Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Gopika Premsankar

Design and Implementation of a Distributed Mobility Management Entity (MME) on OpenStack

Master's Thesis
Espoo, July 15, 2015

Supervisor: Professor Antti Ylä-Jääski
Advisor: Sakari Luukkainen D.Sc. (Tech.)

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Gopika Premsankar

Title: Design and Implementation of a Distributed Mobility Management Entity (MME) on OpenStack

Date: July 15, 2015 **Pages:** 77

Major: Data Communication Software **Code:** T-110

Supervisor: Professor Antti Ylä-Jääski

Advisor: Sakari Luukkainen D.Sc. (Tech.)

Network Functions Virtualisation (NFV) involves the implementation of network functions, for example firewalls and routers, as software applications that can run on general-purpose servers. In present-day networks, each network function is typically implemented on dedicated and proprietary hardware. By utilising virtualisation technologies, NFV enables network functions to be deployed on cloud computing infrastructure in data centers.

This thesis discusses the application of NFV to the Evolved Packet Core (EPC) in Long Term Evolution (LTE) networks; specifically to the Mobility Management Entity (MME), a control plane entity in the EPC. With the convergence of cloud computing and mobile networks, conventional architectures of network elements need to be re-designed in order to fully harness benefits such as scalability and elasticity. To this end, we design and implement a distributed MME with a three-tier architecture common to web applications. We highlight design considerations for moving MME functionality to the cloud and compare our new distributed design to that of a standalone MME. We deploy and test the distributed MME on two separate OpenStack clouds. Our results indicate that the benefits of scalability and resilience can outweigh the marginal increase in latency for EPC procedures. We find that the latency is dependent on the actual placement of MME components within the data center. Also, we believe that extensions to the OpenStack platform are required before it can meet performance and availability requirements for telecommunication applications.

Keywords: Network Functions Virtualisation, Mobility Management Entity, Evolved Packet Core, OpenStack

Language: English

Acknowledgements

I would like to thank my supervisor Professor Antti Ylä-Jääski and my instructor Docent Sakari Luukkainen for giving me the opportunity to work on this topic. I would also like to thank my co-worker Kimmo Ahokas for helping me with the testbed. I am grateful to Dr. Jose Costa-Requena, Vicent Ferrer Guasch and Jesús Llorente Santos for our discussions on this thesis. I would also like to thank Professor Keijo Heljanko for his insights into distributed design and Professor Mario Di Francesco for our discussions on the presentation of scientific results.

I would like to thank my friends and colleagues for their support. Special thanks to my husband for encouraging me and motivating me to do my best. This thesis would not have been possible without the help and support from my family.

Espoo, July 15, 2015

Gopika Premsankar

This work has been performed in the framework of CELTIC-Plus project C2012/2-5 SIGMONA. The authors would like to acknowledge the contributions of their colleagues, although the views expressed are those of the author and do not necessarily represent the project. This information reflects the consortium's view, but the consortium is not liable for any use that may be made of any of the information contained therein.

Abbreviations and Acronyms

3GPP	3rd Generation Partnership Project
APN	Access Point Name
BTS	Base Transceiver Station
DHT	Distributed Hash Table
ECGI	E-UTRAN Cell Global Identifier
ECM	EPS Connection Management
EMS	Element Management System
EMM	EPS Mobility Management
eNodeB	Evolved Node B
ESM	EPS Session Management
EPC	Evolved Packet Core
EPS	Evolved Packet System
E-UTRAN	Evolved Universal Terrestrial Access Network
GPRS	General Packet Radio Service
GRE	Generic Routing Encapsulation
GSM	Global System for Mobile Communication
GTP	GPRS Tunneling Protocol
GTP-C	GPRS Tunneling Protocol for Control Plane
GUMMEI	Globally Unique MME Identifier
GUTI	Globally Unique Temporary Identifier
IE	Information Element
IMSI	International Mobile Subscriber Identity
IMS	IP Multimedia Core Network Subsystem
IP	Internet Protocol
ISDN	Integrated Services Digital Network
LTE	Long Term Evolution
MAC	Medium Access Control
MME	Mobility Management Entity
MSISDN	Mobile Station ISDN Number
NAS	Non Access Stratum

NFV	Network Functions Virtualisation
O&M	Operation and Management
OVS	Open vSwitch
PDCP	Packet Data Convergence Protocol
PDN	Packet Data Network or Public Data Network
PGW	PDN Gateway
PHY	Physical layer
PLMN	Public Land Mobile Network
QoS	Quality of Service
REST	Representational State Transfer
RLC	Radio Link Control
RRC	Radio Resource Control
S1AP	S1 Application Protocol
SCTP	Stream Control Transmission Protocol
SDN	Software-Defined Networking
SGW	Serving Gateway
SMS	Short Message Service
SQL	Structured Query Language
SSH	Secure Shell
TA	Tracking Area
TAI	Tracking Area Identity
telecom	telecommunication
UDP	User Datagram Protocol
UE	User Equipment
UMTS	Univeral Mobile Telecommunications System
VLAN	Virtual Local Area Network
VM	Virtual Machine

Contents

Chapter 1

Introduction

Mobile networks today comprise of specialised routers built on proprietary hardware that is designed to meet high performance requirements. Upgrading or expanding the network demands investment in expensive hardware and the deployment process is slow and cumbersome [60]. Network Functions Virtualisation (NFV) has emerged as a solution for mobile network operators to rapidly meet the growing demand for mobile data [11] while simultaneously allowing them to reduce expenditure in specialised hardware. By using virtualisation technologies, NFV enables network functions to be deployed on cloud computing infrastructure in data centers. Mobile network operators can then utilise distributed data centers to virtualise and decentralise their networks elastically and cost-effectively [59].

This convergence of mobile networks and cloud computing enables the creation of a flexible, scalable and intelligent core network. A flexible core network is required to meet the requirements of future 5G networks and to integrate new radio access technologies with existing Long Term Evolution (LTE) and WiFi access networks [22, 27]. This thesis focuses on the virtualisation of the Evolved Packet Core (EPC) in Long Term Evolution (LTE) networks. We design and implement a distributed Mobility Management Entity (MME), an important control plane element in the EPC network.

1.1 Motivation

The MME is an ideal candidate for virtualisation as it is purely a control plane element and does not handle user data traffic. This means that the MME does not need to make use of specialised hardware for processing data packets and thus its functionality can be moved to general-purpose hardware in the cloud. Furthermore, in present-day networks, signalling traffic

is growing rapidly [24, 53, 62]. This can be attributed to continuous keep-alive signalling generated from smartphones [53] and emerging machine to machine applications [24, 62]. An MME deployed on the cloud can employ virtually infinite computing resources to handle this load. However, moving the MME software as a standalone application to the cloud does not allow us to fully leverage the benefits of cloud computing. Our work in re-designing the MME architecture is motivated by the following factors:

- Network functions in the EPC are designed to run on dedicated hardware. If the signalling traffic increases substantially, operators need to deploy additional MMEs to handle the increasing load. With an intelligent architecture, an MME deployed on the cloud can be made to *scale autonomously* and with minimal network management tasks.

- Network elements in the mobile core are usually overprovisioned to meet peak load demand. However, this can lead to waste of resources during non-peak hours [60]. The MME, being a stateful network element, cannot be scaled in without affecting existing sessions. A good design for the MME software can enable *elasticity* through efficient scaling in of virtual resources when no longer required and without affecting end users.

- The failure of a network element impacts the delivery of services to end users. The failure of an MME is particularly significant as it is the main control element in the EPC and is involved in setting up connections for users [63]. Thus, it is important to ensure *resilience* of the MME and restoration of user sessions in case of failures in the underlying hardware. Currently, an active-standby redundant architecture with vendor-specific built-in support for transparent failover is used to ensure resilience [31, 51]. State checkpointing and network logging are not suited for cloud deployments as they can contribute to network congestion and increased network latency [31]. Virtualisation of network functions requires resilience to be built into the software architecture [46]. Thus, it is necessary to build resilience into the MME software.

1.2 Research statement

Section 1.1 identifies the requirements for a virtualised MME in order to harness the benefits of cloud computing. Our goal is to develop a suitable software architecture to achieve a cloud-optimised design for the MME. We

aim to demonstrate the advantages and disadvantages of the chosen architecture through experimental evaluation.

1.3 Contribution

We present a new three-tier architecture for the MME, designed to be flexibly scaled out or scaled in depending on user traffic. This new design also achieves resilience to the failure of certain Virtual Machines (VMs) in the cloud. We develop a prototype implementation of the distributed MME and demonstrate the benefits of the new architecture. Additionally, we discuss design choices made for each tier of the architecture and evaluate the effects of these choices. We test and evaluate the performance of the MME on two OpenStack installations.

1.4 Research methodology

We assess the new MME architecture and design choices by experimental evaluation. More specifically the methodology used is *experimental computer science* [40], wherein we evaluate our solution through the development of a prototype system. The MME software developed in this thesis is a prototype implementation which supports the basic functionality required to compare its performance to the original design. The developed software is intended as a proof-of-concept to demonstrate the benefits and drawbacks of a novel architecture for the MME.

1.5 Structure of thesis

The rest of the thesis is structured as follows: Chapter 2 introduces the relevant background topics and describes various architecture choices for virtualising elements in the EPC. Chapter 3 presents the chosen architecture for the MME and describes our implementation and design considerations. Chapter 4 describes the testbed and experimental plan to evaluate the distributed MME. Chapter 5 presents the results of the experiments. Chapter 6 discusses the results obtained and enhancements to the testbed. Finally, Chapter 7 concludes the thesis.

Chapter 2

Background

This chapter provides background information on the technologies and concepts relevant to this thesis. Section 2.1 introduces the EPC and its constituent elements. Section 2.2 describes the functions and interfaces of the MME in detail. Section 2.3 discusses two procedures used in the EPC to provide connectivity to end users. Section 2.4 introduces the concept of Network Functions Virtualisation (NFV) and its application to EPC. Section 2.5 describes architecture choices for a virtualised element in the mobile core network. Finally, Section 2.6 provides a literature review of current work in the virtualisation of EPC.

2.1 Evolved Packet Core (EPC)

The continuous need for higher data rates, improved Quality of Service (QoS), lower latencies for connection setup, and shorter round trip times has led to a diverse and evolving set of technologies in mobile networks [59]. Figure 2.1 shows the evolution of mobile network architectures from circuit-switched Global System for Mobile Communications (GSM) networks to the current Evolved Packet System (EPS). GPRS networks first introduced packet switching, thereby removing the need for dedicated end-to-end paths. However, these networks could not sufficiently support the high data rates needed for multimedia services. The demand for faster data rates resulted in a new access network in the next generation of Universal Mobile Telecommunications System (UMTS) networks. UMTS networks have a core network similar to that in GPRS networks. The need for even faster networks led to the development of a completely new core network and access technology in the EPS networks. The 3rd Generation Partnership Project (3GPP), a collaboration between seven telecommunication standard development or-

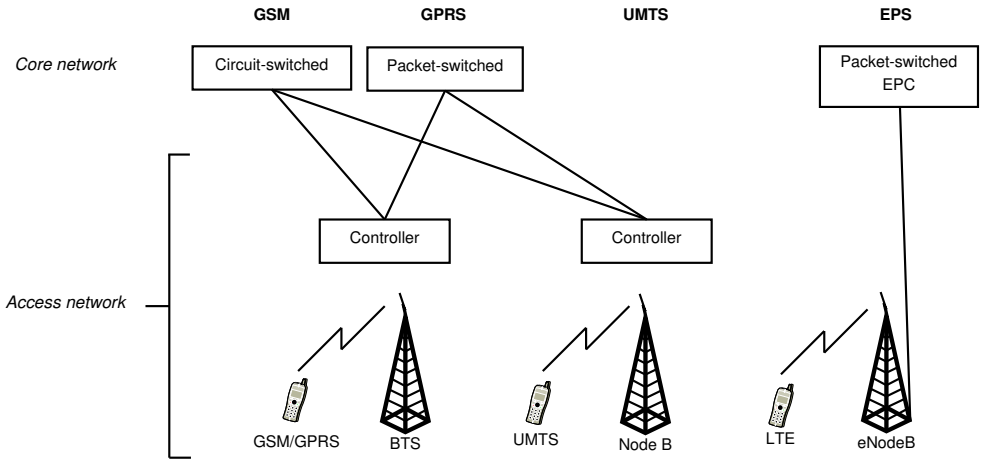


Figure 2.1: Evolution of mobile network architecture [13]

ganisations[1], maintains and develops the standardised specifications related to GSM (including GPRS), UMTS and EPS networks.

The EPS represents the 4th generation (4G) in the evolution of mobile networks. The radio access network of the EPS is known as Long Term Evolution (LTE) or Evolved Universal Terrestrial Access Network (E-UTRAN). The Evolved Packet Core (EPC) is the core network of the EPS. The EPC is an all-IP based packet-switched system. Since the EPC no longer contains a circuit-switched domain, all data, voice and Short Message Service (SMS) services are transported over the IP protocol. The EPC has a flat architecture with only a few nodes involved in handling user traffic, thereby resulting in fewer protocol conversions, lower latencies and higher performance [14]. Another key feature of the EPC architecture is the separation of control plane (or signalling plane) and data plane (or user plane) elements. Control plane signalling consists of messages related to mobility and management, whereas the data plane carries data packets for user sessions. This separation enables operators to dimension and adapt their networks [14].

2.1.1 Architectural elements of EPC

Figure 2.2 shows the basic EPS architecture with the main network elements and interfaces relevant to this thesis. The User Equipment (UE) is the device used by the end user in the network. We use the term UE and user interchangeably in this thesis. The UE connects to the EPC over the E-UTRAN radio access network. The eNodeB is part of the radio access network and is

[1] `http://www.3gpp.org/about-3gpp/`

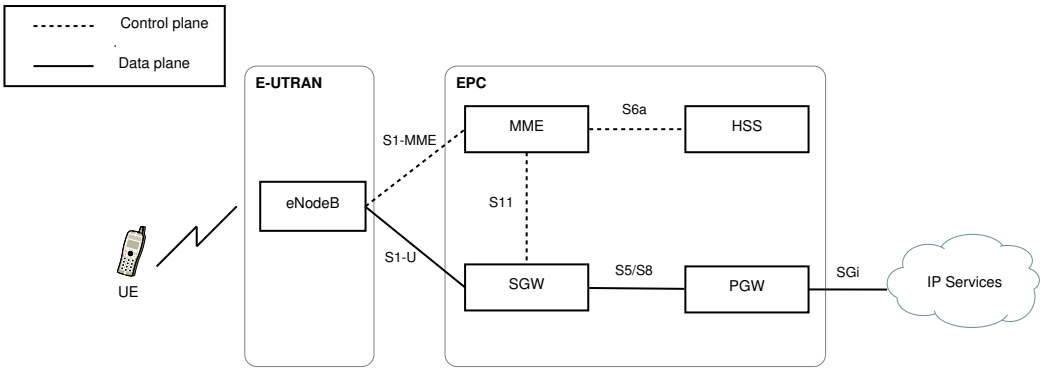


Figure 2.2: Basic EPS architecture comprised of E-UTRAN radio access network and the core network, EPC

responsible for the air interface towards the UE. The EPC includes the Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (SGW) and PDN Gateway (PGW). The EPC is connected to external Packet Data Networks (PDN) which provide IP services to the UE. The EPC could also be connected to the IP Multimedia Core Network Subsystem (IMS) to deliver IP multimedia services (not shown in figure).

