shown. The freemsg function pointer is the destructor function and the showmsg function pointer is the function to print the struct. Both functions have a reference to the object as a parameter.

```
/* This type correspond to InitiatingMessage, SuccessfulOutcome,
UnsuccessfulOutcome types of the standard*/
typedef struct S1AP_PDU_c{
    ProcedureCode_t         procedureCode;
    Criticality_e           criticality;
    uint8_t                 ext;
    ProtocolIE_Container_t  *value;
    void                    *extensionValue;
}S1AP_PDU_t;
```

The S1AP-PDU struct defines the protocol data unit (PDU) of the S2AP. The criticality field defines the actions to be performed if a decoding error is detected. This structure doesn't have any constructor function because it is allocated on the S1AP message constructor.

```
typedef struct ProtocolIE_Container_c {
   uint16_t            size;   /*< Number of IE expected*/
   S1AP_PROTOCOL_IES_t  **elem;
   void     (*freeContainer)(struct ProtocolIE_Container_c*);
   void     (*showIEs)(struct ProtocolIE_Container_c*);
   void     (*addIe)(struct ProtocolIE_Container_c*,
                         S1AP_PROTOCOL_IES_t* ie);
} ProtocolIE_Container_t;

ProtocolIE_Container_t *new_ProtocolIE_Container();
```

The protocolIE-Container is the struct used to store all the information element structures. In addition to the expected constructor, destructor and show functions, a new function is added. The *addIE* is a function pointer to append a new IE struct on the container. The parameters of this function are the object reference and the IE reference. This append function is also present on other IE lists and its usage is common for all of them.

```
typedef S1AP_PROTOCOL_IES_t ProtocolIE_SingleContainer_t;
```

There is a special container of a single IE struct, shown above. It is implemented as an alias of the IE struct, *S1AP_PROTOCOL_IES_t*, explained below.

```
typedef struct S1AP_PROTOCOL_IES_c{
    ProtocolIE_ID_t id;
    Criticality_e   criticality;
    void*           value;
    Presence_e      presence;
    void            (*freeIE)(struct S1AP_PROTOCOL_IES_c *self);
    void            (*freeValue)(void *);
    void            (*showIE)(struct S1AP_PROTOCOL_IES_c *self);
    void            (*showValue)(void *);
}S1AP_PROTOCOL_IES_t;

extern S1AP_PROTOCOL_IES_t * newProtocolIE();
```

The *S1AP_PROTOCOL_IES_t* is the type representing the Information element on the S1AP message. It is composed by the protocol IE Id to identify the IE type, the criticality to instruct the receiver how to act when the IE is not comprehended, the presence to inform about whether the IE is mandatory, conditional or optional and finally the value. The value is where the information is stored, it is a void pointer to allow multiple value types.

On this structure there are other function pointer in addition to the expected ones, *freeIE* and *showIE* refers to the usual destructor and structure print explained before. The other function pointers *freeValue* and *showValue* are the destructor and print functions of the value type. These functions are called on the IE related functions. Using these functions the polymorphism object oriented characteristic is emulated. A better way to emulate the polymorphism characteristic is to use an interface type containing the common fields of the child types, thus used as a parent. The advantage of have the value related methods on this higher hierarchy class is to have the possibility to unlink them as exposed on the *s1ap_setValueOnNewIE* API function description bellow.

```
typedef struct ServedGUMMEIsItem_c{
    void                            (*freeIE)(void *);
    void                            (*showIE)(void *);
    uint8_t                         ext;
    uint8_t                         opt;
    ServedPLMNs_t                   *servedPLMNs;
    ServedGroupIDs_t                *servedGroupIDs;
    ServedMMECs_t                   *servedMMECs;
    ProtocolExtensionContainer_t    *iEext;
}ServedGUMMEIsItem_t;

ServedGUMMEIsItem_t *new_ServedGUMMEIsItem();
```

The last struct is a value type used on the value field of *S1AP_PROTOCOL_IES_t*. All the possible values to be used on *S1AP_PROTOCOL_IES_t* have their own type and *ServedGUMMEIsItem_t* has been chosen to explain the common characteristics.

With the explained polymorphism idea in mind the first fields of the struct are the ones expected to be common on all the value classes, allowing the definition of an interface class. These common fields are the function pointers to the destructor and the show function. Other fields that are similar in almost all the value classes are the *ext* field and the *opt* field. The *ext* is a flag indicating the presence of the protocol extension container. The opt field indicates the presence of the optional fields.

Finally, the rest of the fields are depending on the value type defined on the standard. It is common to nest multiple value types thus the "waterfall design" is really efficient encapsulating the internals of the library, hence simplifying its use.

API In addition to the constructor, other functions are included in this library. The API is explained on this section.

```
extern S1AP_Message_t *s1ap_decode(void* data, uint32_t size);
```

This function decodes a received information and returns an allocated Message structure. The input parameters are *data* with the buffer pointer and *size* with the buffer length. It returns a pointer to the decode message on a *S1AP_Message_t* type. This object should be freed with the *freemsg* function pointer available on the structure after its use.

```
extern void s1ap_encode(uint8_t* data, uint32_t *size,
    S1AP_Message_t *msg);
```

The *s1ap_encode* function encodes the *S1AP_Message_t* type passed as a parameter and fills a previous allocated buffer passed as *data* buffer with the size *size*. The *data* buffer should have enough space for the expected message when the function is called.

```
extern void *s1ap_findIe(S1AP_Message_t *msg, ProtocolIE_ID_t id);
```

In order to find an IE of a received message, the *s1ap_findIE* function can be used. It returns a pointer to the value type of the requested protocol Id *id* if it is present on the *msg* object. The message is not modified with this function, thus after dealocating the message, the IE is not available.

```
extern void *s1ap_getIe(S1AP_Message_t *msg, ProtocolIE_ID_t id);
```

To solve the last problem a variation of the last function is introduced. The *s1ap_getIe* only difference with *s1ap_findIe* is that this function unlinks the Value destructor of the message IE, allowing its storage after the message deallocation. The returned structure should be dealocated after its use with the corresponding function pointer.

```
extern void *s1ap_newIE(S1AP_Message_t *s1msg, ProtocolIE_ID_t id,
    Presence_e p, Criticality_e c);
```

This function is intended to simplify the message creation before the encoding. It allocates the required structures to include a new IE in the message and link the necessary methods of the destructor and print functions. It returns a pointer to the allocated value type depending on the IE identification passed using the *id* parameter. The function allocates the IE and its value objects and integrates them on the passed S1AP message *s1msg*. The rest of the parameters are required to fill the IE object during its creation.

This function is the recommended way to build a S1AP message. This function only works to allocate first level IE values and does not work for nested IE values or IE groups. These cases have to be built manually. Allocating the required resources and linking the function pointer fields of the type. On future implementations, new API functions could be developed to solve these use cases.

```
extern void s1ap_setValueOnNewIE(S1AP_Message_t *s1msg,
    ProtocolIE_ID_t id, Presence_e p, Criticality_e c,
    GenericVal_t *val);
```

The *s1ap_setValueOnNewIE* function is very similar to *s1ap_newIE* function with the difference that the value object is not allocated in the function but passed as a parameter. In addition the destructor of the value object is not linked to the IE object. This allows the inclusion of an existing value on the message and the possibility of maintaining it after the destruction of the message. This function could be used to include certain long term values on a message to encode it without allocating and copying the value on the message. The value reference on this function is passed using an interface class commented before, *GenericVal_t*, using polymorphism instead of a void pointer.

4.5.3 NAS

As the S1AP library, the NAS library has been developed from scratch as part of this project. The library provides message structures and IE types for decoding and encoding purposes as well as functions to create new packets or parse the received ones.

The library is implemented using the C programming language. Only the used procedures are implemented but the simple design allows the addition of new procedures really quick as the basic structures are implemented.

Following are the library details. All these definitions were extracted from the source files include in the shared folder of the NAS library.

Types The defined structures are common for EMM and ESM messages when it is possible. The high level structures are even prepared to implement the security protected messages.

```
typedef union GenericNASMsg_c{
    NAS_Header_t            header;
    NASPlainMsg_t           plain;
    SecurityProtectedMsg_t  ciphered;
}GenericNASMsg_t;
```

The *GenericNASMsg_t* is the high hierarchy type of this library. It is the returned structure in the decoding function. It is a union prepared to contain both a plain message or a ciphered message. The header field is intended to contain the common header in order to identify whether the message is plain or ciphered.

```
typedef struct NAS_Header_c{
    ie_v_t1_l_t protocolDiscriminator;
    ie_v_t1_h_t securityHeaderType;
}NAS_Header_t;
```

The common header implementation can be observed above. It contains the Protocol Discriminator to inform if the message is either EMM type or ESM type. The second field is the security header indicating whether the message is plain or ciphered as commented. The types of this struct field are IE types. The Header is formed with the same IE structures.

```
typedef struct SecurityProtectedMsg_c{
    ie_v_t1_l_t     protocolDiscriminator;
    ie_v_t1_h_t     securityHeaderType;
    uint8_t         procedureTransactionIdentity;
    ie_v_t3_t       messageAuthCode;
    uint8_t         sequenceNum;
    NASPlainMsg_t   msg;
}SecurityProtectedMsg_t;
```

The *SecurityProtectedMsg_t* contains the field for a security enabled message. This structure is not currently used because the security messages are not implemented.

```
typedef union NASPlainMsg_c{
    EMM_Message_t eMM;
    ESM_Message_t eSM;
}NASPlainMsg_t;
```

The *NASPlainMsg_t* type is the first to separate the EMM and ESM messages. It is a union that should be read/written according to the protocol Discrimination Field.

```
typedef struct EMM_PlainMessage_c{
    ie_v_t1_l_t         protocolDiscriminator;
    ie_v_t1_h_t         securityHeaderType;
    ie_t_t2_t           messageType;
    union nAS_ie_member *ie[30];
}EMM_Message_t;

/** ESM Plain message */
typedef struct ESM_PlainMessage_c{
    ie_v_t1_l_t         protocolDiscriminator;
    ie_v_t1_h_t         securityHeaderType;
    uint8_t             procedureTransactionIdentity;
    ie_t_t2_t           messageType;
    union nAS_ie_member ie[30];
}ESM_Message_t;
```

The *EMM_Message_t* and *ESM_Message_t* structures are generic protocol structures implementing the specific protocol header. The IE types are used to allow the cast mechanism with lower hierarchy structures defining a message.

```
typedef struct AttachComplete_c{
    ie_v_t1_l_t protocolDiscriminator;
    ie_v_t1_h_t securityHeaderType;
    ie_t_t2_t messageType;
    ie_lv_t6_t eSM_MessageContainer;
}AttachComplete_t;
```

The *AttachComplete_t* is an example of an Attach Complete message structure. On specific messages, all the mandatory fields are stored on the structure, thus no allocation or deallocation is required.

```
union nAS_ie_member{
    uint8_t     iei;
    ie_v_t1_l_t v_t1_l;
    ie_v_t1_h_t v_t1_h;
    ie_tv_t1_t tv_t1;
    ie_t_t2_t t_t2;
    ie_v_t3_t v_t3;
    ie_tv_t3_t tv_t3;
    ie_lv_t4_t lv_t4;
    ie_tlv_t4_t tlv_t4;
    ie_lv_t6_t lv_t6;
    ie_tlv_t6_t tlv_t6;
    }__attribute__((packed));
```

Finally, the *union nAS_ie_member* is a generic IE union formed by all the IE possible types to allow IE modularity. The *ie_tlv_t6_t* type is included following as an example.

```
typedef struct ie_tlv_t6_c{
uint8_t  t;
uint16_t  l;
uint8_t  v[NASIE_MAX16];
}__attribute__((packed)) ie_tlv_t6_t;
```

In addition to these types a group of enumerated types are included to simplify the code readability, *SecurityHeaderType_t*, *ProtocolDiscriminator_t*, *Procedure-TransactionId_t*, *NASMessageType_t*, *ESMCause_t*, *EMMCause_t.*

API The following functions are used to decode the NAS messages, either ESM or EMM.

```
void dec_NAS(GenericNASMsg_t *msg, uint8_t *buf, uint32_t size);
```

This function fills the *msg* struct pointer with the message information. The encoded message is passed using the *buf* pointer and the length is passed with the *size* parameter.

The *dec_NAS* function uses decoding functions for each message internally. The *dec_AttachComplete* is the corresponding for the Attach Complete message.

```
void dec_AttachComplete(AttachComplete_t *msg, uint8_t *buffer,
     uint32_t size);
```

The following functions are used to build a NAS message. These functions require a previous allocated buffer. The buffer is used to store the encoded NAS message and it is passed to the functions as a double pointer in order to modify it. The buffer pointer acts as an index and after the encoding function execution it points to the next position to be written on the buffer. This design allows the functions to be called sequentially with the same pointer variable to aggregate new IE structures to the message. This approach is possible because all the NAS fields are encoded as IE types, even the header ones, thus all the encoding functions are based on the IE encoding functions.

```
void newNASMsg_EMM(uint8_t **curpos,
    ProtocolDiscriminator_t protocolDiscriminator,
    SecurityHeaderType_t securityHeaderType);

void encaps_EMM(uint8_t **curpos, NASMessageType_t messageType);
```

The *newNASMsg_EMM* encodes the header parameters of the EMM message on the buffer. The buffer pointer is the *curpos* parameter. The other parameters are the header fiels of the EMMS. The EMM message encoding continues with the *encaps_EMM* function to add the rest of the EMM header. These two functions are not combined to allow a future security message encoding function.

```
void newNASMsg_ESM(uint8_t **curpos,
    ProtocolDiscriminator_t protocolDiscriminator,
    uint8_t ePSBearerId);

void encaps_ESM(uint8_t **curpos,
    ProcedureTransactionId_t procedureTransactionIdentity,
    NASMessageType_t messageType);
```

The *newNASMsg_ESM* and *encaps_ESM* are the equivalent function of *newNASMsg_EMM* and *encaps_EMM* for ESM messages.