A brief description of the elements in EPC is provided below. The MME, being the topic of this thesis, is described in greater detail in Section 2.2.

- **HSS** – The HSS is the master database for the UEs. It maintains subscription information for each user, including identification parameters, security information for authentication, ciphering and integrity, user location data, and user profile information [18]. Other network elements provide services to the UE by querying the HSS for the required information.

- **MME** – The MME is a control plane element responsible for management of users. The functions of the MME include authentication, managing user session states, paging, mobility and EPS bearer state management for each UE connected to it.

- **SGW** – The SGW serves a UE by routing IP data packets to and from the UE [14]. It also acts as the anchor point for UE handovers between eNodeBs and also between non-3GPP networks [18].

- **PGW** – The PGW is a data plane element that routes incoming and outgoing IP packets from external data networks. Together, the PGW and SGW enable data packets to be transferred between the UE and

external IP networks. The PGW is also responsible for UE IP address allocation, policy enforcement and packet filtering [18].

To provide data services to the UE, EPS networks employ the concept of *EPS bearers*. An EPS bearer represents a logical connection between the UE and the EPC. It comprises of tunnels between different network elements, over which data packets are actually delivered using IP protocol. IP packets are sent over the radio interface between the UE and eNodeB, tunneled over the S1-U interface between the eNodeB and SGW and transferred over S5/S8 interface between the SGW and PGW. An EPS bearer uniquely identifies traffic flows between a UE and PGW and is associated with a common Quality of Service (QoS) control [17].

2.2 Mobility Management Entity (MME)

The MME is responsible for mobility management of UEs. Right from the moment a user attaches to the LTE network, the MME keeps track of the location of the UE and its state information. It is involved in the initial authentication procedure for a UE and authorises a user to attach to the network. On successful authorisation, the MME manages the establishment of bearers for data connectivity by selecting the appropriate SGW and PGW for a user. It also assigns a temporary identifier to the UE, which is then used in all subsequent procedures to identify the user in the network. When a UE goes into idle mode due to inactivity, the MME is responsible for paging it in case of network-initiated events. It is also the main mobility anchor for a user and handles handovers between eNodeBs as well as to other access networks. The MME terminates the Non Access Stratum (NAS) interface towards the UE and is responsible for ciphering and integrity protection of these NAS messages.

To summarise, among the main functionalities of the MME as listed in [17] are:

- Authentication and authorisation of UEs
- PGW and SGW selection
- Maintaining UE reachability when in idle state
- Bearer management for PDN connectivity
- Mobility management, i.e. management of handovers between eNodeBs or different access networks

- NAS signalling and associated security towards UE
- Lawful interception of signalling traffic

2.2.1 Interfaces and protocol stacks

The interfaces towards the MME, as depicted in Figure 2.2, are described below. In addition to these basic interfaces, the MME has several other interfaces which are not described in this thesis and can be found in [17].

- **S1-MME** – S1-MME is the control plane interface between the eNodeB and MME. This interface uses S1 Application Protocol (S1AP) [16] over Stream Control Transmission Protocol (SCTP) [57]. SCTP is a reliable, stream-oriented transport protocol, which ensures reliable delivery of messages between the eNodeB and MME. The NAS protocol is used for control plane messaging between the UE and MME. NAS messages are delivered in S1AP messages and are transparent to the eNodeB. The functionality performed by the NAS protocol can be broadly classified into EPS Mobility Management (EMM) for mobility of the UE and EPS Session Management (ESM) for the UE's IP connectivity [19]. Figure 2.3 shows the protocol stack for communication between the UE, eNodeB and MME. We do not discuss the radio interface protocol stack for the LTE-Uu interface, details of which can be found in [20].

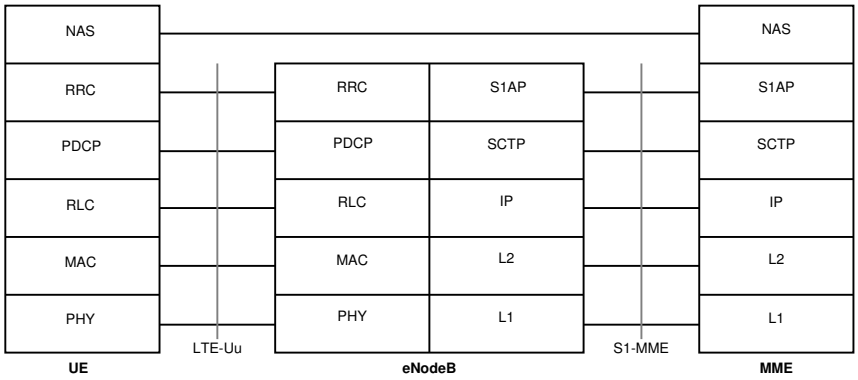


Figure 2.3: Control plane protocol stack between UE, eNodeB and MME

- **S11** – The MME and SGW exchange control plane messages over the S11 interface. This interface uses GPRS Tunneling Protocol for the control plane (GTP-C) [15] over User Datagram Protocol (UDP) [54], as depicted in Figure 2.4. A GTP tunnel is established per UE over the

S11 interface, which is used to exchange all control plane messages for the particular user [15]. The MME and SGW each maintains a Tunnel Endpoint ID (TEID), an IP address and UDP port number to identify the GTP-C tunnel.

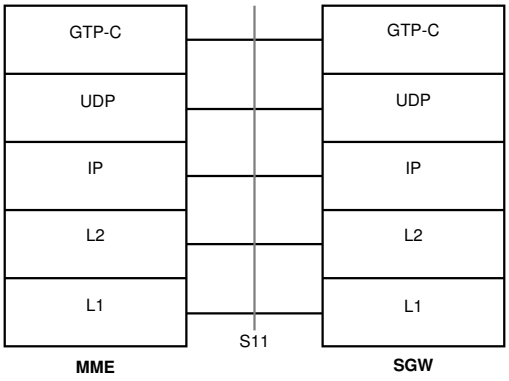


Figure 2.4: Control plane protocol stack between MME and SGW (S11 interface)

- **S6a** – The MME and HSS exchange information over the S6a interface. This interface uses Diameter protocol [32] over SCTP. The protocol stack is depicted in Figure 2.5.

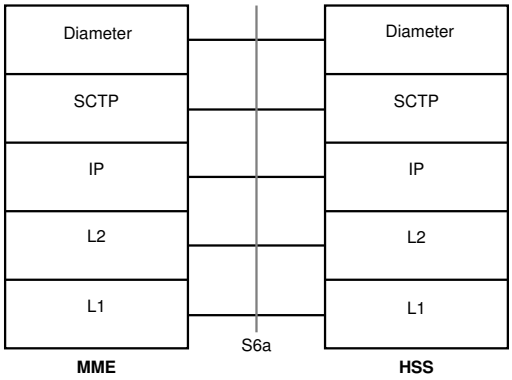


Figure 2.5: Control plane protocol stack between MME and HSS (S6a interface)

2.2.2 State information maintained by MME

An MME maintains information on a UE based on its state in the network. For each UE attached to it, the MME maintains a mobility management

context and information about EPS bearers established. A subset of the information stored on an MME is provided in Table 2.1. Only those fields relevant to this thesis have been included in this table. A complete list of all fields stored on an MME is provided in [19]. The state information maintained on an MME is often referred to as the *UE context*. This information varies depending on the state of the UE in the network. For example, when a UE detaches from the network, a small subset of its information is still stored, such as user identifiers and security context information. However, the location of the UE is no longer stored. When a UE is connected to the network and an EPS bearer exists, the MME associated with the UE knows the location of the UE to the tracking area (TA) where it is registered from.

2.3 EPS procedures

This section describes two procedures – initial attach and detach procedures, used in EPS networks to provide connectivity to end users. We implement only these procedures on the MME.

2.3.1 E-UTRAN initial attach procedure

The attach procedure allows a UE to register to the network and enables IP connectivity by creating a default EPS bearer [17] for the UE. Figure 2.6 shows the call flow for an initial attach to the LTE network. A brief description of the initial attach procedure, including the main messages and their important Information Elements (IEs), is provided below:

1. The UE attempts to attach to the network by sending an **Attach Request**, which includes the UE's International Mobile Subscriber Identity (IMSI), attach type and ESM container for requesting data connectivity.

2. On receiving this message, the eNodeB assigns an identifier to the UE, known as the eNodeB UE S1AP ID. The eNodeB then selects an appropriate MME and forwards the Attach Request to it. The message is sent as part of an S1AP message, encapsulated in a NAS message along with information on the current location of the UE.

3. As part of the authentication and security step, the MME requests for authentication vectors from the HSS using an **Authentication Information Request**. The HSS responds with the requested information

Field	Description
IMSI	IMSI is the subscriber's permanent identity
MSISDN	Basic MSISDN of the UE; depending on whether present in HSS
MM State	MM state of the UE: ECM-IDLE, ECM-CONNECTED or EMM-DEREGISTERED
GUTI	Globally Unique Temporary Identity assigned to the UE
MME IP address for S11	MME IP address for S11 interface used by SGW
MME TEID for S11	MME Tunnel Endpoint Identifier for S11 interface
SGW IP address for S11	SGW IP address for S11 interface
SGW TEID for S11	SGW Tunnel Endpoint Identifier for S11 interface
eNodeB address in use for S1-MME	IP address of the eNodeB currently used for S1-MME interface
eNodeB UE S1AP ID	Unique identity for the UE within eNodeB
MME UE S1AP ID	Unique identity for the UE within MME
TAI of last TAU	Tracking Area Identity of the TA in which the last Tracking Area Update was initiated
E-UTRAN CSG ID	Last known E-UTRAN Cell Global Identity
NAS security context	Key parameters for establishing the NAS security context including security keys
For each active PDN connection	
PDN type	IPv4, IPv6 or IPv4v6
IP address(es)	Allocated IPv4 or IPv6 address
APN in use	Access Point Name (APN) currently used
PGW address (control plane)	IP address of PGW used for sending control plane signalling
PGW TEID for S5/S8 (control plane)	PGW TEID for S5/S8 interface for control plane
APN-AMBR	Maximum Aggregated uplink and downlink MBR values
EPS subscribed QoS profile	Bearer level QoS parameters for the default bearer
For each bearer within the PDN connection	
EPS Bearer ID	Unique identifier for the EPS bearer
SGW IP address for S1-U	IP address of the SGW for S1-U interface towards eNodeB (data plane)
SGW TEID for S1-U	Tunnel Endpoint Identifier of the SGW for the S1-U interface towards eNodeB (data plane)
EPS bearer QoS	QoS parameters for EPS bearer

Table 2.1: UE Context information stored on MME [19]

in an **Authentication Information Answer** message. Once the authentication vectors have been acquired, the MME and UE mutually authenticate each other through **Authentication Request** and **Authentication Response** messages. On completion of authentication, the MME establishes a NAS security association towards the UE to enable ciphering and integrity protection of further NAS messages. The MME assigns an MME UE S1AP ID to the UE to uniquely identify it within the MME. A combination of eNodeB UE S1AP ID and MME UE S1AP ID can be used to identify the S1-MME connection for a UE.

4. The MME now sends an **Update Location Request** message to the HSS to inform the HSS of the user's registration to the network and to request subscription information for the UE.

5. The HSS acknowledges the request by sending an **Update Location Ack** which includes the subscription data for the user, such as subscribed PDN type, QoS profile and Access Point Name (APN). Based on this information, the MME validates the UE's presence in the Tracking Area (TA) and services requested by the UE. If all the checks are successful, a new context is created for the user on the MME.

6. The MME then selects an SGW and allocates an EPS bearer Identity for the default EPS bearer to be created. It sends a **Create Session Request** towards the SGW to request the creation of a default EPS bearer. This includes the IMSI, EPS bearer ID, IP address of the PGW (which is selected based on subscription data), APN and subscribed QoS values.

7. The SGW sends a Create Session Request to the PGW over the S5 interface, along with other required parameters.

8. On receipt of this message, the PGW creates a new entry in its EPS bearer context table and allocates an IP address to the UE. The PGW can now route packets between the SGW and external data network. The PGW also enforces policy control and generates a Charging ID for the bearer to enable charging of the subscriber. The PGW now sends a **Create Session Response** to the SGW, which includes the QoS profile and TEID for establishing the S5 tunnel between the SGW and PGW.

9. The SGW then allocates a TEID for the S1-U interface for data transfer. It includes this information in the Create Session Response and sends it to the MME.

10. The MME sends an **Attach Accept** message to the eNodeB in an S1AP **Initial Context Setup Request** message. The Attach Accept includes a Globally Unique Temporary Identifier (GUTI) allocated by the MME for the UE. The Initial Context Setup Request contains parameters to enable the eNodeB to set up the S1-U bearer to the SGW and allocate radio resources to the UE.

11. The eNodeB forwards the Attach Accept to the UE. This message informs the UE of the newly allocated GUTI and the tracking area list in which the UE can roam freely without having to initiate a Tracking Area Update Procedure.

12. Through an RRC reconfiguration procedure, the eNodeB establishes radio bearers to enable data transfer.

13. The eNodeB sends an **Initial Context Setup Response** to the MME informing the MME of the eNodeB's Tunnel Endpoint Identifier (TEID) and IP address. This is required to set up a GTP tunnel on the S1-U interface for data packet transfer between the eNodeB and SGW.

14. The UE sends an **Attach Complete** message to the MME in response to the Attach Accept.

15. On receipt of both Initial Context Setup Response and Attach Complete messages, the MME sends a **Modify Bearer Request** to inform the SGW of the eNodeB's TEID and IP address.

16. The SGW acknowledges the MME with a **Modify Bearer Response**. With this message, downlink packets can be delivered from the external data network to the UE through the established bearer.

2.3.2 Detach procedure

A detach procedure is used by the UE to disconnect from the EPS network [17]. In this thesis, we only consider the explicit detach procedure initiated by the UE. Figure 2.7 presents the call flow for the detach procedure. A brief description of the call flow and its important messages is as follows:

1. The UE sends a **Detach Request** over NAS, which includes the UE's EPS mobile identity (either GUTI or IMSI) and a switch off indicator. The switch off flag is set if the UE initiates a detach due to the device switching off.