All the IE types have an encoding function following the explained mechanism. The *nasIe_tlv_t6* function is included as example.

```
void nasIe_tlv_t6(uint8_t** p, uint8_t t, uint8_t *v, uint16_t len);
```

4.6 MMEcmd

The MMEcmd tool is used to send commands to the MME remotely during its execution. MMEcmd is intended to be an Operation and Management platform to control the MME but the current functionality is limited as no use cases have been defined yet. Currently, this tool can send signals to the MME engine. It has been used to shutdown the MME application and to perform some basic functionally tests during the development.

4.7 Metrics and testing

At the early stage of the project it was developed a methodology to track and expose the project evolution through different code metrics. The aim of the metrics is to measure the quality of the code and detect possible issues as early as possible on the development. The metrics are automatically triggered and shown on a continuous integration server, Jenkins [47].

The code changes are uploaded to a control version software and the Jenkins server polls it every night to detect changes in the code. The control version server used is subversion. When new code is detected all the metrics are recomputed and the metric changes over time are depicted on some graphics to have a project status overview.

The following are the static metrics performed directly on the source code:

- Source lines of code (SLOC): The simplest metric is the number of source lines of code on the project. This metric is important because a new added functionality can be related with the added lines of code. This allows the estimation of the effort to implement another similar functionality. In addition, if the project or some module increases its size too much it may indicate the need for a refactory or its break into new modules. This metric can be used as input for some cost models as COCOMO [35].

- Duplicated Code: It is a copy paste detector. It can even identify similar code regions. If big sections detected, it means the code can be included in a new function, thus increasing modularity.
- Cyclomatic complexity (CC): This metric counts the number of executable paths on the application. Usually counting the number of if statements. Some studies have proven a correlation between the number of bugs and the high cyclomatic complexity.
- CppCheck: this tool detects errors that usually are not detected by the compiler such as memory leaks.

In addition to these static metrics, a testing framework is used. The tests can be considered dynamic metrics because they measure the application functionality executing a code fragment. The idea is to develop using unit tests defining a function behavior and then implement the function in order to pass the test. This approach is used on some new developing methodologies as extreme programming because allow a quick implementation, an easy refactoring of the code and a method of detecting a bug on a recent source code change. The framework used for this purpose is the Check Testing Framework [48]. The last metric is the test coverage to check the percentage of the code that is being tracked with the tests.

The tests structure hierarchy is the following. The unit tests check a functionality. The unit tests are grouped on a test case to check bigger entities. Finally the test cases are grouped on a test suit to test a whole module. The MME project has a test suite for each library, but the basic PER encoding and decoding functions tests are included in a separated suite because they are considered critical.

4.8 Conclusions

This chapter describes the most important aspects of the MME implementation including design ideas and strategies. The developed application requires future testing to be mature enough to be used on production environments.

The encoding and decoding protocol libraries can also be improved but they are mature enough to be used on other LTE node implementation projects due to its modular design.

A possible bottle neck on the current implementation is the data base implementing the HSS. A real MME connects to an optimized HSS thus if the DB becomes a MME bottle neck it can be considered an emulation issue and its criticality reduced.

5 Testbed performance results

This chapter shows the archived testbed and includes some measurements to prove the compliance with the standard requirements. Section 5.1 describes the actual testbed. Section 5.2 contains the overload measurements and finally the sections 5.3 and 5.4 contain latency measurements on the testbed, the first on unload conditions and the second on loaded conditions.

Almost all the measurements have been taken using the attach procedure as an indication of the general performance

5.1 TestBed Setup

The current testbed is depicted in Figure 32. The blade servers are virtual servers located on the university datacenter. These servers are running on a Xen server [42] using a full virtualised environment.

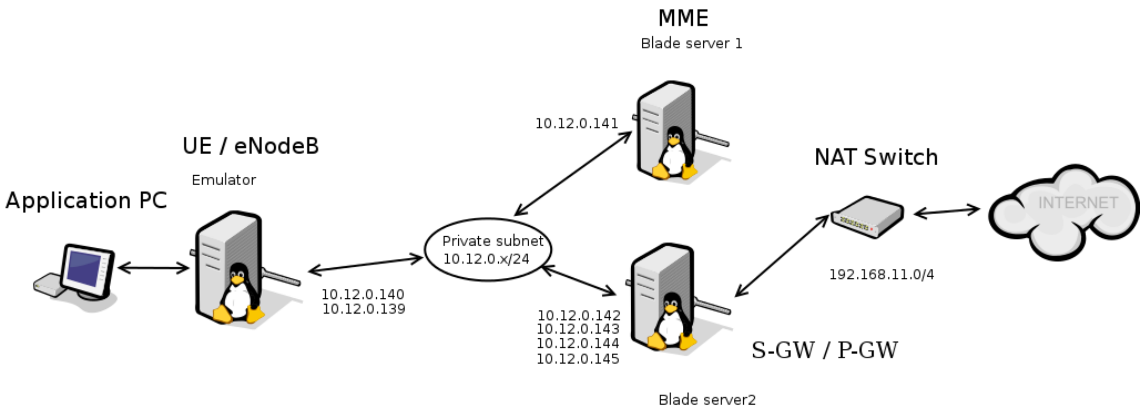


Figure 32: TestBed diagram

The left part of the figure is the host with the third party emulator application. This application emulates the Radio Access network on the testbed and provides a connection to an application PC to generate the User traffic to be tunneled. The Emulator host has two network interfaces (NIC), the first used to connect the emulator to the EPC and the second to connect with the application PC. The emulator runs on a Fedora 17.

The emulator is connected to the blade servers using a VLAN with private IP addresses (10.12.0.0/4). This LAN transports the user data and the signaling, thus it implements the S11, the S1-MME and the S1-U interfaces on the same network.

The Blade server number 1 executes the developed MME application. It only requires one NIC because the S11 and S1-MME interfaces are separated using different protocols and ports. The ideal LTE setup is with 2 different NICs. It runs an Ubuntu 12.04.2 LTS 64 bit version although the MME application has been tested on a Fedora 18 32 bits version. The application is expected to work on other UNIX environments.

Finally, the S-GW and P-GW functionality is performed by an open source SAE-GW application [41] on the second blade server. This blade server has a connection to the common private network 10.12.0.0/4 and another NIC with internet connectivity through a NAT. The application SAE-GW application requires an IP address for each LTE interface and IP alias have been used for this purpose. The used IP addresses are the following:

10.12.0.142	Used on the S-GW's S1-U interface.
10.12.0.143	Used on the S-GW's S11 interface.
10.12.0.144	Used on the S-GW's S5 interface.
10.12.0.145	Used on the P-GW's S5 interface.

During the testing process, some bugs have been detected and corrected on the SAE-GW.

The SAE-GW uses the NAT interface to offer the PDN connectivity to the UE. The SAE-GW detects UE assigned IP addresses on the received packets in order to forward them to corresponding the UE GTP tunnel. Because the lack of a public IP range to assign to the incoming UE a combination of private IP addresses and a NAT is used to serve the UE. This solution simplifies the UE packets routing configuration and offers a whole subnet IP range to allocate the IP addresses for the UE pool.

The problem of the NAT approach is that the SAE-GW host does not respond the UE IP address ARP request from the Switch thus the packets with UE destination are not sent to the SAE-GW host. To solve this problem ARP spoofing techniques are used to poison the switch ARP table and force the UE destination packets to be forwarded to the host.

5.2 Overhead

User data In order to study the protocol overhead on the S1-U interface, the different protocol header sizes should be considered. Table 2 shows the size of the message header per protocol. The considered overhead is the tunneling overhead thus the transported layers above the GTP protocol are considered as the payload.

Protocol	Header (B)
ethernet	16
IP	20
UDP	8
GTP	9
TOTAL	52

Table 2: Header Contribution per Protocol on S1-U

For each IP packet Table 2 shows that 52 bytes would be overhead. In this test case, we consider a video stream over TCP. The captured TCP traffic from the video stream includes the data packets from the PDN to the UE and the ACK packet back from UE to PDN.

The ACK packet has an overhead of 56,52% as detailed on the first row of Table 3. This high overload is the expected one due to the small size of the packet.

Message	Packet	Payload(B)	Header(B)	Overhead(%)
ACK	92	40	52	56,52
Data	1552	1500	52	3,35

Table 3: S1-U Overhead on data

Another inefficiency of using GTP-U happens when user data has to be fragmented. The UE application is sending IP packets with 1500 byte MTU which is the Linux default MTU. These data packets have to be tunneled with GTP header thus resulting in packets with 1552 bytes. These packets do not fit any more in a MTU of 1500 bytes and have to be fragmented.

Message	Packet	Payload(B)	Header(B)	Overhead(%)
Fragment 1	1516	1464	52	3,43
Fragment 2	72	36	36	50,00

Table 4: S1-U Overhead with fragmentation

This results in two IP fragments of 1516 bytes and 72 bytes with the overheads of 30,43% and 50% respectively as shown in Table 4.

The total overhead of the ACK packets together with the fragmented IP packets results on an approximate overhead of 8%. An overhead of 8% is a good result although it depends on the type of traffic the tunnel is transporting.

Signalling We consider signaling overhead all the messages required to establish and manage the data bearers. The transactions going through S1-MME, S11 and S5/S8 interfaces are signaling procedures. Moreover, each of these interfaces have their own overhead because S1-MME runs directly over IP/SCTP but S11 and S5/S8 run over GTPv2.

Table 5 contains the S1AP messages for the attach procedure and their overhead. The cause of the large overhead is the small size of the signalling messages. These small packets can become problematic as they may reduce the throughput or increase the network latency when applying Nagle's algorithm.

Tables 6 and 7 contain the messages for GTPv2 traffic of interfaces S11 and S5 respectively and their overhead.

Finally, in Table 8 we can observe the total signalling overhead from S1-MME, S11 and S5/S8.

	Packet(B)	Payload(B)	Header(B)	Overhead(%)
Initial UE Message	130	66	64	49.23
Authentication Request	138	60	78	56.52
Initial Context Setup Request	234	156	78	33.33
Initial Context Setup Response	118	38	80	67.80
Attach Complete	94	31	63	67.02
SACK	62	0	62	100.00
TOTAL S1AP	890	386	504	56.63

Table 5: Attach S1AP overhead

	Packet(B)	Payload(B)	Header(B)	Overhead(%)
Create Session Request	204	162	42	20.59
Create Session Response	141	99	42	29.79
TOTAL S11	345	261	84	24.35

Table 6: Attach S11 overhead

	Packet(B)	Payload(B)	Header(B)	Overhead(%)
Create Session Request	211	169	42	19.91
Create Session Response	119	77	42	35.29
TOTAL S5	330	246	84	25.45

Table 7: Attach S5 overhead

	Packet(B)	Payload(B)	Header(B)	Overhead(%)
TOTAL S1AP	890	386	504	56.63
TOTAL S11	345	261	84	24.35
TOTAL S5	330	246	84	25.45
TOTAL Attach	1565	893	672	42.94

Table 8: Attach Total overhead

Overhead conclusions The results show the signaling overhead 42,94% is bigger than data overhead (aprox 8%) but signalling impact is negligible on the overall data transfer between UE and PDN. Therefore focus should be on reducing the data overhead once the signalling requirements (latency, etc.) are met.

5.3 Latency on unload conditions

Usually, the latency is measured using network tools as ping but this approach is not applicable on the current setup because there are multiple elemental procedures involved.

The chosen methodology to measure the latency is to split the different parts of the procedure and calculate the elemental transfer delays. These delays are the time stamp difference between messages captured with the network sniffer wireshark. The latency of the EPC can be calculated as the addition of transfer delays between a message with the emulator as source and a message with the EPC as source.

The transfer delay is composed by the transmission delay, the propagation delay, the queue delay and the processing delay. Due to the short distance of the nodes, the propagation delay is not significant. In addition, the transmission delay is not significant too, because of signalling packet's short length and the high capacity link. On unloaded conditions, the queue time is not considered thus the only component of interest is the processing time.

This assumption has been checked using time stamps on the code to measure the processing time and it is equivalent to the delays observed on the capture application.

On this section, the latency results are exposed and commented. The intermediate data has been included only in the first procedure, the attach procedure, and only the final results are included on the other procedures.

Attach Procedure According to the requirements detailed on the standards [18], the maximum allowed C-Plane latency on LTE-A is 50ms. This latency takes into account the core network latency and the radio access latency and it is defined as the transition time from Idle mode to connected mode excluding the S1 transfer delay.

In Table 9 can be observed the relative time stamps of the attach procedure messages captured on the emulator host using the traffic capture application wireshark.

Attach Req	Auth Req	Auth Resp	Attach Accept	Attach Complete
0	0.000776	0.120139	0.126351	0.155557
0	0.000752	0.118305	0.122051	0.154766
0	0.000740	0.119506	0.121718	0.249578
0	0.000763	0.118895	0.126644	0.155799
0	0.000815	0.119042	0.123016	0.150096
0	0.000798	0.118633	0.121944	0.151174

Table 9: Attach time stamps, expressed on seconds

The resulting transfer delays are exposed in Table 10.

As it can be observed, the transfer time does not comply with the required standards for LTE-A. However, when observing the contributions of the EPC and the RAN emulator, it can be noticed an important difference: the major contribution is the emulator.

t1	t2	t3	t4	EPC	Emulator	TOTAL
0.776	119.363	6.212	29.206	6.988	148.569	155.557
0.752	117.553	3.746	32.715	4.498	150.268	154.766
0.740	118.766	2.212	127.86	2.952	246.626	249.578
0.763	118.132	7.749	29.155	8.512	147.287	155.799
0.815	118.227	3.974	27.080	4.789	145.307	150.096
0.798	117.835	3.311	29.203	4.109	147.065	151.174

Table 10: Processing time, expressed on milliseconds

With these results it can be concluded the emulator does not fit the signalling latency requirements. However, the EPC latency is small enough to be combined with a real RAN if a latency lower than 47ms is granted to fit the LTE-A requirements.

Detach Procedure The detach procedure measured is the eNB initiated variation. It is defined as the transition from Active mode to Idle mode, this it is the transfer delay of NAS detach procedure. The obtained result with an average of 6 measurements is a latency of 1,40 ms, thus the detach procedure is compliant with the 50ms latency requirement.

During the detach procedure measurements, it has been noticed a large delay on the EPC to start the UE context Release procedure. This delay is larger than the 50ms and can be problematic on a high load scenario. The origin of this delay is a performance problem caused by interaction between Nagle's algorithm and the Delayed Ack [2]. This issue has been solved disabling Nagle's algorithm with the SCTP_NODELAY socket option but it needs further investigation on high load cases as disabling Nagle's algorithm may increase network congestion.