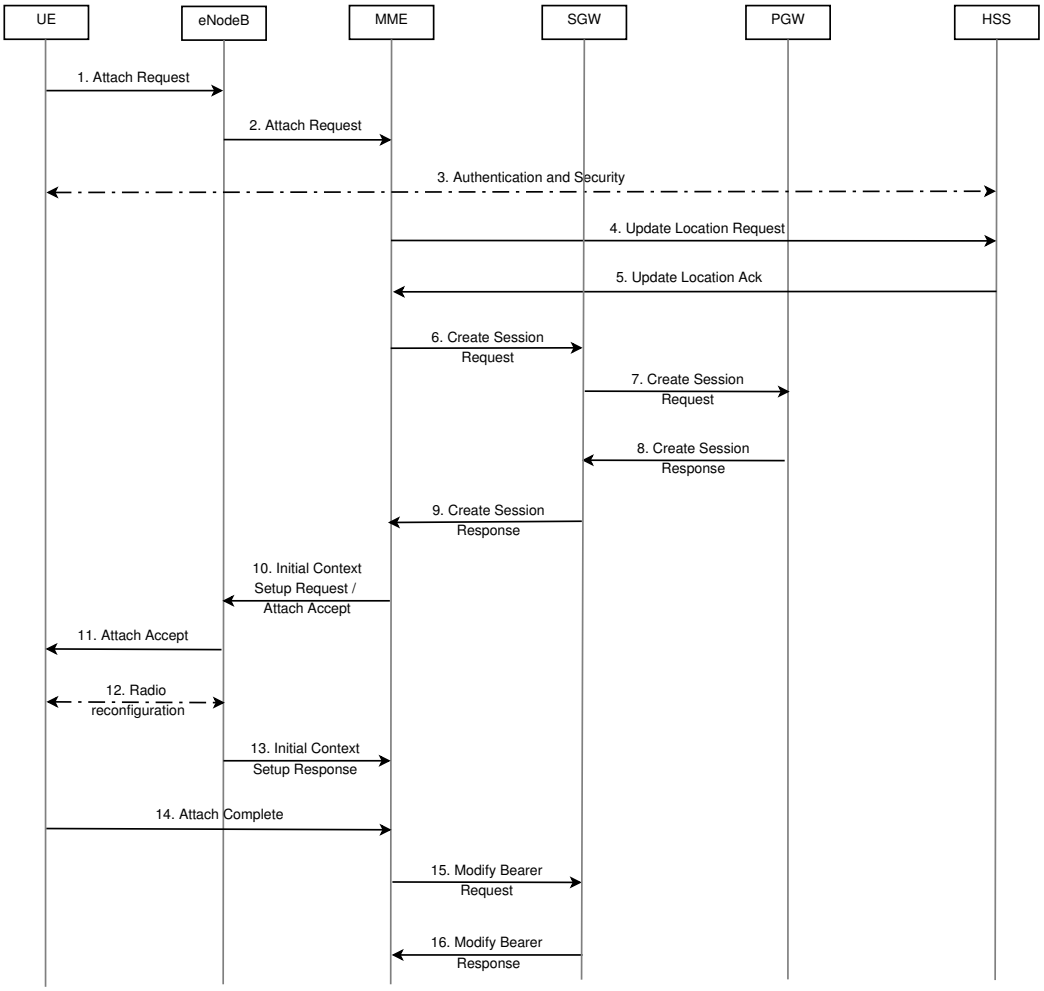


Figure 2.6: Call flow for initial attach procedure

2. The MME sends a **Delete Session Request** to the SGW for each EPS bearer to be deactivated for the particular UE.

3. The SGW releases the EPS bearer context information and initiates the deletion of the bearer on the PGW. It then acknowledges the MME with a **Delete Session Response**.

4. The MME acknowledges the UE with a **Detach Accept** if the switch off indicator was not set in the initial Detach Request.

5. The MME also tears down the S1 signalling connection between the MME and eNodeB for the UE by sending a **UE Context Release Command**.

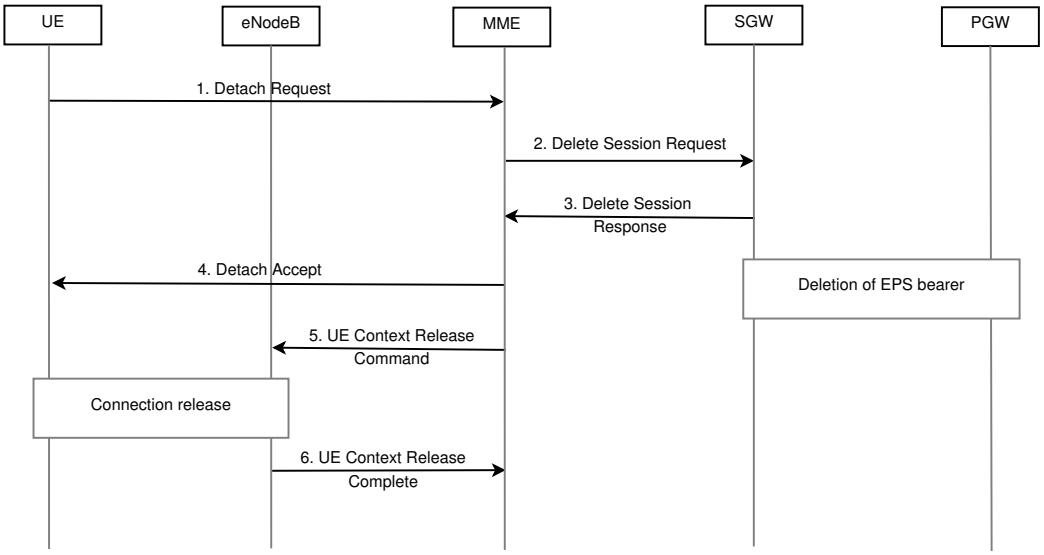


Figure 2.7: Call flow for detach procedure

6. The eNodeB confirms the tear down of the S1 signalling connection by sending a **UE Context Release Complete**.

2.4 Network Functions Virtualisation (NFV)

Network Functions Virtualisation (NFV) involves the implementation of network functions as software modules that can run on commercial off-the-shelf hardware. Currently, network elements run vendor-specific software on proprietary hardware. Typically, each network function is deployed on a dedicated hardware device. NFV aims to decouple the software from the underlying hardware by leveraging IT virtualisation technologies. With this approach, network functions will be implemented as Virtualised Network Functions (VNFs) which can be run on standard hardware in the form of Virtual Machines (VMs). Figure 2.8 illustrates the concept of NFV.

NFV was first introduced in 2012 in an introductory white paper [41] published by the NFV Industry Standards Group (ISG), comprising of seven leading telecom operators within the European Telecommunications Standards Institute (ETSI). By 2015 the number of members has increased to more than 270 companies, including 37 telecommunication operators, network equipment providers and IT vendors. The NFV ISG has also published several specifications including those for architectural framework, NFV infrastructure requirements, and management and orchestration.

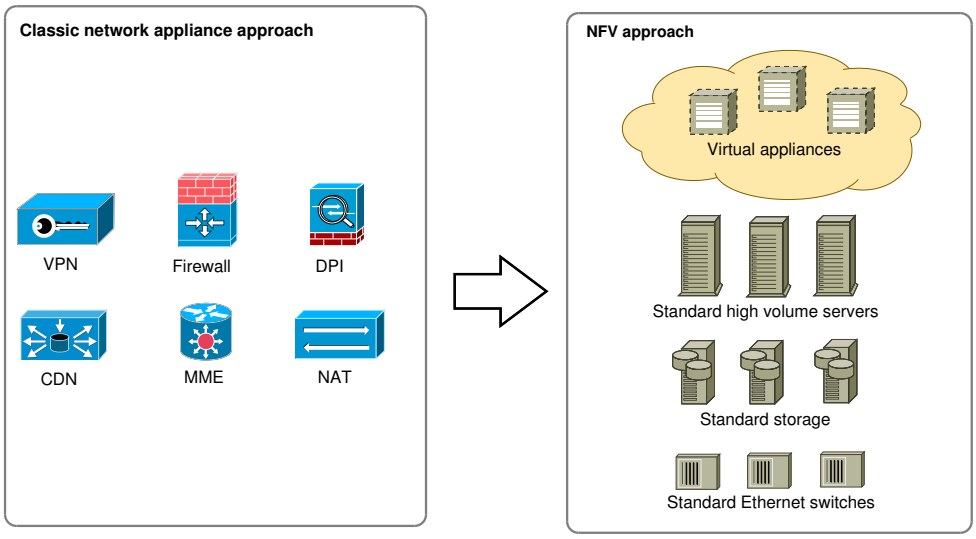


Figure 2.8: NFV approach to virtualise traditional network elements, such as Virtual Private Network (VPN), firewall, Deep Packet Inspection (DPI), Content Delivery Network (CDN), MME, Network Address Translation (NAT), etc. (Adapted from [41])

Among the nine use cases identified by the NFV ISG [42], this thesis focuses on the use case "Virtualization of Mobile Core Networks and IMS (IP Multimedia Subsystem)". Mobile network operators have to keep up with increasing demands from growing mobile data traffic and increasing number of connected devices. Although radio access technologies have evolved to allow faster data rates, the highly centralised core network and dependency on proprietary hardware still constrain network operators from taking full advantage of higher transmission rates [60]. Moving to the cloud computing paradigm allows network operators to decentralise their network architecture and minimise investments in hardware solutions [59].

2.4.1 Benefits

The application of NFV in EPC offers several benefits to network operators as discussed below:

- In present-day networks, significant costs are incurred in acquiring and installing hardware equipment for different network functions. In the NFV approach, different VNFs can be aggregated and run on the same standardised hardware platforms, thereby maximising utilisation of infrastructure and reducing energy consumption.

- NFV aims to spur service innovation in the field of telecommunications, as new services implemented as software modules can be deployed faster in the network. Currently the installation of hardware equipment in the network involves a build-integrate-deploy cycle [41] which requires highly qualified personnel to install, test and manage the devices. NFV reduces the time required for services to be deployed in the network as software-based developments have a shorter integration cycle.

- NFV can also assist network operators in dimensioning their networks. Core network elements are usually overprovisioned in order to handle potential increases in traffic. However, this results in components being unused or underutilised during periods of low activity. By leveraging IT virtualisation, operators can increase the number of virtual instances when subscriber traffic increases and remove these instances when no longer required. Thus, capacity can be dynamically increased or decreased allowing operators to react to real-time traffic patterns.

- NFV offers the additional benefit of improved flexibility, as VMs can be deployed in a location convenient for the operator. VMs can be deployed closer to the end user to reduce latency and improve overall Quality of Experience (QoE) for the user. For example, [59] introduces the concept of a Follow-Me-Cloud, wherein data and network services intelligently follow a user's movement. This allows for an optimal end-to-end service for the user.

2.4.2 Challenges

Although NFV offers several benefits, the virtualisation of network functions in a mobile network brings a unique set of challenges, different from standard IT virtualisation:

- Mobile network operators need to simultaneously meet requirements of high availability and very low latency. An MME that has to be 99.999% available can go offline for only less than 6 minutes per year [31].

- There is a possibility of performance degradation when moving software from proprietary to standard hardware. EPC networks are built on top of highly specialised hardware. This hardware generally offer features such as acceleration engines and dedicated processors for packet forwarding [41, 60]. The challenge is to provide the same level of performance as standardised hardware through software technologies and by using the appropriate hypervisor.

- Another pertinent issue is that of security. Execution of multiple VNFs over the same infrastructure requires that these instances be isolated from each other and the data stored on shared resources be secure. This is more problematic when different vendors are involved in the deployment of the virtualised elements.

- NFV also introduces new management and orchestration challenges in the network. It is now required to keep track of VNF instances, move VMs when required, allocate hardware resources during scaling operations, instantiate VNFs at the appropriate location depending on QoS requirements, and determine faults in VMs as well as underlying infrastructure.

2.5 Architecture of virtualised EPC network elements

With the convergence of cloud computing technologies and mobile networks, architectures of traditional network elements need to be re-invented to take advantage of the benefits of cloud computing. These features include being able to leverage the cloud platform for scalable infrastructure, handle scaling events without downtime or user experience degradation, and ability to scale proactively [67]. Network functions are designed for a static deployment in mobile networks and not for the dynamic environment of cloud computing where frequent scaling and redistribution activities can occur [59]. Furthermore, it is important to consider the resilience and adaptability of Virtualised Network Functions (VNFs) in this dynamic environment. This section describes architecture choices for virtualised elements in the EPC.

2.5.1 1:1 mapping

The simplest architecture for a virtualised EPC element is where its entire functionality is mapped to a single VM. This corresponds to a *1:1 mapping* [8]. In this architecture, the design of the state machine of the virtualised element and interfaces to other network devices remain the same as in current hardware-based solutions. The advantage of a 1:1 architecture is that it is conceptually similar to existing implementations and follows the same deployment model of one function on one device [8]. This allows for rapid and easy migration to the NFV approach for EPC networks.

However, this approach has several disadvantages as outlined in [8]. To explain these disadvantages, we consider the case of a 1:1 mapping for an

MME. First, in a cloud computing domain, a new VM is created when the number of subscribers increases beyond the running capacity of an existing MME. In such a case, each newly created virtual MME needs to be configured with EPC-specific parameters, including interfaces to other network elements and a globally unique identifier for the MME. Handling the dynamic configuration of many VMs may result in scalability issues for the Element Management System (EMS). Furthermore, the addition of a new MME requires informing other elements in the network. For instance, all eNodeBs serving an MME pool area have to be informed of the creation of a new MME. Secondly, once the VMs are configured and serving subscribers, they maintain information of active subscribers in their local storage. In case a VM is no longer required, for example due to reduction in subscriber numbers, a 1:1 mapping does not allow for a simple shutdown of the VM as active sessions will be affected. Although 3GPP specifications [17] allow transferring subscriber information between MMEs in an MME pool area through Overload or S1 Handover with MME Relocation procedures, this results in signalling overhead.

2.5.2 1:N mapping

A second architecture option is a 1:N mapping [8] as depicted in Figure 2.9. In this architecture, a network element is divided into three components - *front end* (FE), *worker* and *state database*. This follows from the three-tier architecture of web applications. The three components are described as follows:

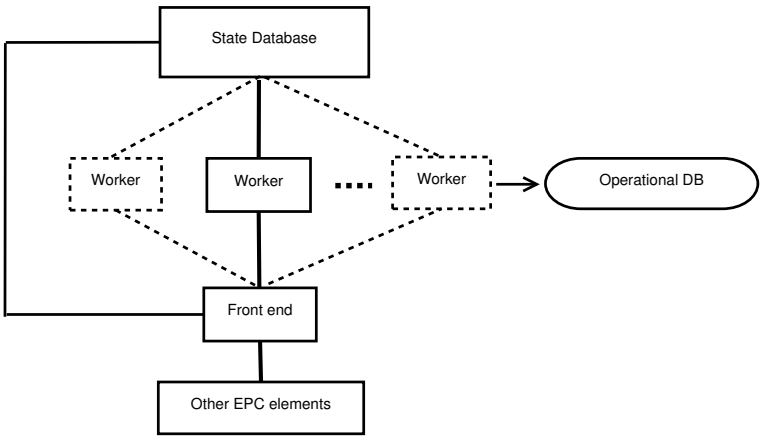


Figure 2.9: 1:N architecture diagram

- The FE maintains communication interfaces towards other elements in the network and balances requests to the workers, which handle the actual processing logic.
- Workers are stateless components which actually implement the functionality of the network element. Each worker is configured with the same network paramaters and is made stateless by moving user state information to the state database. The workers are logically connected to an operational data storage for storing log files and information related to basic operations and troubleshooting.
- The state database maintains user state information.

In this thesis, we choose this architecture for the MME due to the several benefits it offers [8, 60]. First, all workers are configured with the same parameters and in combination with the FE, can be visualized as a single network element. This simplifies operations and management functions as only one configuration scheme is required. Secondly, the workers being stateless can be independently started or stopped, i.e. scaled in or out, without impacting connected users. The change in state of the worker has to be informed only to the FE and is transparent to external network elements. Thirdly, a more granular control of load balancing is possible with this architecture which results in better utilisation of the processing capabilities of workers. Current load balancing techniques are based on weight factors in DNS queries or S1AP messages between the MME and eNodeB [17]. Dynamic changes of DNS records and increased S1AP messaging can be avoided with this architecture. Finally, this architecture leads to greater resilience as the stateless workers can fail with minimal effect on the user sessions.

While this approach offers several advantages, the FE could become a bottleneck for processing as all messages need to pass through the FE to the appropriate worker. A solution to circumvent this could be to implement multiple front end nodes. However, this increases complexity of the architecture and also requires appropriate load balancing procedures for the front end itself. Also, scaling the FE results in new IP addresses, which can lead to service interruption [58]. There may also be an increase in latency due to increased number of nodes through which each message has to pass [60]. Accessing the state database can result in increased network and CPU utilisation [68]. Additionally, synchronisation issues between different virtual components in this architecture can result in serialised access of the state database, which can cause lowered system performance [60].