X2 based handover The X2 based handover procedure latency is defined from the first X2-AP message exchanged between the eNodeBs until the Path switch accept message. The Tracking area update procedure is not included as does not modify any configuration, it only communicates the UE location.

The latency results are really high, more than 2 seconds, but observing the EPC and RAN contributions, the same conclusions as the Attach procedure can be derived. The EPC contribution to the latency is 1,36 ms with four measurements average.

The emulator contribution to the latency cannot be studied because the source code is not available and it uses a closed SCTP implementation.

S1 based handover The latency on the S1 based handover is defined from the eNB message to the MME informing a handover is required until the MME communicates to the eNB that the uplink tunnel is available with the Handover Command.

[2]The detailed problem description can be found on this article [38]. The article describes the problem on TCP but it applies to SCTP too.

This latency definition is more appropriate because the handover notify depends on the UE movement, thus the measure is more objective and stable. The other messages are performed once the UE is changing eNodeB or has already changed, thus are not considered on this latency measurement.

The results of this procedure show the usual difference between the EPC and the RAN contribution, but the total is compliant with the 50 ms limit. The EPC latency is 1,21 ms, the emulator latency is 34,01 ms, hence the total latency is 35,22 ms (< 50ms).

5.4 Latency on load conditions

In order to study the testbed response on load conditions, the loadWithAttach application has been developed. This application is included in the exampleProgram folder as it illustrates the S1AP library use. The application acts as an eNodeB and triggers the attach of dummy UEs. These continuous attach procedures load the testbed EPC allowing its characterization.

The loadWithAttach application inserts all the subscriber information required by the MME for a successful attach procedure in the DB. Two different DB load strategies are considered, 1) insert the subscriber information before initiating each attach, 2) insert the subscriber information before any attach in application's start (-p option). A third option is considered (-r option) that allows a reset of the database after a certain time after the initialization of the execution.

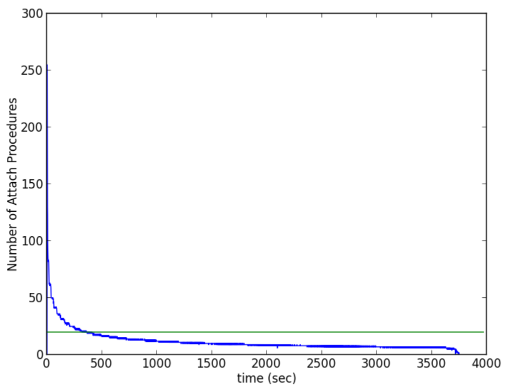


Figure 33: Number of Attachments per second

The results for the first strategy are depicted in Figure 33. The horizontal line on 20 attachments per second corresponds to the latency limit on unload conditions. This 20 attachments per second is a reference to compare loaded and unloaded measurements. The figure shows a decreasing latency due to the UE data additions to the system. The point where the measurements reach the 20 attachments per second corresponds to approximate 12000 subscriber contexts on the system.

The end of the attachments at the right of the figure corresponds to the testbed break. The S/P-GW application stops without any error message.

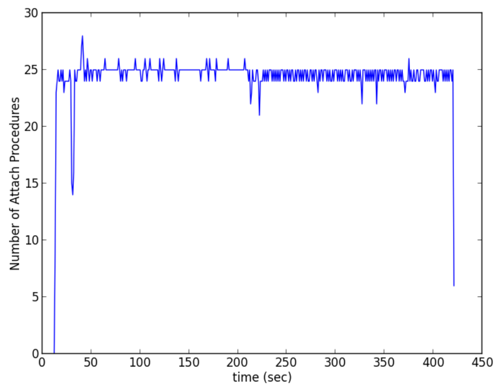


Figure 34: Number of Attachments per second with 10000 UE preload

The second strategy allows the preload of the subscriber information on the HSS emulated database, but not on the MME. With this test it is possible to discriminate whether the bottle neck is on the MME UE storage or on the emulated HSS implemented with a MySQL database. The results are depicted in Figure 34.

The attachments per second with a UE data preload of 10000 subscription context on the HSS are almost constant on 25 att/sec. On a real scenario, the subscriber information is preloaded on the HSS by the operators to manage the subscribers.

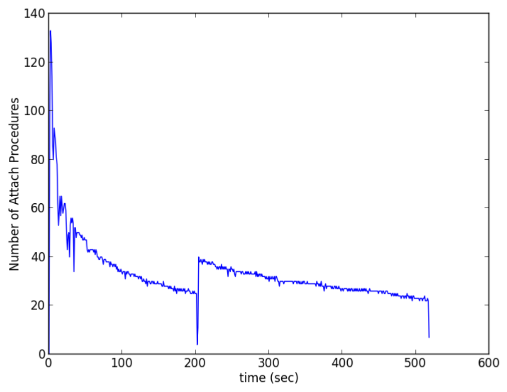


Figure 35: Number of Attachments per second with UE data base reset on 200s

Figure 35 depicts the measurement results applying the database reset variation on 200 seconds after the initialization of the test. The database reset is obvious in the discontinuity upon 200 seconds. Although the reset peak is not as large as it was expected. At the begin of every measurement, the database is rebuilt and without this step the initial peaks are reduced on the following tests.

The decrease of the unloaded database peak may be related with a missing database configuration optimization for extensive reading / reading use cases.

5.5 Conclusions

This chapter has described the EPC testbed developed and contains some measurements to characterize its performance according to the standard's requirements.

With these results it can be stated the standard's requirements are accomplished for unloaded conditions. There are no requirements for loaded conditions but the EPC is able to support at least 12000 subscribers with the current configuration maintaining the same condition requirements included in 3GPP standards.

This number of subscribers is not enough for large deployment where the MME manages multiple Tacking Areas but the measurements show the bottle neck is on the HSS emulator data base, thus the solution should be easily scalable with an optimized database or implementing the S6a interface and use real HSS infrastructure.

Another aspect to pay attention on future changes is the high inefficiency of S1AP signalling protocol as it can produce throughput and latency issues due to the small size of the packets. In special, the delayed SACK in combination with Nagle's algorithm may have a dramatic impact on the latency, increasing it four times.

The goal of implementing a virtualised EPC platform to provide an starting point for future LTE studies has been reached successfully.

6 Conclusions and future work

6.1 Conclusions

The Goal of the project was to build a virtualised LTE mobile core network (EPC) testbed to study its feasibility and provide a platform to implement future modifications on LTE. The major part of the project has been focused on implementing a MME application because it was the only pure LTE core node that was not implemented as an open source solution.

The developed MME is not prepared for production environment yet as it does not implement all the optional procedures detailed on the standards, but it implements the successful cases of the procedures described in Section 3.5, the Attach, the Detach, the X2-based Handover and S1-based Handover. With these procedures, the basic functionality to deploy basic UE connectivity is implemented. In addition, the available S/P-GW source and the third party RAN emulator do not support all the Handover related procedures. The final testbed is a valuable tool for studying and developing new LTE possibilities on virtual environments.

The measurements of the EPC testbed show it is possible to virtualise the core network functions while maintaining the expected performance required by 3GPP standard. The testbed measurements expose a set of performance boundaries on high load conditions and aspects to be observed on future tests. Nagle's algorithm and SACK delay combinations has been solved disabling Nagle's algorithm on the S1AP-MME interface. Future studies should take attention to them to avoid latency issues. The other limitation found is related with the emulated HSS database, the proposed future solution is to use a real HSS or to optimize the current MySQL database for intensive reading and writing.

6.2 Future work

The integration of LTE virtualisation with Software Defined Networking (SDN) concepts is an attractive future scenario. The preferred definition of SDN is the Scott Shenker's view that defines *"SDN by the abstractions it provides to software (and people writing it)"*. The idea is to define network abstractions to simplify the development and operation of networks. These concepts are currently implemented with a SDN controller that manages a set of switches.

The virtualisation offers freedom of deployment on standardized and open environments while SDN offers a central view of the network and the possibility to program it by external applications. These technologies are complementary and have the potential to transform the current network architecture and prepare it for the mobile traffic explosion. An LTE network with SDN integration could deploy mobility, resource management and virtualisation using a shared infrastructure to multiple operators.

There are two possible approaches to integrate the LTE core network, Evolved Packet Core (EPC) with SDN. The SDN controller can be integrated within the MME to be aware of the mobility requirements or can be located as part as the

S/P-GW to control the transport network. The integration of the SDN controller on the MME offers the advantage to have access to the control plane of the LTE architecture. In addition, this access may facilitate a seamless migration to a mobile network with SDN integration or Software Defined Mobile Network (SDMN). The increase of complexity in the MME with the SDN controller will decrease the overall complexity of the LTE network by eliminating some other network elements such as the P/S-GW that becomes obsolete with a switch mesh network managed by the SDN controller.

The SDN / LTE virtualisation integration challenge relies on combining the SDN flows with the EPS bearers. The design decisions taken in order to combine these two concepts will affect the network topology and the interfaces, including its protocol stacks.

The EPS bearer management is performed by the MME which has the access to the location of the UE and manages the mobility. Proposed deployment is to integrate the SDN controller on the MME because the MME has the global network view required by the SDN controller. On large deployments with multiple MMEs that SDN controller should be transformed on a slave controller subordinated to a master controller. This scenario will require new technologies to maintain state consistency between distributed controllers in order to maintain a logically centralized controller.

Another important design decision is whether the GTP tunnels are maintained or can be removed. If the GTP tunnels are maintained on the EPS bearer, the solution deployment should be easily because no change is required on the eNodeBs but a P-GW endpoint should be maintained in order to decapsulate the UE tunnel. However, if the GTP tunnels are dismissed, the eNodeBs should be changed to avoid the tunneling but another SDN mechanism should be defined to manage the UE flows. With the last solution, the PDN is taken up to the eNodeBs and it might be more flexible depending on how the UE flows are designed.

References

[1] 3GPP TS 23.002: *“Network Architecture”.*

[2] 3GPP TS 23.203: *“Policy Control and Charging Architecture (Stage 2)”.*

[3] 3GPP TS 23.401: *“General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access”.*

[4] 3GPP TS 23.402: *“Architecture enhancements for non-3GPP accesses”.*

[5] 3GPP TS 24.007: *“Mobile radio interface signalling layer 3; General aspects”.*

[6] 3GPP TS 24.301: *“Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3”.*

[7] 3GPP TS 29.272: *“Evolved Packet System (EPS); Mobility Management Entity (MME) and Serving GPRS Support Node (SGSN) related interfaces based on Diameter protocol”.*

[8] 3GPP TS 29.274: *“General Packet Radio Service (GPRS); Evolved GPRS Tunnelling Protocol (eGTP) for EPS”.*

[9] 3GPP TS 29.281: *“General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)”.*

[10] 3GPP TS 33.401: *“3GPP System Architecture Evolution (SAE); Security architecture”.*

[11] 3GPP TS 36.300: *“Evolved Universal Terrestrial Radio Access (E-UTRA), Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; stage 2”.*

[12] 3GPP TS 36.401: *“Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Architecture description”.*

[13] 3GPP TS 36.410: *“S1 General Aspects and Principles”.*

[14] 3GPP TS 36.411: *“Evolved Universal Terrestrial Radio Access Network (E-UTRAN); S1 layer 1”.*

[15] 3GPP TS 36.412: *“Evolved Universal Terrestrial Radio Access Network (E-UTRAN); S1 signalling transport”.*

[16] 3GPP TS 36.413: *“Evolved Universal Terrestrial Radio Access Network (E-UTRAN); S1 Application Protocol (S1AP)”.*

[17] 3GPP TS 36.414: *“Evolved Universal Terrestrial Radio Access Network (E-UTRAN); S1 data transport”.*

[18] 3GPP TS 36.913: *"LTE; Requirements for further advancements for Evolved Universal Terrestrial Radio Access (E-UTRA) (LTE-Advanced)"*.

[19] ITU-T Recommendation X.680 (07/2002): *"Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation"*.

[20] ITU-T Recommendation X.681 (07/2002): *"Information technology - Abstract Syntax Notation One (ASN.1): Information object specification"*.

[21] ITU-T Recommendation X.691 (07/2002): *"Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)"*.

[22] R. Stewart, Ed., *Stream Control Transmission Protocol.* Internet Engineering Task Force, Request for Comments 4026, September, 2007

[23] L. Andersson, T. Madsen, *Provider Provisioned Virtual Private Network (VPN) Terminology.* Internet Engineering Task Force, Request for Comments 2547, March, 2005

[24] E. Rosen, Y. Rekhter, *BGP/MPLS VPNs.* Internet Engineering Task Force, Request for Comments 4960, March, 1999

[25] IEEE Std. 802.1Q-2011, *Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks*

[26] R. J. Creasy, *"The origin of the VM/370 time-sharing system"*, IBM Journal of Research & Development, Vol. 25, No. 5, pp. 483-90, September 1981

[27] Melinda Varian, *"VM and the VM community, past present, and future"*, SHARE 89 Sessions 9059-61, 1997

[28] FOX, Armando, et al. *Above the clouds: A Berkeley view of cloud computing.* Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, 2009, 28.

[29] IRTF Virtual Networks Research Group (VNRG) `http://irtf.org/concluded/vnrg`

[30] *Network Functions Virtualisation - Introductory White Paper* ETSI. 22 October 2012. Retrieved 20 June 2013.

[31] Chowdhury, NM Mosharaf Kabir, and Raouf Boutaba. *"Network virtualization: state of the art and research challenges."* Communications Magazine, IEEE 47.7 (2009): 20-26.

[32] Raj Jain, *Virtual LANs*, Class Presentations, 1997, `http://www.cse.wustl.edu/~jain/cis788-97/h_7vlan.htm`

[33] N. Feamster, J. Rexford, E. Zegura, *The Road to SDN: An Intellectual History of Programmable Networks*

[34] J. Touch, *The X-Bone*, NGI Workshop White Paper, March, 1997

[35] B.W. Boehm, *Software Engineering Economics.* Prentice Hall, 1981

[36] Tieto, Intel *Carrier Cloud Telecoms - Exploring the Challenges of Deploying Virtualisation and SDN in Telecoms Networks*

[37] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update*, 2012-2017. White Paper, Feb 6, 2013.