2.5.3 N:1 mapping

In an N:1 mapping, all elements of the EPC network (such as MME, SGW, PGW, HSS) are virtualised and merged into a single virtualised component [8]. A subscriber or a group of subscribers is served by one EPC VM, which maintains subscriber state information in local storage. The only external interfaces for this component are towards the eNodeB and an external data network as depicted in Figure 2.10. Since the remaining 3GPP elements and interfaces are internal to the EPC VM, it is possible to optimise the processing and sending of messages between these internal components. Thus, this architecture option allows for a highly optimised software implementation of the complete EPC with low processing delays and minimal interfaces [60].

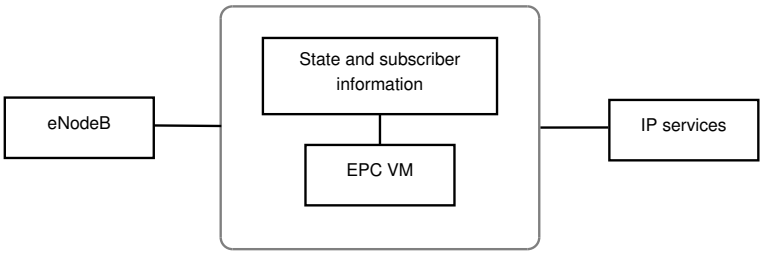


Figure 2.10: N:1 architecture diagram

However, the N:1 architecture option presents scalability problems for the EMS and other external components when maintaining and configuring a large number of EPC VMs. Secondly, the virtualisation of the HSS as part of the EPC VM means that the HSS handles only a part of the subscriber information. Thus, subscriber data management becomes complicated with this architecture. Thirdly, this architecture no longer allows for multi-vendor solutions (wherein different elements in the EPC are developed by different equipment vendors) and the entire EPC has to be provided by a single vendor [60].

2.5.4 N:2 mapping

An N:2 architecture is similar to the N:1 architecture, but with the control and data plane functionalities handled by different components. Figure 2.11 presents the N:2 mapping. There are three components in this architecture:

- A control (CTRL) component implements the complete control plane functionality of the EPC. This includes the functionality of MME, HSS and control plane functions of the SGW and PGW.

- A switch (SW) handles forwarding of user data packets, i.e. the user plane functions of the SGW and PGW. This component need not be virtualised and can thus utilise the high performance of hardware switches.
- A state database maintains the user state information, implemented either as an SQL or NoSQL database.

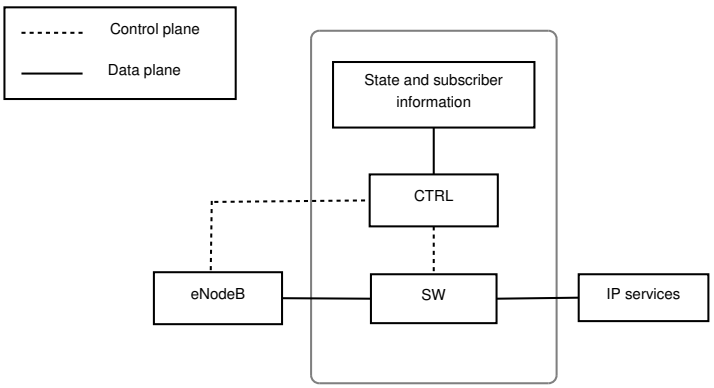


Figure 2.11: N:2 architecture diagram

This architecture option arises from the difference in requirements of data plane and control plane processing [60]. The control plane functionality demands low latency and fast computation, but the throughput requirements are not very high. However, the data plane functionality demands high data throughput, which can be provided by specialised hardware or with appropriate virtualisation technologies. The advantages and disadvantages of the N:2 mapping are similar to that of the N:1 architecture. An implementation of the N:2 architecture is discussed in [9].

2.6 Related work

Virtualisation and distributed design for network elements in the EPC and IMS are active areas of research. In this section, we first review existing literature in distributed design. We then describe the concept of a telecommunication cloud and review research in this area.

2.6.1 Distributed design for network elements

The authors of [25, 26] provide a design for a distributed MME with a reliable object store to save long term user state information. This architecture

separates the processing capabilities of the MME (distributed over several replicas) from the user state storage system. A one hop Distributed Hash Table (DHT) is used for membership management within the MME replicas. By moving the user information out of the MME replicas, they operate as independently as possible. This allows for scaling out or scaling in of replicas when required and migration of user states depending on the locality of the user. An *elastic core architecture* is described in [64], which separates state processing for virtualised network functions from the state information stored in a database. Software-Defined Networking (SDN) is used to flexibly allocate new resources for virtualised elements. The authors further analyse the application of this architecture to a virtualised IMS. [58] presents the application of elastic core architecture to an MME, which is able to provide session continuity during scaling operations.

IMS is an architectural framework designed to deliver multimedia services over IP protocol [21]. [34] discusses the application of different architecture patterns (including 1:1 and 1:N described in Section 2.5) to a virtualised IMS. Clearwater [2] is an open-source implementation of the IMS designed to run on the cloud. In this design, long term state information is moved out of the processing nodes to an external data store. Communication between the front-end and back-end components are enabled using Representational State Transfer (REST) commonly used in web applications. [50] presents an IMS distributed over nodes configured in a DHT overlay network. This architecture increases the robustness of the network elements and enables self-organisation of component nodes.

2.6.2 Moving telecommunication systems to the cloud

The convergence of cloud computing and mobile networks is often referred to as the *telco cloud* or *carrier cloud*. An overview of the telco cloud, its benefits, and challenges for implementation are provided in [10, 31, 59–61, 69]. The authors of [47] discuss the benefits and challenges of implementing virtual EPC networks. The authors propose grouping of virtualised elements in order to reduce control plane signalling between these elements. For example, the MME can be grouped with a front end for the HSS, thereby reducing network interactions. To build resilience and scalability into telecommunication systems, [51] proposes an architecture wherein a replica of each processing node is created and stored on other servers. This architecture allows for dynamic resource control and high availability. [48] discusses the feasibility of dynamic scaling in telco clouds. The authors describe a new architecture and protocols to migrate user sessions between stateful nodes in the IMS.

SDN is a complementary technology to NFV. Together, NFV and SDN,

are seen as key enablers for the telco cloud. SDN decouples the control and data plane and uses a centralised controller to make forwarding decisions. [38] presents a testbed architecture combining both NFV and SDN to demonstrate their applicability for future 5G networks. SDN can simplify the transport layer in mobile core architectures [38] and allow for flexibly controlling the network through software [64].

2.7 Summary

This chapter has introduced the background concepts to understand the work presented on the implementation of a virtualised distributed MME in EPC networks. We have therefore introduced three broad topics.

First, we have discussed the EPC, which forms the core of LTE networks. We have reviewed the EPC architecture and discussed the functionality of each network element to provide mobile data services to end users. We have then discussed in detail the control plane responsibilities of the MME. We have described two important interfaces; the S1-MME between the MME and the eNodeB, and the S11 between the MME and the SGW. We have also discussed the state information maintained by the MME on its local storage. Furthermore, two EPC procedures, the initial attach and UE-initiated detach, have been explained as these have been used to evaluate the behaviour of the distributed MME.

Secondly, we have introduced NFV, which is the virtualisation paradigm enabling network functions to be deployed on the cloud. While NFV leads to significant cost savings and allows network operators to harness the benefits of cloud computing, it also presents challenges in ensuring availability and performance.

Lastly, we have reviewed the architecture choices for virtualised elements in the EPC in order to address some of the challenges of NFV. We have identified the 1:N mapping or three-tier architecture to be most beneficial for the MME. This architecture enables the MME to scale efficiently as well as to be resilient to the failure of stateless workers. Furthermore, we have reviewed existing literature and identified similar examples of distributed architectures for IMS and MME. In addition to demonstrating the benefits of the new architecture, we aim to compare the distributed design to that of the original MME through an evaluation of latency during attach and detach procedures.

Chapter 3 describes our implementation of the three-tier architecture for the MME.

Chapter 3

Implementation

This thesis modifies and extends the MME software developed by Vicent Ferrer as part of his Master's thesis [45]. The original software corresponds to a 1:1 mapping or standalone architecture, wherein the entire MME functionality is implemented on a single VM. This design has disadvantages when deployed on the cloud, which are discussed in Section 2.5.1. We re-design the original MME software to correspond to a three-tier architecture or 1:N mapping described in Section 2.5.2. This new distributed architecture allows the MME to harness the benefits of cloud computing in terms of scalability and elasticity.

This chapter discusses the implementation details of the new architecture for the MME. Section 3.1 first provides an overview of the system design. Sections 3.2, 3.3 and 3.4 describe the functionality and implementation of the three tiers. In each of these sections, we also discuss the trade-offs between design choices made and considerations for further improvement.

3.1 Overview of system

Figure 3.1 depicts the overall system including the new design for the MME. The MME consists of three main components – a *front end* (FE), one or more *workers* and a *state database*. The FE behaves as an intelligent proxy and maintains interfaces to other elements in the EPC network. The workers are responsible for the actual functional processing in handling user mobility and sessions. The UE context for each user is stored in the state database, thereby making the workers stateless. Thus, the workers can be easily scaled out or scaled in depending on network load. This also makes the MME resilient to the failure of workers as user state information is saved on the state database. Any worker with access to the UE context can take over the

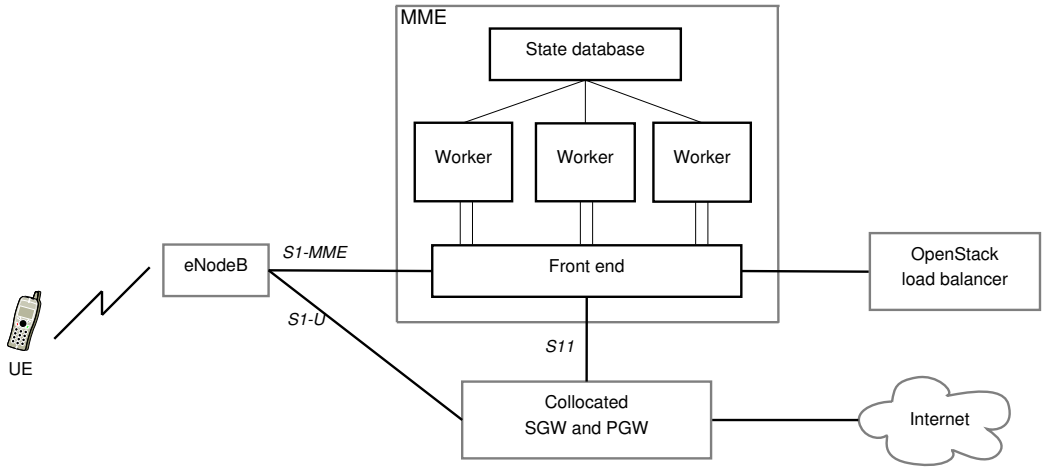


Figure 3.1: Architecture of system

processing in case of failures.

A combination of the three tiers represents a single MME to external elements such as the eNodeBs and SGW/PGW. The workers and state database are transparent to other EPS entities, as all interfaces to external elements terminate at the FE. In addition to the EPS elements (eNodeB, MME, SGW, PGW), the system includes an *OpenStack load balancer* developed as part of [23]. This component is responsible for creating and deleting worker VMs on OpenStack. The functionality of the load balancer corresponds to the roles of the VNF Manager and NFV Infrastructure Manager [43].

3.2 Design of front end (FE)

The main functions of the FE are to:

1. Maintain 3GPP standardised interfaces towards other EPC network elements – In our system, the FE maintains an S1-MME interface towards eNodeBs and an S11 interface towards SGWs.

2. Balance requests between worker nodes – Our FE design employs a simple round-robin balancing scheme to distribute new requests to worker nodes.

3. Inform the OpenStack load balancer when new workers are required to be created or deleted

In order to realise the first two functions, the FE needs to correctly forward S1AP messages between workers and eNodeBs, and GTP messages

between workers and SGWs. To identify the MME worker, eNodeB and SGW responsible for handling a particular user session, the FE maintains a mapping which associates the UE with each of these elements. The FE determines the UE identity based on Information Elements (IEs) present in the message to be forwarded. Sections 3.2.1 and 3.2.2 describe this mapping in further detail. Section 3.2.3 discusses the interface between the FE and OpenStack load balancer.

3.2.1 Forwarding S1AP messages

In our design, the IEs used to identify a UE over the S1-MME interface are the MME UE S1AP ID and eNodeB UE S1AP ID in S1AP messages, and the IMSI in NAS messages. The MME UE S1AP ID and eNodeB UE S1AP ID are unique identifiers for a UE assigned by the MME and eNodeB respectively. When a UE first attaches to the LTE network, it identifies itself with its IMSI in the "Attach Request" message. This message also contains the eNodeB UE S1AP ID. The FE chooses an appropriate worker to forward the request to and maps the UE to the selected worker, as well as to the eNodeB from which the request was received. Once the worker assigns an MME UE S1AP ID, all further S1AP messages to and from the UE contain this IE. Subsequently the FE can appropriately forward messages between the eNodeB and worker node based on the the S1AP IDs.

3.2.2 Forwarding GTP messages

All control plane messages between the MME and SGW related to a specific UE are sent over a unique GTP tunnel. Each endpoint maintains a Tunnel Endpoint ID (TEID), an IP address and UDP port number to identify the tunnel. During establishment of PDN connectivity, the MME worker first sends a "Create Session Request" which includes the UE's IMSI and a TEID generated by the worker. We refer to this TEID as the MME TEID. On receiving this message, the FE saves the mapping between the UE and MME TEID. It then chooses an SGW and forwards the message to it. In the corresponding "Create Session Response", the SGW includes its own TEID (referred to as the SGW TEID) and sets the receiver TEID as the MME TEID. Upon receiving this message, the FE forwards it to the worker based on the mapped MME TEID. It also saves the SGW TEID to the UE's mapping structure. The FE now uses the GTP TEIDs present in all subsequent GTP messages to identify the appropriate SGW and worker.

[15] requires that GTP sequence numbers are maintained per sending queue, i.e. for each triplet of local IP address, local UDP port and remote

peer's IP address. Thus, the FE appropriately updates the sequence number in each outgoing GTP message. To summarise, Table 3.1 lists the messages in the implemented call flow procedures and the corresponding S1AP and GTP IEs used to identify the UE.

3.2.3 Communicating with the OpenStack load balancer

The FE maintains a long-lived TCP connection to the HTTP server of the OpenStack load balancer. When the number of incoming attach requests per worker goes above or below a certain threshold, the FE sends an HTTP request to the load balancer requesting either the creation or deletion of a worker. Creating a new worker corresponds to a scaling out operation. Once a new VM is created and the worker is active, it initiates an SCTP association to the FE. The FE then adds the newly created worker to its list of available workers and forwards new requests to it. The scaling in operation, i.e. deleting workers, requires more careful consideration, so as not to effect any on-going procedures. When the incoming call rate goes below a certain threshold, the FE first marks a worker for deletion and stops forwarding any new procedure requests to this node. However, messages from ongoing procedures are still forwarded to and from this worker so as not to disrupt the user session. The FE then waits for a configurable time period to allow ongoing call flows to complete and the state information for the UEs to be written to the state database. After this time period has elapsed, the FE sends an HTTP delete request to the load balancer with the IP address of the worker marked for deletion.

3.2.4 Design considerations

The performance of the overall MME is dependent on the processing capabilities of the FE. The FE could become a potential bottleneck and single point of failure in the system. In a similar design for a virtualised IMS, [48] proposes deploying multiple Client Elasticity Gateways (CEGs) close to the client. The CEGs are similar in functionality to the FE and such a solution can be used for the FE as well. Additionally, the FE is a stateful entity as it maintains a mapping structure to forward messages between the worker and UE/SGW. To avoid maintaining this information, [58] proposes creating groupings of IMSIs based on hash value and assigning a worker instance to a group of UEs. However, this limits the scalability of the worker nodes, as there should always be atleast one worker available to serve each group of UEs. An alternative solution could be to have the FE query the state

Message	**Protocol**	**Information Elements (IEs) present to identify UE**
Attach procedure		
Attach Request	S1AP	eNodeB (eNB) UE S1AP ID
	NAS	IMSI
Authentication Request	S1AP	eNB UE S1AP ID, MME UE S1AP ID
Authentication Response	S1AP	eNB UE S1AP ID, MME UE S1AP ID
Create Session Request	GTP	TEID to send to = 0 (SGW TEID unknown), IMSI, MME TEID
Create Session Response	GTP	TEID to send to = MME TEID, SGW TEID
Attach Accept	S1AP	eNB UE S1AP ID, MME UE S1AP ID
	NAS	GUTI
Initial Context Setup Response	S1AP	eNB UE S1AP ID, MME UE S1AP ID
Attach Complete	S1AP	eNB UE S1AP ID, MME UE S1AP ID
Modify Bearer Request	GTP	TEID to send to = SGW TEID
Modify Bearer Response	GTP	TEID to send to = MME TEID
Detach procedure		
Detach Request	S1AP	eNB UE S1AP ID, MME UE S1AP ID
	NAS	IMSI or GUTI
Delete Bearer Request	GTP	TEID to send to = SGW TEID
Delete Bearer Response	GTP	TEID to send to = MME TEID
UE Context Release Command	S1AP	eNB UE S1AP ID, MME UE S1AP ID
UE Context Release Complete	S1AP	eNB UE S1AP ID, MME UE S1AP ID

Table 3.1: Important IEs present in messages exchanged during attach and detach procedures

database to identify the worker currently processing a particular UE. This requires more careful design so as to minimise any increase in latency due to database queries. Also, in our current prototype, the MME does not perform NAS ciphering and integrity protection. This requires further logic to be implemented on the FE.

3.3 Design of worker

The design of the worker follows the same principles of the original MME software and is described in [45]. The worker represents the actual functionality of the MME and handles the processing of call flows. Each worker maintains two separate interfaces towards the FE, one for S1AP messaging and the other for GTP messaging. The HSS is implemented as a MySQL database and is co-located with the worker. Thus, the S6a interface is not realised. The open-source C client library hiredis[1] is used by the worker to interface to the Redis server.

3.3.1 Attach procedure

This section describes the implementation of the attach procedure on the worker. The procedure is similar to that described in Section 2.3.1. In order to make the workers stateless, we introduce additional messages to store the UE context on the state database. Figure 3.2 shows the attach call flow implemented in this thesis and can be compared to that in Figure 2.6. For simplicity, the FE is not depicted in Figure 3.2 as it simply forwards messages between the eNodeB and worker. To minimize processing time, the state information is not stored at each step of the call flow but only towards the end of the procedure. The worker stores the UE context at step 8, on receiving a successful "Create Session Response" from the SGW. At this point, the user is considered to be in the registered state and successfully attached to the LTE network.

3.3.2 Detach procedure

Once the UE context has been stored in the state database, any worker can handle further procedures for the registered UE after retrieving the context information from the database. For example, in the detach procedure depicted in Figure 3.3, the worker first queries the state database to retrieve

[1] `https://github.com/redis/hiredis`

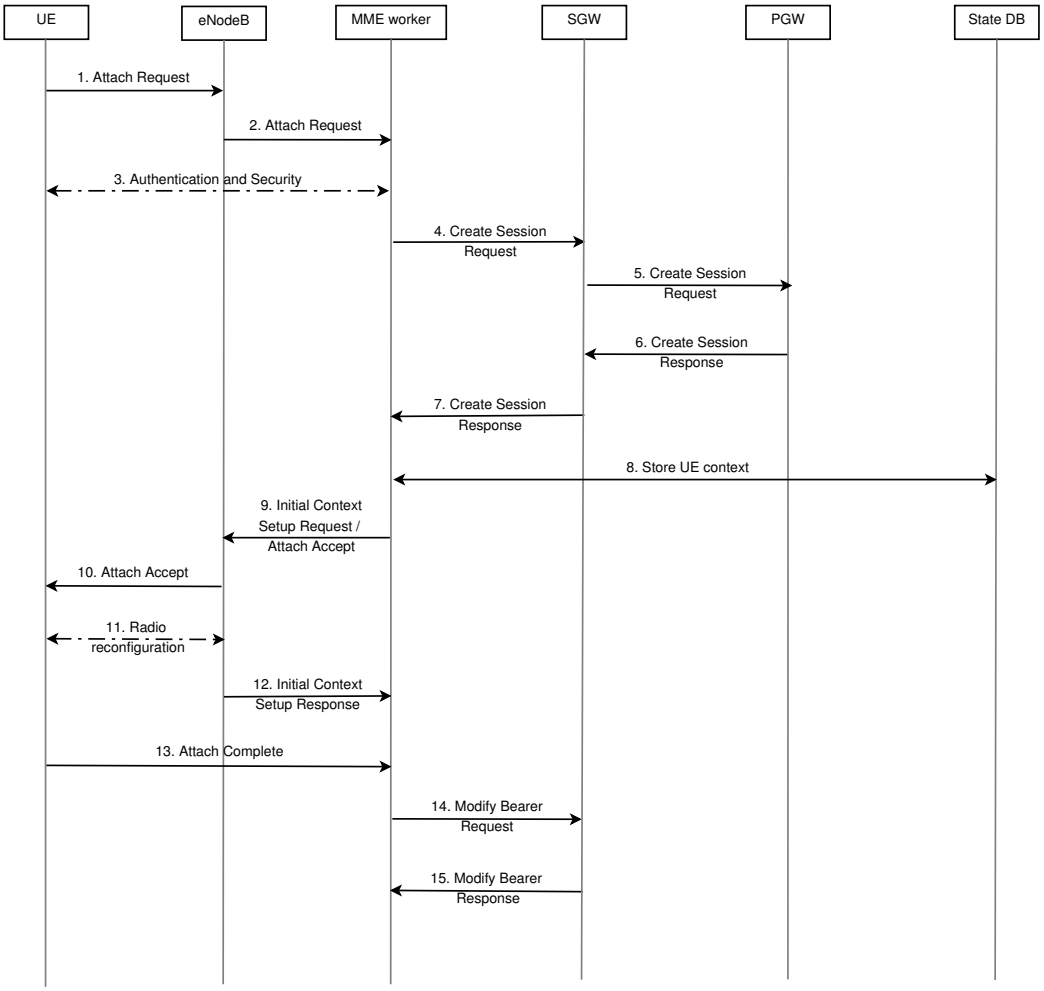


Figure 3.2: Attach call flow implemented on worker

the UE context. Based on the state information, the worker can now handle the processing of the user session and detach the subscriber.

3.3.3 Design considerations

Storing the UE context at each step in the call flow can help to increase the resilience of the MME. In our current implementation, if a worker fails during the processing of a call flow, the procedure has to be re-initiated. However, storing the context at each step leads to increased CPU utilisation and network overhead with external database queries. By choosing to store UE context only after a call flow is complete, there is a trade-off between low latency with fewer database queries and decreased resilience. [58] provides

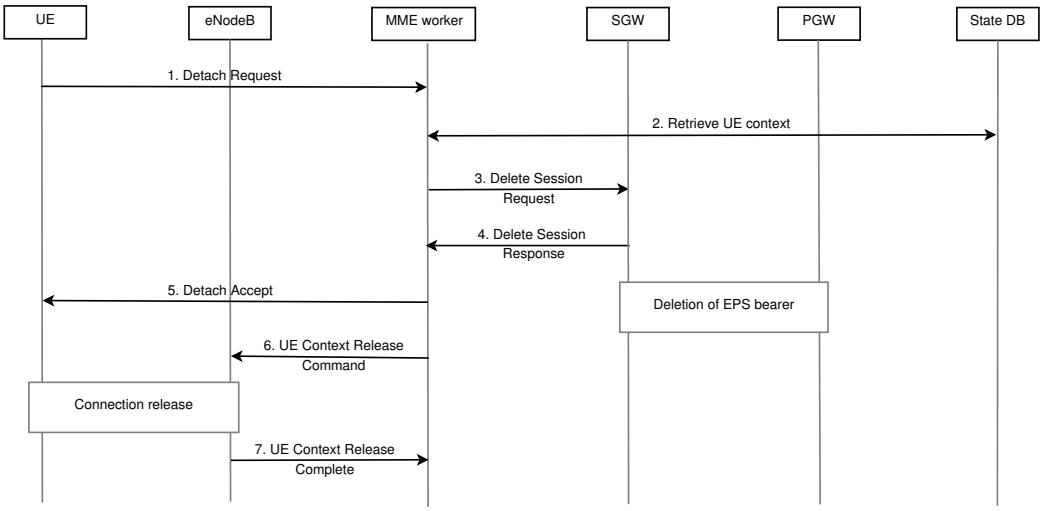


Figure 3.3: Detach call flow implemented on worker

an analysis of the impact of losing state information during a call flow. For example, if state information is lost during an attach procedure, the procedure has to be re-executed by the UE. Similar analysis is carried out for other 3GPP procedures. In [51], the authors propose synchronisation of session states for a distributed IMS only at certain stages, so as not to degrade performance while maintaining an acceptable level of call dropping probability. In a three-tier architecture for a virtualised IMS, [68] proposes using a cache on the client (equivalent to the worker, in our case) on which the application stores serialized data. The state information is eventually stored in the state database. Read operations first attempt to retrieve data from the cache before querying the state database. With this caching mechanism, some of the negative effects of moving long-term state information out of the worker can be reduced.

3.4 Design of state database

The state database is the third tier in the MME architecture and stores state information for each UE attached to the MME. This state information is referred to as the UE context and is described in Section 2.2.2. Three options for implementing the state database are [8]:

- Centralised database – A centralised database can be used to serve a single data center where virtual MME instances are deployed.

- Distributed database – A distributed database or data store can be deployed independent of the virtual instances.
- Distributed filesystem – A distributed file system such as Ceph[2] can be used to store subscriber information. [9] provides details of the implementation and evaluation of its suitability for storing shared state information for virtualised EPC elements.

A distributed design for the state database is possible because the state information for each user is small (in the order of a few kilobytes) and high bandwidth links are available in mobile backhaul networks [26]. We choose a NoSQL data store due to its simplicity of use, high availability and scalable nature [35]. The UE context can be stored in a NoSQL data store as a key-value pair, with the key being a unique identifier. Additionally, only one client (the worker in our case) will access the state information for a particular UE at a time. This further allows for the use of a distributed data store without the need for exclusive access control [68]. There are several distributed data stores available for use, including Apache Cassandra[3], Riak[4] and memcached[5].

3.4.1 Redis cluster

We choose Redis to implement the state database. Redis is an open source, in-memory key-value data store[6]. The in-memory feature ensures that it operates with very low latency. The Redis cluster feature (introduced since Redis 3.0 [5]) allows for sharding data across multiple nodes. Furthermore, its simplicity of use and availability of a client library in C [4] make it an ideal choice.

Communication between the MME worker (Redis client) and Redis server takes place over a TCP connection and uses Redis Serialization Protocol (RESP) [7]. The communication follows a Request/Response pattern. A Redis cluster uses asynchronous replication, which means that the cluster responds to a write operation before it is replicated on a slave. Although there is a small window during which a Redis cluster may lose writes [5] and thereby user state information, this loss can be handled by existing procedures in 3GPP standards. For instance, 3GPP standards are designed to allow for recovery of UE context information or re-initiation of procedures

[2] `http://ceph.com`
[3] `http://cassandra.apache.org`
[4] `http://basho.com/riak`
[5] `http://memcached.org/about`
[6] `http://redis.io`

in case of inconsistencies or loss of information (for example, in cases of discontinuous network coverage) [26]. The Redis cluster used in our system consists of three master nodes. The UE context is stored in binary format on the cluster with the associated key MME UE S1AP ID. The UE context contains the elements listed in Table 2.1.

3.4.2 Redis persistence policies

Redis persistence policies[7] enable the data stored on a Redis server to be logged or saved on local disk storage. This makes it is possible to recover data in case a Redis server fails. There are two persistence policies available – RDB snapshotting and Append Only File (AOF). We use both policies in our implementation of the Redis cluster. With RDB persistence, a snapshot of the dataset is taken at regular configurable time intervals.

The AOF feature allows for durable operation of the database by logging every modification to the dataset in an "append only" file stored on local storage. However, this feature increases latency during write operations to the database. Modifications to the file are flushed to the disk using write() and fsync() operations on the server [6]. Once the write() call is executed, data is committed to the kernel buffers and can be recovered in the case of process failures [6]. With the fsync() operation, data is committed to the disk and it is possible to recover data even in the case of a complete server failure, for example, due to a power outage [6]. However, the fsync() call is a blocking operation and can delay write operations to the data store. Thus, Redis provides three configuration options for AOF feature in order to control fsync() and write() operations. These configurations are as listed below.

1. fsync always – For every write to the Redis server, data is written to the file and flushed to the disk before acknowledging the client. This provides the best durability and also results in highest latency for write operations.

2. fsync every second – Data is both written to the file and flushed to the disk every second.

3. fsync no – In this case, the Redis server acknowledges the client after the change is transferred to the append only file using write(). Data is transferred to storage with an fsync() call when decided by the Operating System (OS).

[7] `http://redis.io/topics/persistence`

In our implementation, we choose the fsync every second configuration to provide reasonable durability without significantly affecting latency.

3.4.3 Design considerations

An alternative choice for the key (in the key-value pair) is the GUTI, which is a unique identifier for the UE regardless of its current state in the network. When further call flow procedures are implemented on the MME, this identifier may be required to identify the worker handling the user session.

Additionally, a vector clock can be used to track modifications to the UE context stored in the data store. A vector clock is a timestamping mechanism used to determine the order in which events occur in a distributed application [28, 55]. The vector clock consists of a pair of values, the worker ID and a monotonically increasing counter for any modification made. This can be used by the worker to resolve any inconsistencies in the data retrieved from the data store.

[35] provides a systematic evaluation of NoSQL data stores and its suitability for telecommunication applications. The authors conclude that the average latencies for read/write operations are within the requirements of a telecommunications system, but exhibit an unacceptable number of outliers where latencies can be as high as 200ms. Based on their observations, they develop a system called Flurry, which relies on the first response from a replica, rather than waiting for a response from a minimum number (quorum) of replicas.

3.5 Summary

This chapter has presented the design and implementation of a three-tier architecture for a distributed MME. The three tiers are the FE, workers and external state database.

The FE behaves as an intelligent proxy and maintains interfaces to other elements in the EPC network. We have described the message fields used by the FE to correctly forward messages between external network elements and the workers. We have discussed how the FE decides when scaling operations for the workers are required. We have also presented some drawbacks of using a FE and methods to mitigate these.

The workers are the components that handle the actual call flow processing. We have described how the call flows are implemented on the worker and when the UE state information is stored on the external Redis data store. We have decided to store the state information only towards the end of a

call flow procedure in order to minimise the number of messages sent over the network. This results in an acceptable level of resilience to the failure of workers during call flows.