[38] Cheshire, S. *TCP Performance problems caused by interaction between Nagle's Algorithm and Delayed ACK*, May 2005

[39] *"ASN1C C/C++ Code Generation for 3GPP and LTE Specifications Objective"*, Objective Systems, Inc., December 2008

[40] *"ASN1C Support for Information Objects and Parameterized Types"*, Objective Systems, Inc., April 2002

[41] Chawre, A. *Evolved Packet Core SAE Gateway* `http://www.amitchawre.net/nw-epc.html`

[42] Xen Server website: `http://www.xenserver.org`

[43] *Osmocom OpenBSC* `http://openbsc.osmocom.org`

[44] Mathewson, N., Provos, N. *Libevent* `http://libevent.org/`

[45] Kegel, D. *The C10k problem*, `http://www.kegel.com/c10k.html`

[46] *MariaDB* `https://mariadb.org/`

[47] *Jenkins Server webpage* `http://jenkins-ci.org/`

[48] *Check Testing Framework* `http://check.sourceforge.net/`

[49] OpenStack Cloud software `http://www.openstack.org/`

[50] Chapple, M. *Database Normalization Basics*, About.com Guide Article. `http://databases.about.com/od/specificproducts/a/normalization.htm`

A HSS Database

A.1 Database structure

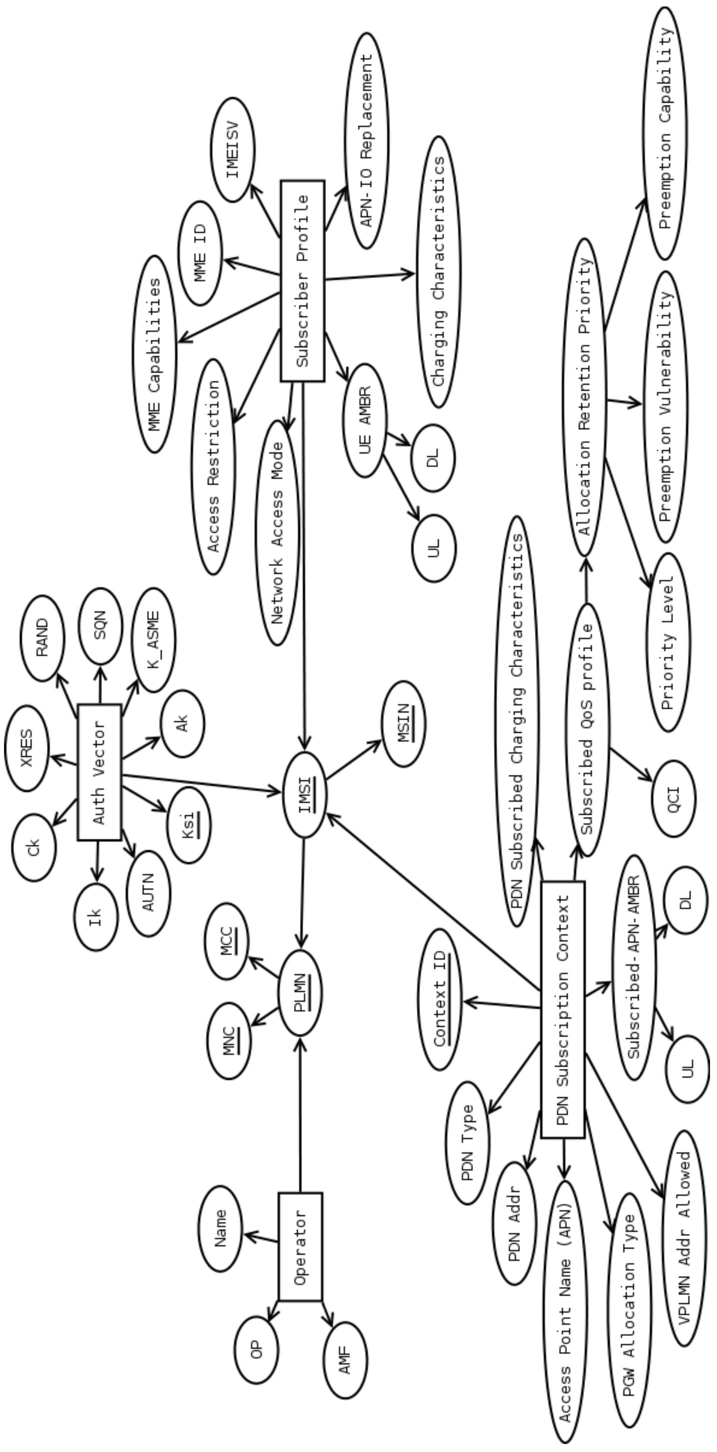


Figure 36: HSS database structure design

A.2 Database definition

Include scripts to create MySQL database.

```
-- MySQL dump 10.14  Distrib 10.0.3-MariaDB, for debian-linux-gnu (i686)
--
-- Host: localhost    Database: hss_lte_db
-- ------------------------------------------------------
-- Server version 10.0.3-MariaDB-1~wheezy-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
 FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Current Database: `hss_lte_db`
--

/*!40000 DROP DATABASE IF EXISTS `hss_lte_db`*/;

CREATE DATABASE /*!32312 IF NOT EXISTS*/ `hss_lte_db`
/*!40100 DEFAULT CHARACTER SET latin1 */;

USE `hss_lte_db`;

--
-- Table structure for table `auth_vec`
--

DROP TABLE IF EXISTS `auth_vec`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `auth_vec` (
  `mcc` smallint(3) unsigned NOT NULL,
  `mnc` smallint(3) unsigned NOT NULL,
  `msin` binary(5) NOT NULL,
  `ksi` bit(3) NOT NULL,
  `ik` binary(16) DEFAULT NULL,
  `ck` binary(16) DEFAULT NULL,
  `rand` binary(16) DEFAULT NULL,
  `xres` binary(8) DEFAULT NULL,
  `autn` binary(16) DEFAULT NULL,
  `sqn` binary(6) DEFAULT NULL,
  `kasme` binary(16) DEFAULT NULL,
  `ak` binary(6) DEFAULT NULL,
  PRIMARY KEY (`mcc`,`mnc`,`msin`,`ksi`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `operators`
--

DROP TABLE IF EXISTS `operators`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `operators` (
  `mcc` smallint(3) unsigned NOT NULL,
  `mnc` smallint(3) unsigned NOT NULL,
  `op` binary(16) DEFAULT NULL,
  `amf` binary(2) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`mcc`,`mnc`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `pdn_subscription_ctx`
--

DROP TABLE IF EXISTS `pdn_subscription_ctx`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `pdn_subscription_ctx` (
  `mcc` smallint(3) unsigned NOT NULL,
  `mnc` smallint(3) unsigned NOT NULL,
  `msin` binary(5) NOT NULL,
  `ctx_id` tinyint(3) unsigned NOT NULL DEFAULT '0',
  `apn` varchar(30) DEFAULT NULL,
  `pgw_allocation_type` bit(1) DEFAULT NULL,
  `vplmn_dynamic_address_allowed` bit(1) DEFAULT NULL,
  `eps_pdn_subscribed_charging_characteristics` binary(2) DEFAULT NULL,
  `pdn_addr_type` bit(2) DEFAULT NULL,
  `pdn_addr` binary(12) DEFAULT NULL,
  `subscribed_apn_ambr_dl` int(10) unsigned DEFAULT NULL,
  `subscribed_apn_ambr_up` int(10) unsigned DEFAULT NULL,
  `qci` tinyint(3) unsigned DEFAULT NULL,
  `qos_allocation_retention_priority_level` tinyint(3) unsigned
 DEFAULT NULL,
  `qos_allocation_retention_priority_preemption_capability` bit(1)
 DEFAULT NULL,
  `qos_allocation_retention_priority_preemption_vulnerability` bit(1)
 DEFAULT NULL,
  PRIMARY KEY (`mcc`,`mnc`,`msin`,`ctx_id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `subscriber_profile`
--

DROP TABLE IF EXISTS `subscriber_profile`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `subscriber_profile` (
  `mcc` smallint(3) unsigned NOT NULL,
  `mnc` smallint(3) unsigned NOT NULL,
  `msin` binary(5) NOT NULL,
  `msisdn` bigint(16) NOT NULL,
  `k` binary(16) DEFAULT NULL,
  `opc` binary(16) DEFAULT NULL,
  `imsisv` binary(8) DEFAULT NULL,
  `mmec` tinyint(3) unsigned DEFAULT NULL,
  `mmegi` smallint(5) unsigned DEFAULT NULL,
  `network_access_mode` tinyint(3) unsigned DEFAULT NULL,
  `ue_ambr_ul` int(10) unsigned DEFAULT NULL,
  `ue_ambr_dl` int(10) unsigned DEFAULT NULL,
  `apn_io_replacement` varchar(30) DEFAULT NULL,
  `charging_characteristics` binary(2) DEFAULT NULL,
  PRIMARY KEY (`mcc`,`mnc`,`msin`)
) ENGINE=MyISAM AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