Finally, we have discussed the Redis data store used to implement the state database. In particular, we have highlighted the persistence policies used on Redis to achieve durability of the data. These policies increase the latency of write operations. We further test these policies to evaluate their effects on latency of EPC procedures.

The next chapter presents the experimental testbed and describes the experiments carried out to evaluate the behaviour of the distributed MME.

Chapter 4

Experimental setup and plan

We deploy and test the distributed MME on two independent OpenStack installations. The OpenStack testbed is set up within Aalto University as part of joint research efforts between the Department of Computer Science and Engineering and Department of Communications and Networking.

This chapter describes the testbed and experiments performed to evaluate the MME. Section 4.1 describes the OpenStack installation and the underlying physical hardware. Section 4.2 presents the overall architecture of the testbed and software components deployed as VMs on the OpenStack clouds. Finally, Section 4.3 presents the experimental plan.

4.1 OpenStack

OpenStack is an open source cloud operating system that manages pools of compute, storage and networking resources in a data center[1]. These resources can be managed through a dashboard on the web or through OpenStack APIs. The testbed used in this thesis was originally set up as part of a Master's thesis at Aalto University [65]. The original testbed consisted of one OpenStack cloud, running Icehouse release 2014.1.3[2]. The testbed has since been expanded to include a second OpenStack installation with the same Icehouse version. A detailed description of the OpenStack services and configuration can be found in [65] and [23].

Each OpenStack installation consists of four identical blade servers. All the blade servers run Ubuntu 14.04 and their hardware properties are listed in Table 4.1. The servers are used as follows:

[1] `http://www.openstack.org/software`

[2] `https://wiki.openstack.org/wiki/Releases`

CPU	2 x Intel Xeon E5-2665 (2.4 GHz, 64-bit, 8 cores, Hyper-Threading enabled)
RAM	128 GB DDR3 1600 MHz
Hard disk space	150 GB
Networking	10GbE interconnect

Table 4.1: Hardware properties of blade servers (HP ProLiant BL460c Gen8)

- We configure two blade servers as compute hosts on which VMs are run. The compute hosts use a Kernel-based Virtual Machine (KVM)[3] hypervisor.
- The OpenStack controller runs on a third blade server. The controller provides a dashboard to manage project properties and to launch, terminate and manage VMs.
- We deploy networking services on the fourth blade server. This provides network connectivity between VMs using an Open vSwitch[4] virtual switch and allows for mapping the physical networks to the virtual networks. OVS is configured with a hybrid setup to allow both Generic Routing Encapsulation (GRE) tunnels and Virtual Local Area Networks (VLANs).

Our test environment consists of VLAN-based virtual networks which allows us to separate traffic within the VMs deployed in the testbed. Three VLAN networks are configured on both installations (with the same VLAN tags), namely *openstack*, *ltemgnt* and *lteuser*. The openstack network is used for S1AP messages and communication to the Redis nodes. The ltemgnt network is used for GTP messaging, and the lteuser network is used for user data traffic. Additionally, a GRE network *ue net* is configured to provide Internet connectivity to route traffic between the SGW and public networks as well as to support UE addressing. The ue net network is not used in the experiments performed in this thesis.

4.2 Software components and VMs

Figure 4.1 shows an overview of the testbed with the VMs deployed on the two OpenStack installations. The following VMs are deployed:

[3] `http://www.linux-kvm.org/page/Main_Page`
[4] `http://openvswitch.org`

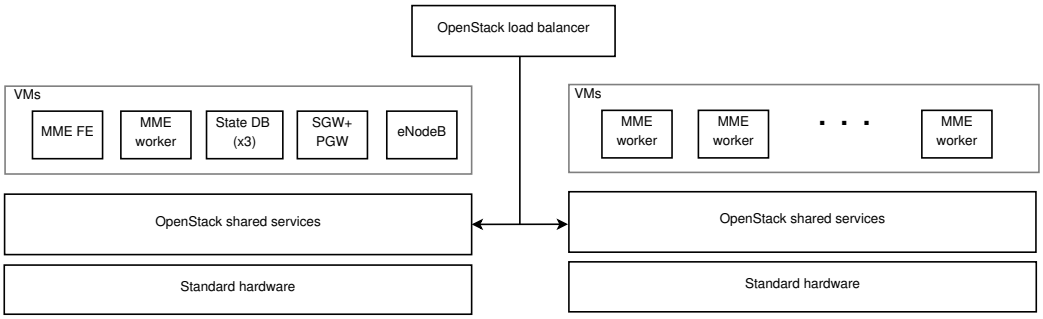


Figure 4.1: Overview of testbed comprising of VMs deployed on two OpenStack clouds

- **FE** – The FE of the distributed MME is deployed on one VM and contains EPC-specific configurations for the MME.
- **Worker** – Each MME worker is deployed on a separate VM and configured to connect to the FE. One OpenStack cloud is dedicated to running MME workers, although workers can be deployed on either OpenStack installation.
- **State database** – The state database consists of three Redis servers, each on a different VM and configured as a cluster of three master nodes. Redis 3.0.1 is used on all three VMs.
- **eNodeB** – The eNodeB is a simple C program developed to test the MME application as part of [45]. It sequentially sends messages required to test attach and detach procedures.
- **SGW and PGW** – An open source implementation of the gateways [1] is used to run a collocated SGW and PGW on a VM.
- **Standalone MME** – The MME developed in [45] represents the original standalone MME and is deployed on a VM (not shown in figure).

All components except the state database are run on VMs with *m1.small* flavor. The Redis servers are run on VMs with *m1.medium* flavor as recommended in [3]. Additionally we modify the kernel settings for the Redis VMs as suggested in [3]. All VMs run Ubuntu 14.04 and the characteristics of the flavors used in our testbed are listed in Table 4.2.

Flavor	**vCPU(s)**	**RAM**	**Disk space**
m1.small	1	2048 MB	10 GB
m1.medium	2	4096 MB	20 GB

Table 4.2: Characteristics of small and medium flavor

4.3 Experiments

We evaluate the behaviour of the distributed architecture for the MME with the experiments described below. Experiments 1 to 4 measure latency, experiment 5 demonstrates the scalability of our architecture and experiment 6 demonstrates the resilience of the architecture to failures.

1. Measure attach latency for distributed MME and standalone MME – We measure attach latency on the eNodeB as the time elapsed between the eNodeB sending an Attach Request and receiving an Attach Accept. We capture packets on the eNodeB VM using *tcpdump* and calculate attach latency using a Lua script. We run this experiment for both the distributed and standalone MME to compare their average latency.

2. Measure attach latency for different placement configurations of FE and worker – We measure attach latency (as described in previous experiment) for the distributed MME with the following placement configurations:

   (a) Worker and FE on different OpenStack installations

   (b) Worker and FE on same compute host in same OpenStack installation

   (c) Worker and FE on different compute hosts in same OpenStack installation

3. Measure time taken to retrieve UE context from the state database – The aim of this experiment is to compare the time taken to retrieve UE context in the distributed MME to that taken by the standalone MME. For the distributed MME, we measure the retrieval time on the worker. The measured value includes the time taken to send a request over the network, execute a query on the Redis server and receive a response from the Redis server. On the standalone MME, UE context is stored in memory as a C structure in a uthash[5] hash table. The time measured is simply the retrieval time from the hash table.

[5] `https://troydhanson.github.io/uthash`

4. Measure attach latency for different persistence settings on the state database – Section 3.4 describes the configuration of persistence feature on the Redis server. To evaluate the effect of the AOF feature on attach latency, we measure the time taken for write operations on the Redis server with the following settings:

   (a) AOF disabled – The AOF feature is disabled and an append only log file is not maintained.

   (b) fsync no – The Redis server acknowledges the client immediately after the change is written to the append only file. The OS on the Redis server decides when to flush the data to disk.

   (c) fsync every second – Data is written to the log file and flushed to the disk once every second.

   (d) fsync always – Every modification to the dataset is first logged in the AOF file and flushed to the disk storage before the server acknowledges the client.

5. Demonstrate autonomous scaling out and scaling in of workers based on incoming call rate – We test autonomous scaling by varying the rate at which attach requests are sent from the eNodeB. When the incoming rate of attach requests is above a certain threshold, the FE requests the creation of a new worker. When the incoming call rate is lowered, a worker is deleted. We capture logs on the FE and worker to monitor the times at which attach requests are received and HTTP requests are sent to create or delete workers.

6. Demonstrate resilience – In order to demonstrate resilience, we attach a user to the distributed MME with two active workers and then manually shut down the worker handling the user session. We then proceed to initiate a detach procedure for the UE. We capture logs on the FE and the active worker to verify that the detach procedure completes successfully.

4.4 Summary

This chapter has presented details of the testbed and the experiments performed to evaluate the distributed MME.

We have used OpenStack as our cloud computing platform and have set up two OpenStack clouds. Each cloud consists of two physical compute hosts on which VMs are deployed. Furthermore, we have described the placement

of the FE, worker and state database VMs on the two clouds. Apart from the EPC components, the testbed contains a load balancer which creates and deletes worker VMs on the two OpenStack clouds.

We have also presented our experimental plan consisting of six experiments. With a distributed architecture, latency is expected to be the main bottleneck. Experiment 1 compares the attach latency of the distributed MME to that of a standalone MME. Experiment 2 further examines attach latency of the distributed MME with different placement configurations of the FE and workers. Experiment 3 measures the time taken to retrieve UE context from the state database. Experiment 4 measures the attach latency with the different Redis persistence policies. Experiment 5 demonstrates autonomous scaling of workers and lastly, experiment 6 demonstrates resilience of the architecture to worker VM failures.

Chapter 5 now presents the results of the experiments.

Chapter 5

Evaluation

This chapter presents the results of our experiments. The six experiments performed are listed in Section 4.3.

5.1 Attach latency compared to original MME

Table 5.1 presents the average latency of attach procedure on both the standalone and distributed MME. Each experiment consists of attaching 50 UEs sequentially at a call rate of 1 call per second. The latency for the distributed MME is the average of results obtained from running five sets of experiments for each of the following placement configurations:

(a) Worker and FE on different OpenStack installations

(b) Worker and FE on same compute host in same OpenStack installation

(c) Worker and FE on different compute hosts in same OpenStack installation

The latency for the original MME is calculated as average of the results obtained from running ten sets of experiments.

Type of MME	**Average latency**	**Average latency with 95% confidence interval**
Standalone	8.399 ms	7.836 ms to 8.962 ms
Distributed	12.782 ms	12.574 ms to 12.990 ms

Table 5.1: Average of measured attach latency for distributed and original standalone MME

The increase in attach latency for the distributed MME is on average 4.4 milliseconds. This increase in latency can be attributed to the exchange of messages between the FE and worker. Each message exchanged between the worker and an external network element has to be forwarded through the FE. Thus, the network latency between the components of the distributed MME significantly affects measured attach latency.

5.2 Attach latency depending on placement of FE and worker

As network performance greatly affects the attach latency for a distributed MME, we further compare attach latency for different placements of FE and workers. These placement configuration, labelled (a), (b) and (c), are listed in Section 5.1. Figure 5.1 presents the cumulative distribution function (CDF) of measured attach latency and Table 5.2 presents the average attach latency for the three different placement configurations.

The measured latency for case (b) wherein the FE and worker are on the same compute host shows the lowest average value of 12.368 milliseconds. In this configuration the lowest time measured is 6.394 milliseconds and the measured maximum value shows a ten-fold increase to 65.553 milliseconds. The reason for this high outlier is not fully understood, but is seen more often in this placement configuration than in other cases. The fluctuation in attach latency is represented by the high standard deviation, as listed in Table 5.2. The distributions of attach latency for cases (a) and (c), wherein the FE and worker are on different physical compute hosts, are quite similar. The maximum measured attach latency does not exceed 25 milliseconds in both these cases.

Configuration	**Average latency**	**Average latency with 95% confidence interval**	**Standard deviation**
(a)	12.914 ms	12.691 ms to 13.136 ms	1.779
(b)	12.368 ms	11.863 ms to 12.872 ms	4.041
(c)	13.065 ms	12.777 ms to 13.353 ms	2.307

Table 5.2: Attach latency for placement configurations of FE and worker on (a) different OpenStack installations; (b) same compute host, same OpenStack installation; (c) different compute hosts, same OpenStack installation

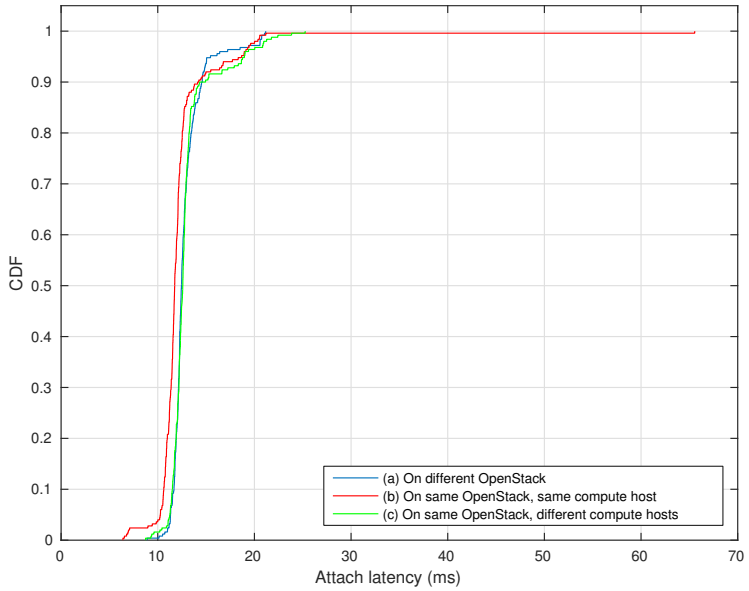


Figure 5.1: CDF of attach latency measured for three different placement configurations of FE and worker

5.3 UE context retrieval time

We measure the time taken to retrieve UE context for both the distributed and standalone MME. Table 5.3 presents the average value of 250 retrievals. For the distributed MME, the main contributor to the measured time is the network latency between the worker and Redis server. The time taken for execution of commands on the Redis server (monitored using *redis slowlog*[1]) is in the order of a few microseconds.

Type of MME	**Average time**	**Average time with 95% confidence interval**
Standalone	20.7 μs	20.025 μs to 21.375 μs
Distributed	1256.724 μs	1238.695 μs to 1274.752 μs

Table 5.3: Average retrieval time for UE context on distributed and original MME

[1]http://redis.io/commands/slowlog

5.4 Effect of Redis persistence policy on attach latency

We measure the time taken to write UE context to the Redis server with each of the four different persistence policies listed in Section 4.3. The measured value includes the time to send a request from the worker to the Redis server, execute the write operation on the Redis server and receive an acknowledgement from the Redis server. Figure 5.2 presents the average time to write to the Redis data store for these persistence policies. Each value is an average of 250 write operations. In the case of *fsync always* policy, the average write time is the highest as the worker (Redis client) receives a response only after the modification is flushed to local storage. The lowest average time for a write operation is seen in the case where the append only policy is disabled. For persistence policies *fsync no* and *fsync everysec*, the average time to write to the data store is similar.

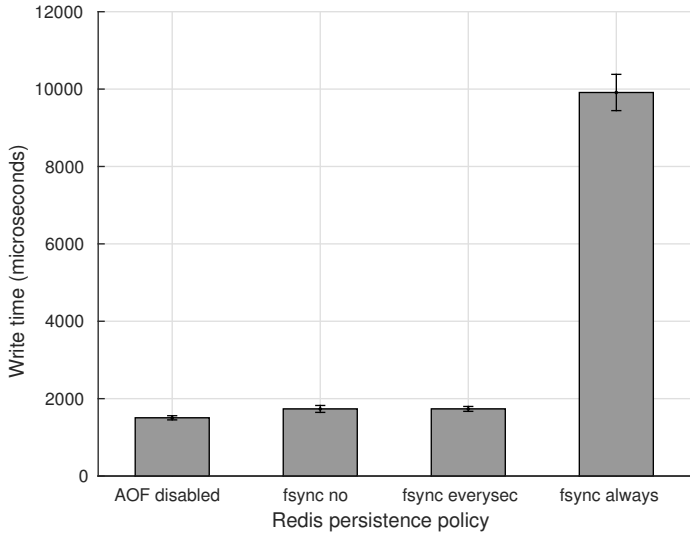


Figure 5.2: Average time for write operations on Redis server (±95% confidence interval) for four different Redis persistence policies

As the *fsync always* policy results in an unacceptable latency, we do not further test this policy. To investigate the behaviour with higher loads on the Redis servers, we repeat the experiments with all three Redis servers of the cluster on the same VM. The experiments are repeated with AOF disabled, with policy *fsync everysec* applied and *fsync no* applied. Figure 5.3 presents the scatter plot of measured write times and Table 5.4 presents the average write time and standard deviation for these experiments. As

Redis Configuration	Average time	Average time with 95% confidence interval	Standard deviation
AOF disabled	1434.472 μs	1413.010 μs to 1455.934 μs	171.954
fsync everysec	1611.320 μs	1576.418 μs to 1646.222 μs	279.634
fsync no	1754.936 μs	1646.133 μs to 1863.738 μs	869.946

Table 5.4: Average time for write operations on Redis server (when all Redis servers are on same VM) for three different persistence policies

expected, the average write time is lowest for the case where the AOF feature is disabled. As seen in Figure 5.3(a), there are only three points where the write time exceeds 2000 microseconds. This figure can be considered as a baseline to understand the effect of using AOF policies. With policy "fsync everysec", the average latency is higher. There are more occurrences of latency exceeding 2000 microseconds in Figure 5.3(b) as compared to Figure 5.3(a). With policy "fsync no" applied, the average latency increases by 143 microseconds as compared to "fsync everysec". However, Figure 5.3(c) shows a higher frequency and magnitude of outliers exceeding 2000 microseconds.

5.5 Demonstration of autonomous scaling on distributed MME

Figure 5.4 presents the autonomous scaling of workers depending on the rate of incoming attach requests. We start the experiment with only one worker. When the rate of incoming attach requests is one call per second, the FE sends an HTTP request to the load balancer to create a new worker VM at time 07:34:01. Once the VM boots up and the worker is started, the worker initiates an SCTP association to the FE. This occurs at time 07:34:28 when the worker endpoint is added to the FE. From this point, attach requests are forwarded to the new worker based on a round-robin balancing scheme between the two workers. As can be seen from the figure, in our testbed it takes at least 25 seconds for the VM to be started and worker to be running.

When the rate of incoming attach requests reduce, the second worker is marked for deletion at time 07:35:15. At this point the FE stops forwarding any new attach requests to the worker marked for deletion. All further attach requests are forwarded to the first worker. After five seconds have elapsed (a duration configured in the FE code), the FE sends an HTTP request to the load balancer to delete the VM. The first worker continues to process any incoming procedures.

5.6 Demonstration of resilience

In this experiment, the distributed MME is started with two active workers. After the UE attaches to one worker, we manually stop the worker to which the UE is attached. We then initiate a detach procedure for the UE. The detach request is forwarded to the available active worker. From the logs on the worker, we verify the UE context is retrieved from the state database and the callflow proceeds successfully. The log file is not presented in this thesis.

5.7 Summary

This chapter has presented the results of the experiments listed in Section 4.3. We have successfully demonstrated a working prototype of the distributed MME and have compared its behaviour to a standalone MME.

In a distributed MME, the additional messaging between the FE and workers contributes to an increase in latency during attach procedures. We find that the increase in attach latency for the distributed MME is on average 4.4 milliseconds. Furthermore, we have identified variations in attach latency when the FE and workers of the distributed MME are deployed on the same physical compute host or different compute hosts. The measured latency is the least when the worker and the FE reside on the same compute host but the standard deviation of the measured values is also higher. With the FE and workers on different compute hosts, the latency increases marginally as compared to the same physical host. However, the latency measured is more predictable, with a smaller standard deviation in measurements.

The time taken to retrieve the UE context increases in the distributed MME as compared to the standalone MME. This increase in latency is mainly attributed to the request and response between the worker and Redis server.

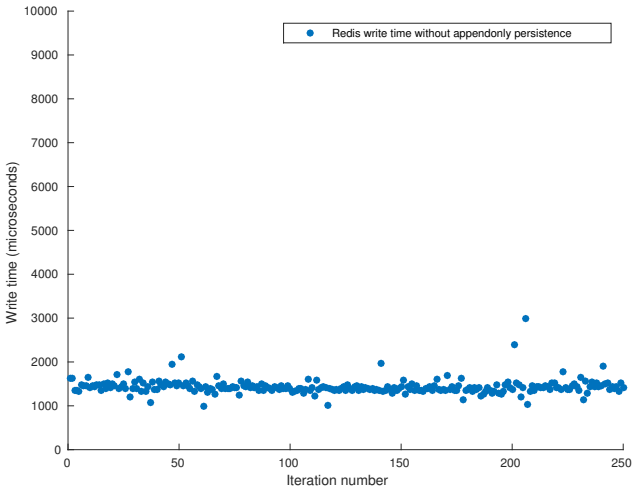
We have also investigated the effect of Redis persistence policies on the time taken for write operations to the database. From the results, we have identified that the fsync always policy leads to unacceptable latency, while the fysnc no and fsync everysec policy provide a reasonable level of durability and an acceptable response time. On further comparing the fysnc no and fynsc everysec policy, we notice that fysnc everysec is a better choice as it leads to a more predictable write time.

We have also demonstrated the autonomous scaling of MME workers. Our implementation uses the rate of incoming attach requests to determine when an MME worker has to be created or deleted. We notice that it takes at least 25 seconds for a new worker to start handling calls. Furthermore,

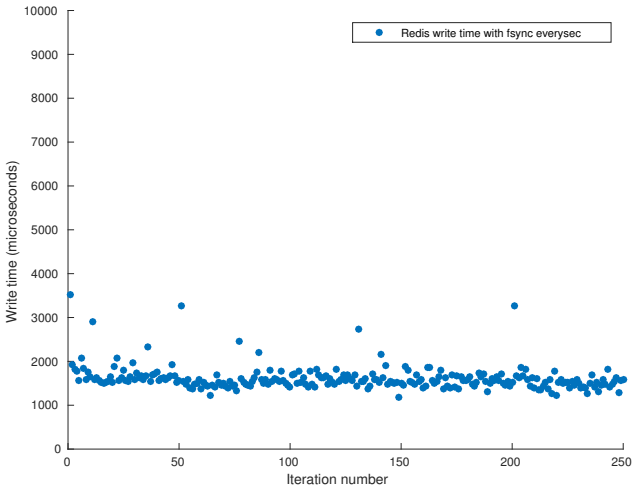
we identify that there should be a delay before an existing worker is deleted in order to allow on-going procedures to complete and the UE context to be stored on the state database.

Lastly, we have demonstrated the resilience of workers in a distributed architecture. Even if the worker responsible for a specific UE goes down, another worker is able to retrieve the UE context from the state database and process the call flow. Hence, the UE does not need to re-attach to the EPC network.

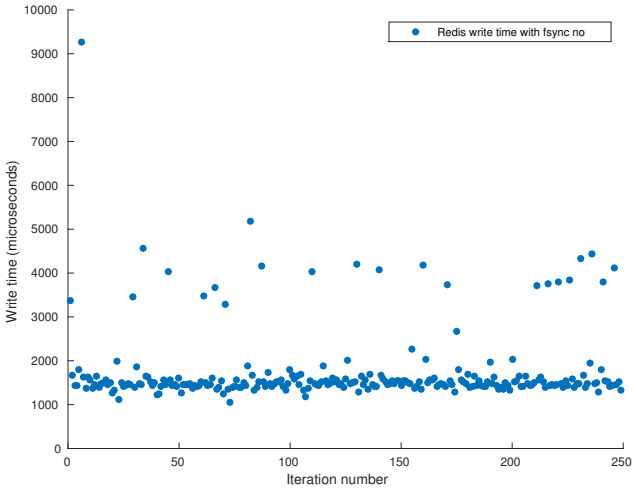
Chapter 6 now interprets the results presented in this chapter and also discusses methods to mitigate the negative effects of a distributed architecture.



(a) AOF disabled



(b) fsync everysec



(c) fsync no

Figure 5.3: Scatter plots of time taken to write UE context to Redis server (all master servers on one VM) with different persistence policies

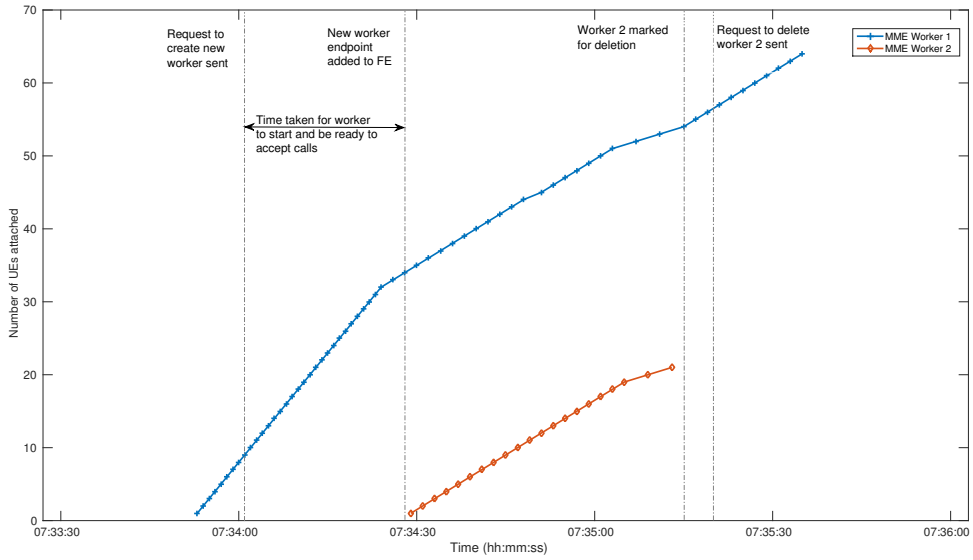


Figure 5.4: Number of attach requests sent to each worker against the time during which the experiment is run. Each point represents an attach request forwarded to a worker.

Chapter 6

Discussion

This chapter discusses some key aspects of virtualising elements in the EPC taking into consideration results presented in Chapter 5. Section 6.1 compares our MME with those presented in Section 2.6 and highlights methods for reducing the negative effects of a distributed design. Section 6.2 briefly describes the suitability of OpenStack for deploying VNFs that meet carrier-grade requirements. Finally, Section 6.3 discusses limitations of our testing and identifies improvements of the testing framework.

6.1 Evaluation of distributed design for MME

The MME presented in [58] is similar to our three-tier architecture. However, the authors have focused on scaling operations alone. Our results additionally provide a comparison with a non-distributed design for the MME and a demonstration of resilience. The distributed MME in [25, 26] consists of stateless message processors organised in a DHT with an external user state storage system. In this case, migration to another MME is possible only when the UE is in idle state. When the UE is in active state and attached to one MME, if a network event for that UE reaches another MME instance, the request has to be forwarded to the correct MME. In our design, the FE forwards the request to the correct MME and further redirections are not required. We believe additional redirections towards the correct MME instance can increase latency of EPC procedures.

The results presented in Section 5.2 indicate that the latency for procedures executed on the MME varies based on placement of the components of a distributed MME. When the components are on the same physical host, the average measured latency is lowest. This can be attributed to the networking setup on OpenStack, wherein communication between VMs co-located on

the same physical host occur locally and need not be sent to the networking host [37]. However, these local communications can increase CPU and memory overhead making the runtime effects hard to predict and possibly leading to performance degradation [37]. For VMs on different physical hosts, the latency increases with a more predictable value. However, in our implementation, all physical hosts reside on the same rack server. Hence it will be valuable to test the behaviour of the distributed MME with geographically distributed data centers.

With an intelligent orchestrator and placement algorithm, it is possible to deploy the components of a distributed MME taking into account different considerations such as latency, resilience, networking bandwidth and CPU interference. [36] proposes a new metric, *link gradient*, which measures the impact of changes in link latencies on end-to-end response times in distributed multi-tier architectures. This metric can be used to appropriately place components of the MME in order to achieve high availability while maintaining an acceptable response time. [52] highlights the importance of placement of VMs for performance-critical distributed applications. The authors describe an implementation of placement functions on OpenStack to optimise performance and maintain resilience for VNFs. [30] presents a model for the placement of VNFs in mobile core networks for optimised mobile core network topologies. The authors claim the same model can be used for decomposed network functions and more complex network service chains, which suits our requirement of optimised placement for a distributed MME. [29] analyses possible placements of VNFs in multiple data centers and proposes a placement model for virtualised SGWs and PGWs that minimises network load within a certain threshold for data plane latency.

Results from Section 5.4 provide an example of how configuration of the state database can effect attach latency. For a distributed design, the increased complexity means more careful design choices are required so as to meet performance requirements and reduce latency of operation. We have discussed alternate design options for each of the three tiers of the MME in Chapter 3.

In Section 5.5, the worker VM takes a long time to start and be ready to accept new procedures. Container-based virtualisation has been identified as a possible alternative to VMs with lower start times and better performance [44, 56]. While a VM runs a complete operating system, containers make modifications to the existing operating system to ensure isolation [44]. Thus containers are a lightweight alternative to VMs.

6.2 Suitability of OpenStack for telecommunication applications

OpenStack is a flexible and modular cloud platform which supports the deployment of NFV. The modular architecture of OpenStack enables service providers to choose the appropriate backend module depending on the type of service being deployed [12]. However, the suitability of OpenStack for telecommunication applications needs to be further investigated. [12] highlights a few extensions required by OpenStack before network services that meet carrier-grade performance requirements can be deployed over it. The identified extensions include assigning a dedicated number of CPU cores to a VNF (known as vCPU pinning) and guaranteed memory allocation [12, 44] so as to ensure a deterministic environment in which VMs are run. These features have been accepted for the Juno release of OpenStack, but have not been implemented yet[1]. OpenStack also supports baremetal machines using PXE boot[2]. These baremetal machines can provide better performance for applications which require high packet processing rates, deterministic performance and low latencies [12]. The complex network architecture of OpenStack may also result in performance variations or degradations. [33] investigates the performance of virtual networking under critical traffic conditions and identifies bottlenecks in the networking architecture of OpenStack.

6.3 Testing framework

Our testing of the MME is limited by the lack of an eNodeB emulator. With a suitable testing solution, the behaviour of the MME under load should be evaluated along with further testing for robustness and resilience in the case of VM failures. Despite these limitations, we were able to meet our aim of presenting a working solution of a three-tier architecture for a stateful network element.