```
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2013-07-12 19:37:25
# DB access rights
grant delete,insert,select,update on hss_lte_db.* to hss@localhost
 identified by 'hss';
```

B SAE-GW patches

B.1 GTP-U Length Fix

```
diff -crB nwepc-0.16-old/nw-gtpv1u/src/NwGtpv1u.c
  nwepc-0.16/nw-gtpv1u/src/NwGtpv1u.c
*** nwepc-0.16-old/nw-gtpv1u/src/NwGtpv1u.c
      2011-05-30 16:16:14.000000000 +0300
--- nwepc-0.16/nw-gtpv1u/src/NwGtpv1u.c
      2013-07-18 18:30:14.000000000 +0300
***************
*** 174,180 ****
      (pMsg->npduNumFlag);

    *(msgHdr++)         = (pMsg->msgType);
!   *((NwU16T*) msgHdr) = htons(pMsg->msgLen);
    msgHdr += 2;

    *((NwU32T*) msgHdr) = htonl(pMsg->teid);
--- 174,180 ----
      (pMsg->npduNumFlag);

    *(msgHdr++)         = (pMsg->msgType);
!   *((NwU16T*) msgHdr) = htons(pMsg->msgLen - 8);
    msgHdr += 2;

    *((NwU32T*) msgHdr) = htonl(pMsg->teid);

diff -crB nwepc-0.16-old/nw-gtpv1u/src/NwGtpv1uTrxn.c
          nwepc-0.16/nw-gtpv1u/src/NwGtpv1uTrxn.c
*** nwepc-0.16-old/nw-gtpv1u/src/NwGtpv1uTrxn.c
        2011-05-16 17:37:56.000000000 +0300
--- nwepc-0.16/nw-gtpv1u/src/NwGtpv1uTrxn.c
      2013-07-18 18:30:25.000000000 +0300
***************
```

```
*** 336,342 ****
                          (pMsg->npduNumFlag);

    *(msgHdr++)         = (pMsg->msgType);
!   *((NwU16T*) msgHdr) = htons(pMsg->msgLen);
    msgHdr += 2;

    *((NwU32T*) msgHdr) = htonl(pMsg->teid);
--- 336,342 ----
                          (pMsg->npduNumFlag);

    *(msgHdr++)         = (pMsg->msgType);
!   *((NwU16T*) msgHdr) = htons(pMsg->msgLen - 8);
    msgHdr += 2;

    *((NwU32T*) msgHdr) = htonl(pMsg->teid);
```

B.2 Ping response CheckSum Fix

```
diff -crB nwepc-0.16-old/nw-sdp/src/NwSdp.c
          nwepc-0.16/nw-sdp/src/NwSdp.c
*** nwepc-0.16-old/nw-sdp/src/NwSdp.c
      2011-09-16 10:51:08.000000000 +0300
--- nwepc-0.16/nw-sdp/src/NwSdp.c
      2013-07-19 16:04:18.000000000 +0300
***************
*** 502,507 ****
--- 502,538 ----
    return NW_SDP_OK;
  }

+ static NwSdpRcT
+ nwChecksum(NwU8T *data, NwU16T checklen, NwU8T *chksm)
+ {
+       NwU32T sum = 0;
+       NwU16T answer = 0;
+       NwU16T wordData[checklen];
+       NwU16T *startpos = wordData;
+
+       memcpy(startpos, data, checklen);
+
+       while (checklen > 1)
+       {
+             sum += *startpos++;
+             checklen -= 2;
+       }
+
+       if (checklen == 1)
+       {
+             *(NwU8T *)(&answer) = *(NwU8T *)startpos;
+             sum += answer;
+       }
+
+       sum = (sum >> 16) + (sum & 0xffff);
+       sum += (sum >> 16);
+       answer = ~sum;
+
+       memcpy(chksm, &answer,2);
+
+       return NW_SDP_OK;
+ }
+
  NwSdpRcT
  nwSdpProcessGtpuDataIndication(NwSdpT* thiz,
               NwSdpFlowContextT* pFlowContext,
***************
*** 546,552 ****
              memcpy(pingRspPdu + 14, pIpv4Pdu, pingRspPduLen);
              memcpy(pingRspPdu + 14 + 16, pIpv4Pdu + 12, 4);
              memcpy(pingRspPdu + 14 + 12, pIpv4Pdu + 16, 4);
!             /* TODO: Add ip-checksum */
              rc = nwSdpProcessIpv4DataInd(thiz, 0, pingRspPdu,
                                           pingRspPduLen + 14);
            }
            else
--- 577,587 ----
              memcpy(pingRspPdu + 14, pIpv4Pdu, pingRspPduLen);
              memcpy(pingRspPdu + 14 + 16, pIpv4Pdu + 12, 4);
              memcpy(pingRspPdu + 14 + 12, pIpv4Pdu + 16, 4);
!             /* Add ip-checksum */
!            *(pingRspPdu + 14 + 16 + 4 + 2)=0x00;
!            *(pingRspPdu + 14 + 16 + 4 + 3)=0x00;
!            nwChecksum(pingRspPdu + 14 + 16 + 4,
!                      pingRspPduLen - 20,
!                      pingRspPdu + 14 + 16 + 4 + 2);
!
              rc = nwSdpProcessIpv4DataInd(thiz, 0, pingRspPdu,
                                           pingRspPduLen + 14);
            }
            else
```