Robust testing solutions are required to characterise the performance of VNFs and ensure carrier-grade performance. [66] proposes the use of an embedded instrumentation framework, which can collect and present data on real-time processing in virtualisation platforms. The authors highlight the importance of the framework to investigate performance of complex network services deployed within or across data centers. The data obtained from the

[1] `http://specs.openstack.org/openstack/nova-specs/specs/juno/index.html`

[2] `https://wiki.openstack.org/wiki/Ironic`

framework can help in optimising performance and diagnosis of VNFs. This data can also help identify the cause of outliers in latency obtained in Section 5.2. [39] presents a project for evaluating reliability of the infrastructure over which VNFs are deployed. This includes testing by injecting faults on the VMs as well as on the OpenStack platform. A comprehensive analysis of fault resilience in OpenStack is provided in [49]. Such a testing framework can be used to enhance the results presented in Section 5.6.

Chapter 7

Conclusion

This chapter first provides a summary of the work presented in this thesis. We then describe directions for future work in Section 7.2.

7.1 Conclusion

This thesis has described the application of NFV to mobile core networks. We have provided an overview of the benefits and challenges of virtualising the EPC and reviewed software architecture choices for developing VNFs. We chose the MME as a representative case to analyse the benefits of implementing a three-tier architecture for a control plane element in the EPC.

We have developed a working proof-of-concept implementation of a three-tier architecture for the MME. This new architecture enables us to decompose the MME into stateless workers with the UE state information saved in an external Redis data store. We use an intelligent front end to balance requests between the workers and to maintain interfaces to other network elements. This architecture allows the MME to scale easily based on incoming requests and also makes it resilient to worker VM failures. The workers being stateless can be independently started or stopped, thus enabling their scaling operations based on load in the network. Additionally, the workers require only a single configuration file, thereby simplifying the management of the MME. Once the UE context is stored on a Redis server, any worker can handle the processing of procedures for this UE after retrieving its context from the database. Thus, the failure of a worker will not affect end users if their context information can be retrieved. From the results presented in this thesis, we believe that the benefits of scalability and resilience outweigh the marginal increase in latency due to the distributed architecture. However, increased complexity of the architecture demands careful software design and

intelligent placement of the components so as not to degrade performance of the overall system.

As mobile network operators look to utilise distributed data centers to implement NFV for the core network, management and orchestration functions will play an important role. Distributed architectures and decomposition of network functions means that VM placement algorithms need to take into account end-to-end latency, reliability and performance requirements. Furthermore, the dynamic instantiation and removal of VMs results in changes in the network and management systems need to be able to handle these changes seamlessly. We have briefly discussed the suitability of OpenStack for the deployment of telecommunication applications. Further extensions to OpenStack are required before it is able to meet the performance requirements for telecommunication applications.

7.2 Future work

Our current implementation only supports a subset of the MME functionality. To further evaluate the effects of a distributed design, the addition of further features to the MME is required. For example, S1 handover and tracking area update procedures need to be implemented to analyse how the FE handles forwarding of messages in these procedures. Also, the HSS implementation needs to be changed from the existing co-located MySQL database. The behaviour of the MME under load has also not been verified. To achieve this, the eNodeB program needs to be modified to generate higher load on the MME. Additionally, the testing framework needs to be improved to include collection of network metrics and compute host load during the experiments. Lastly, the virtualisation platform could make use of Linux Containers instead of hypervisors. This may lead to improvements in boot times and performance of worker VMs.

Bibliography

[1] nwEPC - EPC SAE Gateway . http://sourceforge.net/projects/nwepc/. Online; Accessed on 02.06.2015.

[2] Project Clearwater. http://www.projectclearwater.org/. Online; Accessed on 02.07.2015.

[3] Redis Administration. http://redis.io/topics/admin. Online; Accessed on 18.06.2015.

[4] Redis Clients. http://redis.io/clients#C. Online; Accessed on 19.05.2015.

[5] Redis Cluster specification. http://redis.io/topics/cluster-spec. Online; Accessed on 19.05.2015.

[6] Redis persistence demystified. http://oldblog.antirez.com/post/redis-persistence-demystified.html. Online; Accessed on 19.06.2015.

[7] Redis Protocol specification. http://redis.io/topics/protocol. Online; Accessed on 15.05.2015.

[8] MCN D4.1, Mobile Network Cloud Component Design, European Commission, EU FP7 Mobile Cloud Networking public deliverable, November 2013.

[9] MCN D4.2, First Mobile Network Cloud Software Components, European Commission, EU FP7 Mobile Cloud Networking public deliverable, May 2014.

[10] Virtualizing network services - the telecom cloud. Tech. rep., Ericsson Review, March 2014. http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2014/er-telecom-cloud.pdf Online; Accessed on 02.07.2015.

[11] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019. White paper, Cisco, February 2015. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf`.

[12] OpenStack as the API framework for NFV: the benefits, and the extensions needed. Tech. rep., Ericsson Review, April 2015. `http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2015/er-openstack-api-nfv.pdf` Online; Accessed on 02.07.2015.

[13] 3GPP. LTE. `www.3gpp.org/technologies/keywords-acronyms/98-lte`. Online; Accessed on 13.04.2015.

[14] 3GPP. The Evolved Packet Core. `http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core`. Online; Accessed on 13.04.2015.

[15] 3GPP. 3GPP Evolved Packet System (EPS); Evolved General Packet Radio Service (GPRS) Tunnelling Protocol for Control plane (GTPv2-C); Stage 3 (Release 12). TS 29.274, 3rd Generation Partnership Project (3GPP), June 2014.

[16] 3GPP. Evolved Universal Terrestrial Radio Access Network (E-UTRAN); S1 Application Protocol (S1AP) (Release 12). TS 36.413, 3rd Generation Partnership Project (3GPP), December 2014.

[17] 3GPP. General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access (Release 12). TS 23.401, 3rd Generation Partnership Project (3GPP), December 2014.

[18] 3GPP. Network Architecture (Release 13). TS 23.002, 3rd Generation Partnership Project (3GPP), December 2014.

[19] 3GPP. Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3 (Release 13). TS 24.301, 3rd Generation Partnership Project (3GPP), December 2014.

[20] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2 (Release 12). TS 36.300, 3rd Generation Partnership Project (3GPP), March 2015.

[21] 3GPP. IP Multimedia Subsystem (IMS); Stage 2 (Release 13). TS 23.228, 3rd Generation Partnership Project (3GPP), 2015.

[22] Agyapong, P., Iwamura, M., Staehle, D., Kiess, W., and Benjebbour, A. Design considerations for a 5G network architecture. *Communications Magazine, IEEE 52*, 11 (2014), 65–75.

[23] Ahokas, K. Load balancing in LTE core network with OpenStack clouds: Design and implementation. Master's thesis, Aalto University School of Science, 2015. To be published.

[24] Amokrane, A., Ksentini, A., Hadjadj-Aoul, Y., and Taleb, T. Congestion control for machine type communications. In *Communications (ICC), 2012 IEEE International Conference on* (2012), IEEE, pp. 778–782.

[25] An, X., Pianese, F., Widjaja, I., and Acer, U. G. dMME: Virtualizing LTE mobility management. In *Local Computer Networks (LCN), 2011 IEEE 36th Conference on* (2011), IEEE, pp. 528–536.

[26] An, X., Pianese, F., Widjaja, I., and Günay Acer, U. DMME: a distributed LTE mobility management entity. *Bell Labs Technical Journal 17*, 2 (2012), 97–120.

[27] Andrews, J. G., Buzzi, S., Choi, W., Hanly, S. V., Lozano, A., Soong, A. C., and Zhang, J. C. What will 5G be? *Selected Areas in Communications, IEEE Journal on 32*, 6 (2014), 1065–1082.

[28] Baldoni, R., and Raynal, M. Fundamentals of distributed computing: A practical tour of vector clock systems. *IEEE Distributed Systems Online 3*, 2 (2002), 12.

[29] Basta, A., Kellerer, W., Hoffmann, M., Morper, H. J., and Hoffmann, K. Applying NFV and SDN to LTE mobile core gateways, the functions placement problem. In *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges* (2014), ACM, pp. 33–38.

[30] Baumgartner, A., Reddy, V. S., and Bauschert, T. Mobile core network virtualization: A model for combined virtual core network function placement and topology optimization. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on* (2015), IEEE, pp. 1–9.

[31] Bosch, P., Duminuco, A., Pianese, F., and Wood, T. L. Telco clouds and virtual telco: Consolidation, convergence, and beyond. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on* (2011), IEEE, pp. 982–988.

[32] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and Arkko, J. Diameter Base Protocol. RFC 3588, RFC Editor, September 2003. `http://www.rfc-editor.org/rfc/rfc3588.txt`.

[33] Callegati, F., Cerroni, W., Contoli, C., and Santandrea, G. Performance of Network Virtualization in cloud computing infrastructures: The OpenStack case. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on* (2014), IEEE, pp. 132–137.

[34] Carella, G., Corici, M., Crosta, P., Comi, P., Bohnert, T. M., Corici, A. A., Vingarzan, D., and Magedanz, T. Cloudified IP Multimedia Subsystem (IMS) for Network Function Virtualization (NFV)-based architectures. In *Computers and Communication (ISCC), 2014 IEEE Symposium on* (2014), IEEE, pp. 1–6.

[35] Chang, F., Fales, P. S., Steiner, M., Viswanathan, R., Williams, T. J., and Wood, T. L. Mitigating High Latency Outliers for Cloud-Based Telecommunication Services. *Bell Labs Technical Journal 17*, 2 (2012), 121–142.

[36] Chen, S., Joshi, K. R., Hiltunen, M. A., Schlichting, R. D., and Sanders, W. H. Using link gradients to predict the impact of network latency on multitier applications. *IEEE/ACM Transactions on Networking (TON) 19*, 3 (2011), 855–868.

[37] Corradi, A., Fanelli, M., and Foschini, L. Vm consolidation: A real case based on openstack cloud. *Future Generation Computer Systems 32* (2014), 118–127.

[38] Costa-Requena, J., Santos, J., Guasch, V., Ahokas, K., Premsankar, G., and Luukkainen, S. SDN and NFV Integration in Generalized Mobile Network Architecture. In *European Conference on Networks and Communications (EuCNC)* (2015). To appear.

[39] Cotroneo, D., De Simone, L., Iannillo, A., Lanzaro, A., Natella, R., Fan, J., and Ping, W. Network function virtualization: Challenges and directions for reliability assurance. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on* (2014), IEEE, pp. 37–42.

[40] Dodig-Crnkovic, G. Scientific methods in computer science. In *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia* (2002), pp. 126–130.

[41] ETSI. Network Functions Virtualisation - An Introduction, Benefits, Enablers, Challenges, Call for Action. Introductory white paper, ETSI, October 2012. `http://portal.etsi.org/NFV/NFV_White_Paper.pdf`.

[42] ETSI. Network Functions Virtualisation (NFV); Use Cases V1.1.1. GS NFV 001, ETSI, 2013.

[43] ETSI. Network Functions Virtualisation (NFV); Architectural Framework V1.2.1. GS NFV 002, ETSI, 2014.

[44] Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. An updated performance comparison of virtual machines and linux containers. *technology 28* (2014), 32.

[45] Guasch, V. F. LTE network Virtualisation. Master's thesis, Aalto University School of Electrical Engineering, October 2013.

[46] Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. Network function virtualization: Challenges and opportunities for innovations. *Communications Magazine, IEEE 53*, 2 (2015), 90–97.

[47] Hawilo, H., Shami, A., Mirahmadi, M., and Asal, R. NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC). *Network, IEEE 28*, 6 (2014), 18–26.

[48] Janssens, N., An, X., Daenen, K., and Forlivesi, C. Dynamic scaling of call-stateful SIP services in the cloud. In *NETWORKING 2012*. Springer, 2012, pp. 175–189.

[49] Ju, X., Soares, L., Shin, K. G., Ryu, K. D., and Da Silva, D. On fault resilience of openstack. In *Proceedings of the 4th annual Symposium on Cloud Computing* (2013), ACM, p. 2.

[50] Matuszewski, M., and Garcia-Martin, M. A. A distributed IP multimedia subsystem (IMS). In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a* (2007), IEEE, pp. 1–8.

[51] Nishimura, H., Iwasa, E., Irie, M., Kondoh, S., Kaneko, M., Fukumoto, T., Iio, M., and Ueda, K. Applying flexibility in scale-out-based web cloud to future telecommunication session control systems. In *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on* (2012), IEEE, pp. 1–7.

[52] Oechsner, S., and Ripke, A. Flexible support of VNF placement functions in OpenStack. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on* (2015), IEEE, pp. 1–6.

[53] Penttinen, J. T. *The Telecommunications Handbook: Engineering Guidelines for Fixed, Mobile and Satellite Systems*. John Wiley & Sons, 2015.

[54] Postel, J. User Datagram Protocol. STD 6, RFC Editor, August 1980. `http://www.rfc-editor.org/rfc/rfc768.txt`.

[55] Raynal, M., and Singhal, M. Logical time: Capturing causality in distributed systems. *Computer 29*, 2 (1996), 49–56.

[56] Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., and Peterson, L. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *ACM SIGOPS Operating Systems Review* (2007), vol. 41, ACM, pp. 275–287.

[57] Stewart, R. Stream Control Transmission Protocol. RFC 4960, RFC Editor, September 2007. `http://www.rfc-editor.org/rfc/rfc4960.txt`.

[58] Takano, Y., Khan, A., Tamura, M., Iwashina, S., and Shimizu, T. Virtualization-Based Scaling Methods for Stateful Cellular Network Nodes using Elastic Core Architecture. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on* (2014), IEEE, pp. 204–209.

[59] Taleb, T. Toward carrier cloud: Potential, challenges, and solutions. *Wireless Communications, IEEE 21*, 3 (2014), 80–91.

[60] Taleb, T., Corici, M., Parada, C., Jamakovic, A., Ruffino, S., Karagiannis, G., and Magedanz, T. EASE: EPC as a service to ease mobile core network deployment over cloud. *Network, IEEE 29*, 2 (2015), 78–88.

[61] Taleb, T., Ksentini, A., and Kobbane, A. Lightweight mobile core networks for machine type communications. *Access, IEEE 2* (2014), 1128–1137.

[62] Taleb, T., and Kunz, A. Machine type communications in 3gpp networks: potential, challenges, and solutions. *Communications Magazine, IEEE 50*, 3 (2012), 178–184.

[63] Taleb, T., and Samdanis, K. Ensuring service resilience in the EPS: MME failure restoration case. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE* (2011), IEEE, pp. 1–5.

[64] Tamura, M., Nakamura, T., Yamazaki, T., and Moritani, Y. A study to achieve high reliability and availability on core networks with network virtualization. Technical journal, NTT Docomo, July 2013. Online; Accessed on 02.07.2015.

[65] Tolonen, A. Dynamic virtualized network functions on an openstack cloud. Master's thesis, Aalto University School of Science, September 2014.

[66] Veitch, P., McGrath, M. J., and Bayon, V. An instrumentation and analytics framework for optimal and robust nfv deployment. *Communications Magazine, IEEE 53*, 2 (2015), 126–133.

[67] Wilder, B. *Cloud Architecture Patterns*. O'Reilly Media, Inc., 2012.

[68] Yamasaki, T., Khan, A., Tamura, M., Shimizu, T., and Iwashina, S. A database access scheme for elastic-core architecture. In *Wireless and Mobile, 2014 IEEE Asia Pacific Conference on* (2014), IEEE, pp. 138–143.

[69] Zhiqun, X., Duan, C., Zhiyuan, H., and Qunying, S. Emerging of telco cloud. *Communications, China 10*, 6 (2013), 79–85.